Frank Bomarius
Hajimu Iida (Eds.)

Product Focused Software Process Improvement

LNCS 3009

5th International Conference, PROFES 2004
Kansai Science City, Japan, April 2004
Proceedings

Springer

Lecture Notes in Computer Science 3009

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Frank Bomarius Hajimu Iida (Eds.)

Product Focused Software Process Improvement

5th International Conference, PROFES 2004
Kansai Science City, Japan, April 5-8, 2004
Proceedings

Springer

eBook ISBN: 3-540-24659-2
Print ISBN: 3-540-21421-6

Created in the United States of America

Visit Springer's eBookstore at: http://ebooks.springerlink.com
and the Springer Global Website Online at: http://www.springeronline.com

Preface

On behalf of the PROFES organizing committee we are proud to present to you the proceedings of the 5th International Conference on Product Focused Software Process Improvement (PROFES 2004), held in Kansai Science City, Japan.

Since 1999, PROFES has established itself as one of the recognized international process improvement conferences. In 2004 the conference left Europe for the first time and moved to Japan. Japan and its neighboring countries are intensifying their efforts to improve software engineering excellence, so it was a logical step to select Japan as the venue for PROFES 2004.

The purpose of the conference is to bring to light the most recent findings and results in the area and to stimulate discussion between researchers, experienced professionals, and technology providers. The large number of participants coming from industry confirms that the conference provides a variety of up-to-date topics and tackles industry problems. The main theme of PROFES is professional software process improvement (SPI) motivated by product and service quality needs. SPI is facilitated by software process assessment, software measurement, process modeling, and technology transfer. It has become a practical tool for quality software engineering and management. The conference addresses both the solutions found in practice and the relevant research results from academia. This is reflected in the 41 full papers, which are a balanced mix of academic papers as well as industrial experience reports.

The business of developing new applications like mobile and Internet services or enhancing the functionality of a variety of products using embedded software is maturing and meeting the harsh business realities. The necessity for professional software development, quality, and cost effectiveness is becoming evident and there is a need to spread SPI beyond its traditional areas. Some of the accepted papers focus especially on the latest activities in Japanese software engineering, which is facing new challenges in developing new types of software in new ways, such as mobile networking embedded software, in ever-shorter times.

We wish to thank the Nara Institute of Science and Technology (NAIST), the Fraunhofer IESE, the University of Osaka, and VTT Electronics for supporting the conference. We are also grateful to the authors for high-quality papers, the program committee for their hard work in reviewing the papers, the organizing committee for making the event possible, and all the numerous supporters, including the Software Engineers Association of Japan, who helped in organizing this conference.

Last, but not least, many thanks to Patrick Leibbrand at Fraunhofer IESE for copyediting this volume, Dr. Masahide Nakamura at NAIST for developing the PROFES 2004 web pages, and Gaby Klein at IESE and Junko Inui at NAIST for helping in the organization of this conference.

January 2004

Frank Bomarius
Hajimu Iida

This page intentionally left blank

Conference Organization

General Chair

Seija Komi-Sirvio, VTT Electronics (Finland)

Organizing Chair

Ken'ichi Matsumoto, Nara Institute of Science and Technology (Japan)

Program Co-chairs

Frank Bomarius, Fraunhofer IESE (Germany)
Hajimu Iida, Nara Institute of Science and Technology (Japan)

Tutorial, Workshop, Panel Chair

Shinji Kusumoto, Osaka University (Japan)

Publicity Chairs

Europe	Pekka Abrahamsson, VTT Electronics (Finland)
USA	Ioana Rus, Fraunhofer Center, Maryland
Japan	Yasunari Takagi, OMRON
	Takeshi Hayama, NTT Data

PROFES Advisor

Markku Oivo, University of Oulu (Finland)

Program Committee

Andreas Birk, SD&M (Germany)
Reidar Conradi, NTNU (Norway)
Paolo Donzelli, University of Maryland, College Park (USA)
Ilkka Haikala, Tampere University of Technology (Finland)
Tua Huomo, Cybelius Software (Finland)
Katsuro Inoue, University of Osaka (Japan)
Janne Jarvinen, Solid Information Technology (Finland)
Ross Jeffery, University of New South Wales (Australia)
Erik Johansson, Q-Labs (Sweden)
Natalia Juristo, Universidad Politecnica de Madrid (Spain)
Haruhiko Kaiya, Shinshu University (Japan)
Kari Kansala, Nokia Research Center (Finland)
Toshihiro Komiyama, NEC (Japan)
Jaana Kuula, Lapinliitto (Finland)
Pasi Kuvaja, University of Oulu (Finland)
Mikael Lindval, Fraunhofer Center, Maryland (USA)
Makoto Matsushita, Osaka University (Japan)
Kenichi Matumoto, NAIST (Japan)
Maurizio Morisio, University of Turin (Italy)
Kumiyo Nakakoji, University of Tokyo (Japan)
Paolo Nesi, University of Florence (Italy)
Risto Nevalainen, STTF (Finland)
Hideto Ogasawara, Toshiba (Japan)
Markku Oivo, University of Oulu (Finland)
Paivi Parviainen, VTT Electronics (Finland)
Teade Punter, Fraunhofer IESE (Germany)
Karl Reed, La Trobe University (Australia)
Harri Reiman, Ericsson (Sweden)
Günther Ruhe, University of Calgary (Canada)
Iona Rus, Fraunhofer Center, Maryland (USA)
Kurt Schneider, University of Hannover (Germany)
Carolyn Seaman, UMBC, Baltimore (USA)
Veikko Seppanen, University of Oulu (Finland)
Forrest Shull, Fraunhofer Center, Maryland (USA)
Dag Sjoeberg, University of Oslo (Norway)
Reijo Sulonen, Helsinki University of Technology (Finland)
Rini van Solingen, CMG (The Netherlands)
Matias Vierimaa, VTT Electronics (Finland)
Otto Vinter, DELTA (Denmark)
Giuseppe Visaggio, University of Bari (Italy)
Yingxu Wang, University of Calgary (Canada)
Hironori Washizaki, Waseda University (Japan)
Isabella Wieczorek, Federal Ministry of Research and Education (Germany)
Claes Wohlin, Blekinge Institute of Technology (Sweden)

We would also like to thank the following people who helped in reviewing the papers: Silvia T. Acuña, Oscar Dieste, Christian Bunse, Klaus Schmid, Alf Inge Wang, Ralf Kalmar, Davide Rogai, Pierfrancesco Bellini, Ivan Bruno, Maria Teresa Baldassarre, Danilo Caivano, Aderemi Adewumi, Nguyen Cong Vu, Stein Grimstad, Erik Arisholm, June Verner, Felicia Kurniawati, Barbara Kitchenham, and Ming Huo.

This page intentionally left blank

Table of Contents

Software Process Improvement

Software Quality

Measurement

Methods and Tools

Experimental Software Engineering

Industrial Experiences

Agile Methods

Software Process Assessment

Requirements Engineering

Software Reuse / COTS

A Model for the Implementation of Software Process Improvement: An Empirical Study

Mahmood Niazi, David Wilson, Didar Zowghi, and Bernard Wong

Faculty of Information Technology, University of Technology Sydney
{mkniazi, davidw, didar, bernard}@it.uts.edu.au

Abstract. Advances have been made in the development of software process improvement (SPI) standards and models, i.e. Capability Maturity Model (CMM), more recently CMMI, and ISO's SPICE. However, these advances have not been matched by equal advances in the adoption of these standards and models in software development which has resulted in limited success for many SPI efforts. The current problem with SPI is not a lack of a standard or model, but rather a lack of an effective strategy to successfully implement these standards or models. In the literature, much attention has been paid to "what activities to implement" rather than "how to implement" these activities. We believe that identification of only "what activities to implement" is not sufficient and that knowledge of "how to implement" is also required for successful implementation of SPI programmes.

The aim of this research paper is to empirically explore the viewpoints and experiences of practitioners regarding SPI implementation and based on the findings to develop a model in order to guide practitioners in effectively implementing SPI programmes. This SPI implementation model has six phases and provides a very practical structure within which to implement SPI programmes in organizations.

1 Introduction

Improving the quality of software process is a key information systems issue. This is due to the role software systems play in modern-day business and, to some extent, modern-day living. The issue of software quality has been brought into sharp focus due to the disappointing performance of some high profile software projects, e.g. One.Tel [1], the London Ambulance Service [2] and Explosion of the Ariane 5 [3]. These unsatisfactory results have led to some very costly commercial disasters. The search for solutions to improve software quality has continued for many years and software organizations are now realizing that one of their fundamental problems is the inability to effectively manage the software process [4; 5]. In order to address the effective management of software process different approaches have been developed, of which software process improvement (SPI) is the one most widely used.

SPI models such as the Capability Maturity Model (CMM) [6], more recently CMMI [7], and standards such as ISO's SPICE [8] focus on processes to produce quality software. Research shows that the effort put into these models and standards can assist in producing high quality software, reducing cost and time, and increasing productivity [4; 5]. However, little attention has been paid to the effective

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 1-16, 2004.

implementation of these models and standards [9] which has resulted in limited success for many SPI efforts. Studies show that 67% of SPI managers want guidance on how to implement SPI activities, rather than what SPI activities to actually implement [10]. However, despite the importance of the SPI implementation process, little empirical research has been carried out on developing ways to effectively implement SPI programmes.

The aim of this research paper is to empirically explore the viewpoints and experiences of practitioners regarding SPI implementation and based on the findings to develop a model in order to guide practitioners in effectively implementing SPI programmes. In order to design this SPI implementation model (SPI-IM) we have extended the concept of critical success factors (CSFs) [11].

We have analysed the experiences, opinions and views of practitioners regarding SPI implementation through the literature (i.e. case studies, technical reports, experience reports and journal's articles as shown in Appendix A). Furthermore, we have carried out an empirical study about factors that have a positive or negative impact on the implementation of a SPI program. We have conducted 31 CSF interviews with Australian practitioners in different organizations with the specific aim of:

- Establishing what their typical SPI implementation experiences are
- Identifying their major concerns about SPI implementation
- Exploring different phases/steps necessary for the implementation of SPI programmes

There are five research questions that have motivated the work reported in this paper:
RQ1. What factors, as identified in the literature, have a positive impact on implementing SPI?
RQ2. What factors, as identified in the empirical study, have a positive impact on implementing SPI?
RQ3. What factors, as identified in the literature, have a negative impact on implementing SPI?
RQ4. What factors, as identified in the empirical study, have a negative impact on implementing SPI?
RQ5. What are the necessary phases/steps for the implementation of SPI programmes?

This paper is organised as follows. Section 2 describes the background. Section 3 describes study design. Findings are described in Section 4. In Section 5 a model for SPI implementation is described in detail. Section 6 provides the conclusion and future work.

2 Motivation

A number of empirical studies have investigated factors that positively or negatively impact SPI, e.g. [9; 12-15]. To highlight few of these: in the survey of 138 individuals in 56 software organizations, Goldenson and Herbsleb [9], identified the factors necessary for implementing a successful SPI programme. Stelzer and Werner [12] determined ten factors that affect organizational change in SPI. Rainer and Hall [14] have conducted a questionnaire survey of UK companies and identified the key success factors that can impact on SPI implementation.

In other studies different researchers have described their experiences of SPI implementation (for complete references see Appendix A).

The work we report in this paper extends work previously done in the above studies. Many of these studies have adopted the questionnaire survey method for the identification of factors, despite the fact that Rockart [11] proposed CSF interviews as the data gathering approach. A disadvantage of the questionnaire survey method is that respondents are provided with a list of possible factors and asked to select from that list. This tends to pre-empt the factors investigated and to limit them to those reported in existing studies - respondents only focus on the factors provided in the list. It is also possible that respondents may misinterpret the factors provided in the questionnaire. These problems are exacerbated if the respondents do not have a sufficient concept of the CSF approach and if they lack sufficient details about the factors provided in the list. In order to provide more confidence in the study it is important that practitioners' experiences and perceptions should be explored independently and without any suggestion from the researcher.

In the last few years we have seen technical quality initiatives such as CASE tools and organizational initiatives such as CMM [6] and more recently CMMI [7]. These initiatives aim to improve software processes. It has been suggested that whether a quality initiative is technical or organizational, ineffective implementation of these initiatives can significantly affect the success of SPI efforts [16-18]. We have focused on this issue and designed a model that provides a very practical structure within which to implement SPI programmes. The basis of this model is what we have studied in the literature as well as the findings from our empirical study of SPI in practice

3 Study Design

3.1 The Companies

From November 2002 to August 2003 the first author visited 26 software companies and conducted 31 interviews. All of the 26 companies responded to a request for participants, which was posted via the email. The target population in this research was those software-producing companies that have initiated SPI programmes. Although we do not claim this is a statistically representative sample, Appendix B does show that companies in the study range from a very small software house to very large multinational companies and cover a wide range of application areas. It is further important to acknowledge that the data was collected from companies who were tackling real SPI implementation issues on a daily basis; therefore we have high confidence in the accuracy and validity of data.

Thirty-one practitioners voluntarily participated in this study. By volunteering to participate they have become a self-selecting sample. Self-sampling as opposed to random sampling though more practical is often prone to bias [19]. In this research because the sample of companies form an original self-selected group (that is software producing companies), it is important to ensure that one particular group is not over represented [20]. This research addresses the issue of over representation by using a sample of companies of varying complexities, size, nature of business, type of applications etc. A similar approach has been used by other researchers [21; 22].

It is further important to acknowledge that the practitioners sampled within companies are representative of practitioners in organisations as a whole. A truly representative sample is impossible to attain and the researcher should try to remove as much of the sample bias as possible [20]. In this research, in order to make the sample fairly representative of SPI practitioners in particular organization, 31 practitioners from 26 organisation self-selected to participate. The sample of practitioners involved in this research includes developers, business analysts, methodology analyst, technical directors, project managers and senior management.

In addition to our empirical study we have also analysed 50 published experience reports, case studies and articles in order to identify factors that can play a positive or negative role in the implementation of SPI programmes. The studies we have analysed appeared to be of well-known organizations (Appendix A). We consider these to be important publications because the 34 organizations include all the five organizations that have been awarded the IEEE Computer Society Award for Process Achievement.

3.2 Data Collection Methods

SPI literature. We undertook an objective reading and identified a number of SPI implementation factors. SPI literature consists of case studies, experience reports and high-level software process texts. Most of the studies describe real life experiences of SPI implementation and provide different factors that play a positive or negative role in SPI implementation. This literature analysis was entirely objective and only one researcher was involved. According to Leedy and Ormrod [23] if the judgement of the literature is entirely objective then one person is sufficient.

CSFs interviews. The CSF interview [11] is a unique opportunity to assist the managers in better understanding their information needs. “The CSF interview often presents the initial occasion to interact with the manager on the types of information support that might be useful to her” [24].

CFS interviews were conducted with three groups of practitioners:

- The first group was made up of designers/testers/programmer/analyst. Referred to as “developers”.
- The second group was made up of team leaders/project managers. Referred to as “managers”.
- The third group was made up of senior managers/directors. Referred to as “senior managers”.

Questioning was both open and close-ended with frequent probing to elaborate and clarify meaning. The negotiated interview duration was half an hour, however, the researcher and interviewee would determine the pace of the interview.

3.3 Data Analysis Methods

Content Analysis. This research seeks to identify perceptions and experiences of practitioners about SPI implementation. In order to identify common themes for the

implementation of SPI programmes, the following process has been adopted in this research [25; 26]:

- Identifying themes for SPI implementation from transcripts: All the interview transcripts were read to identify the major themes for SPI implementation. These themes were noted down and compared to the notes made during the CSF interviews in order to reassure that the transcripts being analysed are indeed a true reflection of the discussion in the CSF interviews. These two process steps also verify that the transcription process has not changed the original data generated in the CSF interviews.
- Generate categories: All the CSF interview transcripts were read again to generate categories for responses. Different themes were grouped together under three categories, i.e. CSF, critical barrier and phases/steps needed for SPI implementation. For example, budget, funds etc were grouped together under CSF category "resources". Poor response, user unwilling to be involved etc were grouped together under critical barrier category "lack of support". Each category represents a CSF, critical barrier and steps/phases needed for the implementation of SPI programme.

Frequency analysis. According to Seaman [27] coding in empirical research is one method of extracting quantitative data from qualitative data in order to perform some statistical analysis. In this research, data from the literature and CSF interviews is categorised and coded in order to perform frequency analysis and also to perform some comparative analysis of SPI implementation factors. We have used frequency analysis at two levels. Firstly, we measured the occurrence of key factors in a survey of the literature. We recorded the occurrence of a key factor in each article. By comparing the occurrences of a key factor in a number of articles against occurrences of other key factors in the same articles, we calculated the relative importance of each factor. For example, if a factor is mentioned in 10 out of 20 articles, it has an importance of 50% for comparison purposes. In this way we compared and ranked the factors. Secondly, we measured the occurrence of key factors in an empirical study. In order to analyse the CSF interview transcripts we recorded the occurrence of a key factor in each CSF interview transcript. By comparing the occurrences of a key factor in a number of CSF interview transcripts against occurrence of other key factors in the same CSF interview transcript, we calculated the relative importance of each factor. Finally, conclusions are drawn regarding the factors that are critical in the literature and in the empirical study.

4 Findings

In this section we discuss the results relating to RQ1, RQ2, RQ3 and RQ4. This section shows the CSFs and critical barriers cited in the literature and empirical study and the frequency with which they occurred. The percentage shows the proportion of literature and practitioners that cited a particular factor.

Table 1. CSFs identified through literature and empirical study

Success Factors	Occurrence in literature (n=47)			Occurrence in CSF interviews (n=31)		
	Freq	%	Rank	Freq	%	Rank
Assignment of responsibility of SPI	12	26	**7**	-	-	-
Clear and relevant SPI goals	12	26	**7**	-	-	-
Company culture	-	-	-	2	7	**12**
Creating process action teams/external agents	15	31	**5**	2	7	**12**
Customer satisfaction	-	-	-	2	7	**12**
Encouraging communication and collaboration	10	21	**9**	5	16	**8**
Experienced staff	13	28	**6**	13	42	**5**
Facilitation	-	-	-	9	29	**7**
Formal code review	-	-	-	3	10	**10**
Formal documentation	-	-	-	3	10	**10**
Formal methodology	-	-	-	11	36	**6**
Formalised relationship between development team	-	-	-	1	3	
Internal leadership	6	13	**11**	-	-	-
Logical sequence/order of SPI implementation	-	-	-	2	7	**12**
Managing the SPI project	7	15	**10**	4	13	**9**
Measurement	-	-	-	2	7	**11**
Process ownership	11	23	**8**	-	-	-
Providing enhanced understanding	7	15	**10**	-	-	-
Quality assurance	-	-	-	5	16	**8**
Reviews	13	28	**6**	2	9	**11**
Reward schemes	7	15	**10**	-	-	-
Senior management commitment	31	66	**1**	20	65	**2**
SPI Awareness	-	-	-	18	58	**3**
SPI people highly/well respected	5	11	**12**	-	-	-
Staff involvement	24	51	**2**	10	36	**6**
Staff time and resources	18	38	**4**	14	45	**4**
Standards and procedures	4	9	**13**	-	-	-
Tailoring improvement initiatives	7	15	**10**	2	7	**12**
Training and mentoring	23	49	**3**	22	71	**1**
Tools/packages	-	-	-	1	3	**13**

4.1 CSFs Identified through Literature and an Empirical Study

In order to answer RQ1 and RQ2, Table 1 shows the list of CSFs cited in the literature and an empirical study.

The most frequently cited factor in the literature is senior management commitment, i.e. 66%. This suggests that in practitioners' opinion sponsorship can play a vital role in the implementation of SPI programs. Other frequently cited factors in the literature are staff involvement (51%), and training and mentoring (49%). It shows that practitioners consider their involvement, and training and mentoring

imperative for the successful implementation of SPI programs. The results also show that staff time and resources and creating process action teams are also important factors. A quarter of the literature cited reviews, experienced staff, clear and relevant SPI goals and assigning of responsibilities as CSFs.

Table 1 shows that, like the literature, the most frequently cited factors in the CSF interviews are training and senior management commitment, i.e. cited 71% and 65% respectively. Two new CSFs – awareness and formal methodology – have been identified in our empirical study which have not been identified in the literature. Other frequently cited factors are resources, staff involvement and experienced staff. Other factors are less frequently cited in the CSF interviews.

Table 2. Critical Barriers identified through literature and an empirical study

Barriers	**Occurrence in literature (n=14)**			**Occurrence in interviews (n=31)**		
	Freq	%	Rank	Freq	%	Rank
Changing the mindset of management and technical staff	2	14	**5**	-	-	**-**
Inertia	1	7	**6**	-	-	**-**
Inexperienced staff/lack of knowledge	5	36	**2**	7	23	**4**
Lack of awareness	-	-	**-**	11	36	**3**
Lack of communication	-	-	**-**	2	7	**7**
Lack of formal methodology	-	-	**-**	12	39	**2**
Lack of project management	-	-	**-**	3	10	**6**
Lack of resources	7	50	**1**	11	36	**3**
Lack of sponsorship	-	-	**-**	6	19	**5**
Lack of support	3	21	**4**	14	45	**1**
Lack of tools	-	-	**-**	1	3	**8**
Lack of training	-	-	**-**	3	10	**6**
Negative/Bad experience	1	7	**6**	2	7	**7**
Organizational politics	4	29	**3**	14	45	**1**
Paperwork required/formal procedures	1	7	**6**	7	23	**4**
SPI gets in the way of real work	4	29	**3**	2	7	**7**
Staff turnover	4	29	**3**	-	-	**-**
Time pressure	5	36	**2**	6	19	**5**

4.2 Critical Barriers Identified through Literature and an Empirical Study

Our aim of identifying critical barriers [9; 28] is to understand the nature of issues that undermine the SPI implementation programmes. In order to answer RQ3 and RQ4, Table 2 shows the list of critical barriers cited in the literature and an empirical study. The results show that most of the practitioners in literature consider lack of resources a major critical barrier for the implementation of SPI. The results also suggest that in practitioners' opinion time pressure and inexperienced staff can undermine the success of SPI implementation programs. It shows that practitioners prefer to avoid organizational politics during the implementation of SPI programs.

Organizational politics and lack of support are on the top in CSF interviews, i.e. equal at 45%. Two new critical barriers – lack of formal methodology and lack of awareness – have been identified in our empirical study which have not been

identified in the literature. The critical barriers lack of resources is cited 36% in the CSF interviews. Other barriers are less frequently cited in the CSF interviews.

5 A Model for the Implementation of SPI Programmes

We have used the findings from the literature and an empirical study to develop a SPI-IM (as shown in Figure 1) in order to guide organizations to successfully implement SPI programmes. The empirical study has led us to design different phases for SPI implementation. We have selected those phases in our SPI-IM, which were frequently cited by practitioners. We have also identified CSFs and critical barriers for SPI implementation (Table 1 and Table 2). We have divided these factors among different phases of the SPI-IM. In order to have more fine-grained activities within each phase of our SPI-IM, we have designed a list of practices for each CSF and critical barrier. The SPI-IM in Figure 1 shows that organizations should address each factor in order to successfully implement each phase of the model.

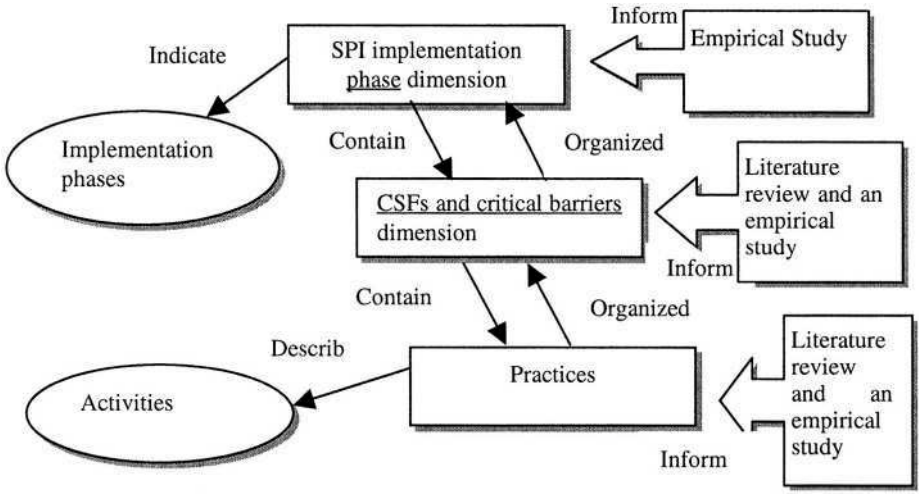


Fig. 1. SPI implementation model

The structure of our SPI-IM is built upon the following dimensions:

- SPI implementation phase dimension
- SPI implementation CSF and critical barrier dimension

We have high confidence that these 2 dimensions of SPI-IM can guide SPI practitioners in effectively implement SPI initiatives. This is because we have collected data of SPI practitioners who were dealing with SPI implementation issues on a daily basis. Furthermore, practitioners' experiences and perceptions were explored independently and without any suggestion from the researcher.

5.1 SPI Implementation Phase Dimension

Using the content analysis of the recorded interviews, we have identified six phases for the implementation of SPI programmes. In this section we briefly describe each phase in turn and in the appropriate sequence. Figure 2 shows SPI implementation phase dimension.

Awareness. Practitioners felt the need for awareness of SPI programmes in order to fully understand the benefits of SPI. Practitioners said that, as SPI implementation is the process of adoption of new practices in the organization, it is very important to promote awareness activities of SPI and to share knowledge among different practitioners. Practitioners suggested involving all the staff members in these awareness programmes.

Awareness has emerged as a first phase for the implementation of SPI programmes. This is because SPI is an expensive and long-term approach and it takes a long time to realise the real benefits of this approach. Hence, in order to get support of management and practitioners and to successfully continue SPI initiatives it is very important to provide sufficient awareness at the very beginning of SPI implementation programmes. SPI implementation is not as beneficial without sufficient awareness of its benefits. Moitra [29] has emphasised explanation and sharing of "how the improved processes will help the individuals in terms of their efficiency, productivity and performance". The necessary investment of time and money and the need to overcome staff resistance are a potential barrier to SPI implementation [12]. These obstacles cannot be overcome without sufficient SPI awareness within the organization.

Learning. Learning appears as an important factor for SPI implementation success. For learning, practitioners emphasized training in SPI skills in order to achieve mastery of its use. This phase involves equipping the practitioners with the knowledge of the critical technologies which are required for SPI.

Different studies have confirmed training as an important source of learning for the implementation of SPI programmes [4; 30; 31]. Learning comprises acquiring and transferring knowledge of SPI activities. Managers and employees usually have a general idea of the software process but they do not have complete understanding of the necessary details and also they do not understand how their work adds to the organization mission and vision [32]. SPI can only be successfully implemented if staff members have enough understanding, guidance and knowledge of all the SPI activities [12; 33; 34].

Pilot implementation. Practitioners advised to first implement SPI programs at a low level and see how successful it is within a particular department. This is also important for practitioners in order to judge their SPI skills in the pilot implementation. This is the phase where practitioners can decide how much resources, training and commitment is required in order to implement SPI practices across the organization.

Our results have confirmed the results of [33; 34] where they recommend "start small and slow" for real quality improvement.

SPI implementation action plan. Practitioners stressed the need for proper planning and management. They said after pilot implementation a proper plan with activities, schedule, allocated resources, responsibilities, budget and milestone should be designed. This plan should be based on the results and experiences of the pilot implementation.

SPI implementation without planning and project management leads to chaotic practices. Different studies emphasised managing the SPI project [12; 30; 31]. Often, the improvement projects have no specified requirements, project plan, or schedule [12]. It was recommended by the practitioners to treat SPI as a real project and it must be managed just as any other project.

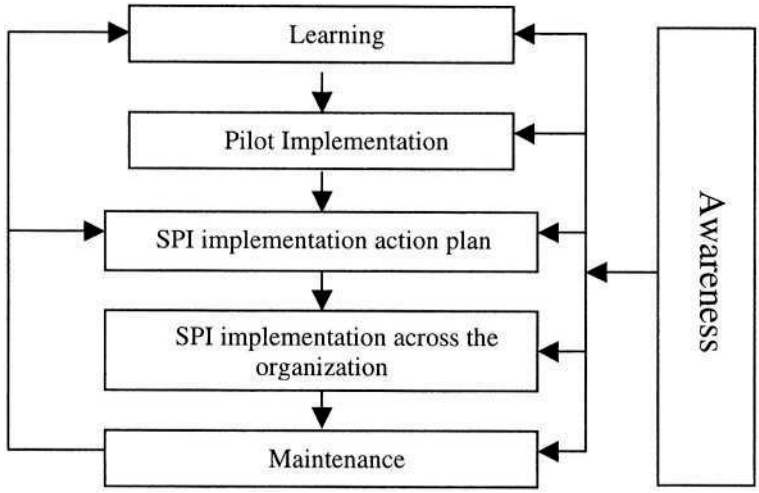


Fig. 2. SPI implementation phase dimension

Implementation across the organization. After proper planning and using the pilot implementation experience, practitioners suggested to implement SPI practices in other areas/departments of the organization in order to have uniform development approach and maturity across the organization. It is also important to illustrate the results of the pilot implementation to different departments in order to get support and confidence. In order to avoid risks and to implement SPI programmes more effectively, practitioners suggested project-by-project implementation. This is because each project experience can be viewed to determine what was accomplished and how the organization can implement SPI programmes more effectively for future projects. Practitioners emphasised that senior management commitment plays a very important role in this phase. They also suggested providing sufficient resources for SPI implementation in this phase.

Maintenance. The important theme in maintenance is to continuously monitor and support the previously implemented SPI activities. Practitioners suggested continuing awareness and training programmes to be incorporated into maintenance activities as practitioners often switch jobs.

SPI efforts do not have long lasting effects because practitioners often slide back to their old habits [12]. It is therefore very important to continuously provide them with feedback, guidance, motivation and reinforcement to stay involved in the improvement effort [12].

Table 3. SPI implementation CSFs dimension

Phase	CSFs	Critical Barriers
Awareness	▪ Snr management commitment ▪ Staff involvement ▪ SPI awareness	▪ Organizational politics ▪ Lack of awareness ▪ Lack of support
Learning	▪ Snr management commitment ▪ Training and mentoring ▪ SPI awareness	▪ Time pressure ▪ Lack of awareness ▪ Lack of support ▪ Lack of resources
Pilot Implementation	▪ Snr management commitment ▪ Creating process action teams ▪ Experienced staff ▪ Formal methodology ▪ SPI awareness	▪ Inexperienced staff ▪ Lack of formal methodology ▪ Lack of support ▪ Lack of resources
SPI implementation action plan	▪ Snr management commitment ▪ Experienced staff ▪ Formal methodology ▪ Reviews ▪ SPI awareness	▪ Inexperienced staff ▪ Time pressure
SPI implementation across the organization	▪ Snr management commitment ▪ Staff time and resources ▪ Staff involvement ▪ Experienced staff ▪ SPI awareness ▪ Formal methodology	▪ Organizational politics ▪ Time pressure ▪ Inexperienced staff ▪ Lack of methodology ▪ Lack of support ▪ Lack of resources ▪ Lack of SPI awareness
Maintenance	▪ Snr management commitment ▪ Reviews ▪ SPI awareness	▪ Inexperienced staff

5.2 SPI Implementation CSF and Critical Barrier Dimension

The CSFs approach has been applied to different areas of IT and management and different studies have confirmed the value of the CSF approach [35-37]. Implementation of SPI programmes require real life experiences where one learns from mistakes and continuously improves the implementation process. CSFs are often identified after the successful completion of certain activities. Hence these factors are near-to real life experiences. Therefore, we believe that CSFs approach can also be useful in the implementation of SPI.

A review of the CSF literature reveals that the CSF concept has not been employed to any great degree in research on the topic of SPI implementation. Literature also shows that little attention has been paid to developing ways in order to improve the SPI implementation process [10]. A few studies have identified key factors for SPI implementation [9; 14] but these studies have not described how and where to use

these factors. We believe that the identification of factors by themselves is not sufficient for the effective design of SPI implementation strategies but a holistic approach is required in order to successfully implement SPI programmes.

Keeping in view these facts we have identified different CSFs and critical barriers through the literature and an empirical study. We used the frequency analysis technique and calculated the relative importance of each factor (see Tables 1 and 2). As CSFs are a small number of important issues on which management should focus their attention [11], we have only considered those factors that are critical (i.e. with high percentage) either in both data sets or in any data set. We have divided CSFs and critical barriers among different phases of the SPI-IM as shown in Table 3. The basis of this division is the perceived coherence between the CSFs and critical barriers identified. It should also be pointed out that these factors and barriers are not necessarily mutually exclusive and there may be a certain degree of overlap among them.

Table 3 suggests that practitioners should consider these CSFs and critical barriers in order to successfully implement each phase of the SPI-IM because we have confidence that a factor does indeed have an impact on SPI implementation if it is critical in both data sets or in any data set. For example, in the *learning* phase it is important to have higher management support, SPI training and SPI awareness activities in order to provide sufficient knowledge about SPI and its benefits to the organization. Similarly, in the *learning* phase sufficient attention should be paid to avoid barriers such as time pressure, lack of awareness, lack of support, lack of resources in order to prevent undermining of SPI implementation process.

In order to have more fine grained activities within each phase of our SPI-IM, we have designed a list of practices for each CSF and critical barrier (a few examples are shown in Appendix C) using our empirical study and the literature [9; 12-14; 38; 39]. These practices guide practitioners on how to implement each CSF and critical barrier in practice.

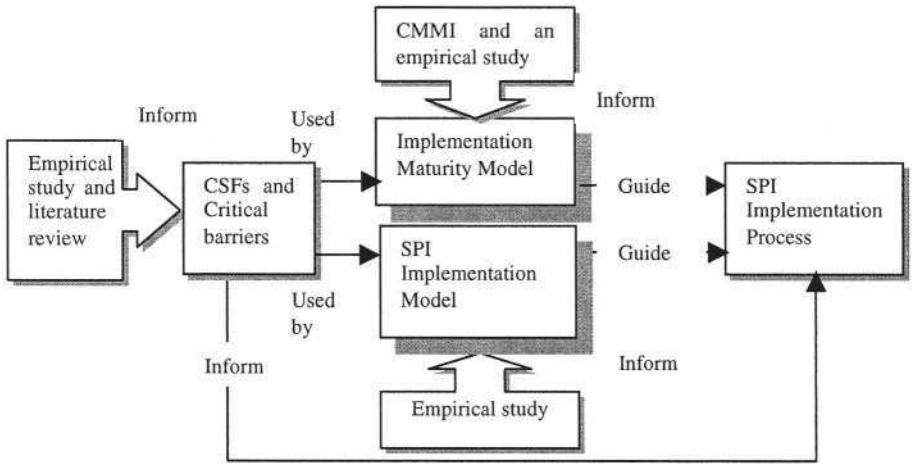


Fig. 3. SPI implementation framework

6 Conclusion and Future Work

In this paper a new model is presented that has the potential to help companies substantially improve their SPI implementation processes. This model has six phases and provides a very practical structure within which to implement SPI programmes.

Our ultimate aim of conducting an empirical study and developing the implementation model was to develop a SPI implementation framework. We have used our empirical findings to design a framework for guiding the design of effective implementation strategies for software process improvement [17] (Figure 3). As shown in Figure 3, the framework comprises an Implementation Maturity Model [40] and an SPI Implementation Model [41]. The CSFs and critical barriers identified through empirical study are used by the two models. These CSFs and critical barriers are used in order to provide practitioners with sufficient knowledge about the nature of issues that play a positive or negative role in the implementation of SPI programmes. The two models will guide the SPI implementation process.

This paper has described only one component of the framework, i.e. the SPI implementation model. The framework is currently in its initial stages and will be further refine on the basis of interviews with industry practitioners to confirm and extend the findings of the literature survey. The framework will then be evaluated using multiple case studies.

References

1. Paul, B.: On for young and old as James and Kerry began to fret. The Sydney Morning Herald. http://www.smh.com.au/articles/2002/03/20/Pbonetel.htm: Site visited 12-9-2003 (2002)
2. Finkelstein, A.: Report of the Inquiry Into The London Ambulance Service. International Workshop on Software Specification and Design Case Study Electronic: http://www.cs.ucl.ac.uk/staff/A.Finkelstein/las/lascase0.9.pdf: site visited 4-3-2003, 1993. (1993)
3. Lions, J. L.: http://www.ima.umn.edu/~arnold/disasters/ariane5rep.html: site visited 4-3-2003. (1997)
4. Pitterman, B.: Telcordia Technologies: The journey to high maturity, IEEE Software (July/August). (2000) 89-96.
5. Yamamura, G.: Software process satisfied employees, IEEE Software (September/October). (1999) 83-85.
6. Paulk, M., Curtis, B., Chrissis, M. and Weber, C.: Capability Maturity Model for software, Version 1.1. CMU/SEI-93-TR-24, Software Engineering Institute USA (1993)
7. SEI: Capability Maturity Model® Integration (CMMISM), Version 1.1. SEI, CMU/SEI-2002-TR-029 (2002)
8. ISO/IEC-15504: Information technology - Software process assessment. Technical report -Type 2 (1998)
9. Goldenson, D. R. and Herbsleb, J. D.: After the appraisal: A systematic survey of Process Improvement, Its benefits, And Factors That Influence Success. SEI, CMU/SEI-95-TR-009 (1995)
10. Herbsleb, J. D. and Goldenson, D. R.: A systematic survey of CMM experience and results. 18th international conference on software engineering (ICSE-18). Germany (1996)
11. Rockart, J. F.: Chief executives define their own data needs, Harvard Business Review (2). (1979) 81-93.

12. Stelzer, D. and Werner, M.: Success factors of organizational change in software process improvement, Software process improvement and practice 4 (4). (1999)
13. El-Emam, K., Fusaro, P. and Smith, B.: Success factors and barriers for software process improvement. Better software practice for business benefit: Principles and experience, IEEE Computer Society (1999)
14. Rainer, A. and Hall, T.: Key success factors for implementing software process improvement: a maturity-based analysis, Journal of Systems & Software (62). (2002) 71-84.
15. Rainer, A. and Hall, T.: A quantitative and qualitative analysis of factors affecting software processes, Journal of Systems & Software, Accepted awaiting publication (2002)
16. Florence, A.: Lessons learned in attempting to achieve software CMM Level 4, CrossTalk (August). (2001) 29-30.
17. Niazi, M., Wilson, D. and Zowghi, D.: A framework for guiding the design of effective implementation strategies for software process improvement. International Conference on Knowledge Engineering and Software Engineering (SEKE 03). (2003) 366-371.
18. Niazi, M. and Wilson, D.: A Maturity Model for the Implementation of Software Process Improvement. International Conference on Software Engineering Research and Practice (SERP03). (2003) 650-655.
19. Krippendorf, K.: Content Analysis: An introduction to its Methodologies. sage London.(1980)
20. Coolican, H.: Research Methods and Statistics in Psychology. Hodder and Stoughton, London.(1999)
21. Baddoo, N. and Hall, T.: Motivators of software process improvement: An analysis of practitioner's views, Journal of Systems and Software (62). (2002) 85-96.
22. Baddoo, N. and Hall, T.: De-Motivators of software process improvement: An analysis of practitioner's views, Journal of Systems and Software 66 (1). (2003) 23-33.
23. Leedy, P. and Ormrod, J.: Practical research: planning and design. Prentice Hall-New Jersey.(2001)
24. Bullen, C. V. and Rockart, J. F.: A primer on critical success factor, Centre for Information Systems research. Sloan School of Management, Working Paper No. 69 (1981)
25. Baddoo, N.: Motivators and De-Motivators in software process improvement: an empirical study, PhD, University of Hertfordshire UK (2001)
26. Burnard, P.: A method of analysing interview transcripts in qualitative research, Nurse education today (11). (1991) 461-466.
27. Seaman, C.: Qualitative methods in empirical studies of software engineering, IEEE Transactions on Software Engineering 25 (4). (1999) 557-572.
28. Weigers, K. E.: Software process improvement: Eight traps to avoid, CrossTalk (September). (1998) 9-12.
29. Moitra, D.: Managing change for (SPI) initiatives: A practical experience-based approach, Software Process Improvement and Practice (4). (1998) 199-207.
30. Butler, K.: Process lessons learned while reaching Level 4, CrossTalk (May). (1997) 1-6.
31. Paulk, M.: Practices of high maturity organizations. SEPG Conference. (1999) 8-11.
32. Dyba, T.: An Instrument for measuring the key factors of success in SPI, Empirical software engineering (5). (2000) 357-390.
33. Hall, T. and Wilson, D.: Views of software quality: a field report, IEEE Proceedings on Software Engineering 144 (2). (1997)
34. Wilson, D. and Hall, T.: Perceptions of software quality: a pilot study, Software quality journal (7). (1998) 67-75.
35. Huotari, M. l. and Wilson, T. D.: Determining organizational information needs: the critical success factors approach, Information research 6 (3). (2001)
36. Khandelwal, V. and Natarajan, R.: Quality IT management in Australia: Critical success factors for 2002. Technical report No. CIT/1/2002, University of Western Sydney (2002)

37. Somers, T. and Nelson, K.: The impact of critical success factors across the stages of Enterprise Resource Planning Implementations. 34th Hawaii International Conference on System Sciences. (2001)
38. Johnson, A.: Software process improvement experience in the DP/MIS function: Experience report. IEEE International Conference on Software Engineering, ICSE. (1994) 323-329.
39. Zubrow, D., Hayes, W., Siegel, J. and Goldenson, D.: Maturity Questionnaire. CMU/SEI-94-SR-7 (1994)
40. Niazi, M., Wilson, D. and Zowghi, D.: A Maturity Model for the Implementation of Software Process Improvement: An empirical study: To Appear in, Journal of Systems and Software (2003)
41. Niazi, M., Wilson, D. and Zowghi, D.: A model for the implementation of software process improvement: A pilot study, To appear in the proceedings of. International Conference on Software Quality (QSIC03). (2003)

Appendix A: Experience Reports, Case Studies, and Papers

- Advanced information services
- AVX Ltd
- Boeing's Space Transportation Systems
- Bull HN
- Corning Information Services
- Eastman Kodak Comp.
- Fastrak Training Inc.
- High-Tech Measurement
- Hughes
- Lucent Technologies
- MITRE Corporation
- Master Systems
- NASA SEL
- Network Products
- Nokia
- Oerlikon Aerospace
- Ogden Air Logistics Centre
- Oklahoma City Air Logistics Centre
- Raytheon
- Rolls-Royce
- Sacramento Air Logistics Centre
- Schlumberger
- SEI
- Siemens
- SINTEF Telecom and Informatics
- Space Shuttle Software Project
- Sybase
- Tata Consulting Services
- Texas Instruments
- Telcordia Technologies
- Trident Data Systems
- University of Hertfordshire
- Xerox

Note: References to these organizations are available from authors

Appendix B: Participant Organization Information

Company	Scope	Age (yrs)	Software size	Primary function	SPI age (yrs)
1	Australian	3	14	Software	< 1
2	Multi-national	21-50	DK	Services	> 5
3	Multi-national	>50	101-500	Services	> 5
4	Multi-national	11-20	501-2000	Services	1-2

5	Australian	6-10	<10	Software	>5
6	Australian	21-50	30	Software/Services	3-5
7	Multi-national	21-50	DK	Software/Services	>5
8	Multi-national	>50	26-100	Software	>5
9	Multi-national	>50	>2000	Software/Services	>5
10	Australian	>50	11-25	Services	3-5
11	Multi-national	>50	>2000	Financial services	3-5
12	Australian	<5	<10	Software/Services	1-2
13	Multi-national	>50	DK	Software/Services	>5
14	Multi-national	11-20	>2000	Software/Services	3-5
15	Australian	21-50	101-500	Software/Services	1-2
16	Multi-national	21-50	>2000	Software/Services	>5
17	Multi-national	11-20	11-25	Beverages	>5
18	Multi-national	>50	101-500	Software	>5
19	Australian	11-20	11-25	Software	1-2
20	Australian	21-50	DK	Investment Management	>5
21	Multi-national	<5	11-25	Software	1-2
22	Australian	11-20	11-25	Software	3-5
23	Multi-national	6-10	26-100	Software	3-5
24	Australian	<5	<10	Software/services	3-5
25	Australian	6-10	101-500	Services	>5
26	Australian	6-10	26-100	Services	>5

Appendix C: List of Practices for CSFs and Critical Barriers

CSFs and Critical barriers	List of Practices
Creating process action teams	P1. SPI implementation action groups have been established with experienced people P2. Responsibilities have been assigned to provide technical support to the process action teams P3. A mechanism has been established to monitor the progress of each process action team P4. A mechanism has been established to collect and analyze the feedback data from each process action team and to extract the main lessons learned P5. A process has been established to distribute the lessons learned to the relevant staff members
Time pressure	P1. Staff members have been allocated time for SPI efforts and staff members are happy with allocated time P2. Work has been done to avoid staff from time pressure P3. Work has been done that SPI will not get in the way of real work P4.The SPI implementation effort has been staffed by people who indicated interest and commitment in the effort. P5 Work has been done to facilitate staff members during SPI implementation

Does Use of Development Model Affect Estimation Accuracy and Bias?

Kjetil Moløkken[1], Anette C. Lien[1], Magne Jørgensen[1], Sinan S. Tanilkan[2], Hans Gallis[1], and Siw E. Hove[1]

[1] Simula Research Laboratory, P.O.Box 134, 1325 Lysaker, Norway
{kjetilmo, anetteli, magnej, hansga, siweh}@simula.no
[2] Department of Informatics, P.O.Box 1080 Blindern, 0316 Oslo, Norway
sinant@ifi.uio.no

Abstract. *Objective.* To investigate how the use of incremental and evolutionary development models affects the accuracy and bias of effort and schedule estimates of software projects. *Rationale.* Advocates of incremental and evolutionary development models often claim that use of these models results in improved estimation accuracy. *Design of study.* We conducted an in-depth survey, where information was collected through structured interviews with 22 software project managers in 10 different companies. We collected and analyzed information about estimation approach, effort estimation accuracy and bias, schedule estimation accuracy and bias, completeness of delivered functionality and other estimation related information. *Results.* We found no impact from the development model on the estimation approach. However, we found that incremental and evolutionary projects were less prone to effort overruns. The degree of delivered functionality and schedule estimation accuracy, on the other hand, were seemingly independent of development model. *Conclusion.* The use of incremental and evolutionary development models may reduce the chance of effort overruns.

1 Introduction

Software projects are prone to exceed their budgets. A review of surveys on effort estimation indicates that 60-80 % of all software projects encounter budget overruns [1]. The average overrun of a software project appears to be 30-40%

Expert estimation seems to be the preferred approach employed by most software professionals [1, 2]. In order to improve estimation accuracy, several estimation models have been developed, based on a variety of properties such as lines of code [3], function-points [4] or regression [5]. There is, however, no conclusive evidence that the employment of such models results in improved estimation accuracy [1, 2]. Another line of research has been directed towards improving expert estimation by using a variety of processes, including work breakdown structures (WBS) [6, 7], checklists [8, 9], experience databases [10] and group based estimates [9, 11-14].

Much less attention has been paid to ways in which choice of development model may affect estimation accuracy, even though it is claimed that the use of certain development models results in improved accuracy. The use of both incremental and

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 17–29, 2004.

evolutionary development models is believed to facilitate more accurate estimation, when compared to traditional models such as the Waterfall model [15, 16].

The estimation approach associated with the incremental development model is believed to be more meaningful than those approaches associated with other development methods and to lead to better managed projects [3, 16-18]. For example, when using the incremental model, the first increments provide early feedback on important development challenges and thereby better estimates on subsequent estimation revisions [16]. If a project suffers from overruns, later increments can be eliminated, and the most useful parts of the system will still be produced within the original budget. There may, however, be management and control problems in connection with incremental development [16], and hence the total impact of the use of the incremental development model is not obvious.

Improved project estimation has also been attributed to evolutionary development [19]. For example, Gilb claims that "Evolutionary delivery, and the feedback it produces, often leads to management acceptance of revised estimates for time and money at early stages of the project."

Another of the claimed benefits of incremental and evolutionary development is that use of these models enables tighter control of the development process [19]. This may enable a focus on developing core functionality, and less emphasis on what is often labeled as "nice to have" functionality. Thus, effort and schedule overruns that occur due to the development of unimportant or unspecified functionality, which could result from the use of other development models, may be avoided.

The remainder of this paper is organized as follows. Section 2 examines earlier empirical studies on the relation between development method and estimation accuracy and bias. Section 3 states our research questions. Section 4 gives a brief description of differences between common development methods. Section 5 describes our research method. Section 6 the results of our survey. These results are discussed in section 7. A conclusion is presented in section 8.

2 Previous Studies

Unfortunately, it is difficult to find empirical research that investigates the above claims. In fact, we have been unable to find *any* survey or experiment on software effort estimation that treats the development model as a factor that may influence project outcome. We were only able to find experience with development models and estimation in case studies, with little control and limited samples.

One example is experience with evolutionary development of the IBM Federal System [20]. That case study reports that all 45 deliveries during a 4-year period were on time and did not exceed the allocated budget.

Also, it was observed that using the evolutionary spiral model for a PC-based ground control system developed at the Marshall Space Flight Centre at NASA achieved good results (Hendrix and Schenider 2002). The researchers found that the spiral model forced risks to be identified, planned for, and resolved, while providing the flexibility to meet volatile constraints. A lifecycle based on the spiral model provided a way of addressing risks inherent in the project at the start, and offered a framework flexible enough to meet changing priorities [21]. Before entering the first iteration of the spiral, it was, of course, unknown what number of iterations that

would be needed for completion of the project. Hence, the team planned only the first iteration and did not engage in detailed, long-term spiral planning. Once the first iteration was complete and the second begun, an estimate was made for the additional iterations needed to address the current release requirements. The team gained knowledge and experience as the project moved through the spiral iterations, thereby increasing the quality and accuracy of planning and risk assessment. The experience from the projects showed that the level of accuracy of estimates performed at the beginning of each spiral iteration was 90–95% [21].

Royce [22] describes experiences when using an incremental development model for large software systems for North American Aerospace Defence Command/Air Force Space Command. Observations indicated that use of the incremental model increases the probability that a project will be completed according to schedule and budget.

However, there are also reports with less positive observations, for example a study describing experiences from two projects that used the evolutionary development model [23]. That study found that a lot of user initiated changes caused one of the projects to experience delays amounting to half the total effort. That project used 8600 person-hours, whereas the initial estimate was 4300 person-hours. The release was delayed by three months. However, the end users and the developers strongly believe that the delay, which was due to a great portion of change requests from the user, was necessary for the company to deliver a product that met the real needs of the customer. The project management, however, regarded the delay as a disaster, and they were very confused. They believed that they lost control of the project because the requirements changed continuously, without the formal documentation being changed accordingly. Therefore, the evolutionary model was abandoned and replaced by a formal waterfall model. This change back to a waterfall model was believed to be necessary to get budget and time schedules under control. In the other project, however, the software cost estimate was accurate, despite a number of requests for changes by the end users.

3 Research Questions

Due to the lack of previous research regarding the impact of the use of particular development models on estimation accuracy, we decided not to state precise hypotheses for investigation. Instead we present more informal research questions.

RQ1: *Does the use of incremental or evolutionary development models affect the software estimation approach*?

RQ2: *Does the use of incremental or evolutionary development models affect effort estimation accuracy*?

RQ3: *Does the use of incremental or evolutionary development models affect effort estimation bias?*

RQ4: *Does the use of incremental or evolutionary development models affect schedule estimation accuracy*?

RQ5: *Does the use of incremental or evolutionary development models affect schedule estimation bias?*

RQ6: *Does the use of incremental or evolutionary development models reduce the amount of unspecified functionality developed?*

4 Software Development Models

For the purpose of our analyses we classified the development models into several, coarse-grained categories. It is important to note that some of these categories are not disjunctive, i.e., some projects may use more than one category of model. Also, categories of development models may include process support elements such as component based development (CBD) and prototyping. CBD is a software development strategy in which existing components are reused instead of developing the system from scratch each time [24]. Prototyping is used to provide initial versions of a software system during development [19].

4.1 The Waterfall Model

The traditional waterfall model is a linear-sequential model [25]. It separates system development into distinct phases. Each phase should in principle be terminated before moving to the next phase, but in practice, stages overlap and feed information to each other.

4.2 Incremental Development

Incremental development is a stepwise development of a system in which the system is partitioned into parts (i.e., increments or code modules) [16]. Analysis, design, implementation and testing are performed for each increment. The increments are delivered either sequentially or in parallel. Each of the increments provides a subset of the product's functionality. To minimize risks, the most critical part of the system may be delivered first.

4.3 Evolutionary Development

In evolutionary development, the system is developed cyclically, with system deliveries in versions [19], in contrast to the "big bang" delivery provided in the traditional Waterfall model. The delivered versions are adjusted according to customer response and delivered again for a new assessment. Evolutionary development differs from incremental development in that development at each stage is based on previous deliveries that have been evaluated by the end users. The number of deliveries required can be tailored to the situation at hand.

4.4 Light and Agile Development

Abrahamson, Salo et al. [26] defines an agile development model as having the following properties: incremental (small software releases with rapid cycles), cooperative (customer and developers working constantly together with close communication), straightforward (the model itself is easy to learn and to modify, and is well documented), and adaptive (last minute changes can be made). Light and agile development address only the functions most needed by the client (the functions that deliver most business value to the customer). These functions are then delivered as quickly as possible, and feedback collected, which will then be used to prompt further development.

5 Method

The survey was conducted from February to May 2003. Due to the explorative nature of the study, there was more emphasis on qualitative than quantitative aspects. However, data was collected from a sufficient number of projects to enable basic statistical analyses.

5.1 The Participating Companies

In order to ensure a representative sample, stratified random sampling [27] was used. This is a reasonable approach, since we were going to investigate a limited number of companies and we wanted to ensure that we had companies that represented different types of organization, such as software houses developing products for the mass market, contractors who develop for customers and the internal development departments of large companies.

Each organization was contacted by phone and the study presented to them. If they agreed to participate, they were given time to prepare before they were visited by our researchers. Each company submitted one to three projects, depending on available resources for participation. The ten organizations that participated had between 30 and 750 employees, with an average of 155. About half of their development was new projects, while the rest was maintenance.

5.2 Data Collection and Analysis

We collected data via personal interview, which approach yields data of high quality and ensures that ambiguities are resolved [27]. This was especially important in our survey, since there may be variations in use of, and interpretations of, terms related to development models and estimation methods. It also allows the respondents to add valuable information that does not fit into a predefined questionnaire. Another point in favor of this approach is that our personal involvement indicates a seriousness of intent to the participants, and this may increase the likelihood of serious contributions from them. The main limitation of the approach is that it is time-consuming and hence

prevents us from investigating as many companies and projects as would be possible by using mailed questionnaires. All interviews were taped.

Following data collection, the questionnaires and tapes were registered into databases and processed by independent personnel who had no stake in the research. This is especially important, because it ensures that there are no biases regarding how the development models or estimation approaches are classified.

In cases where the participants reported that they followed a company-defined development model or estimation approach, we asked them to provide thorough descriptions. After all of the interviews had been conducted, the independent analyst listened to the tapes and categorized the customized models according to the predefined categories presented in the previous section.

In order to provide a meaningful analysis, those projects that used elements of the incremental and evolutionary development models are, in the remainder of this paper, described as the incremental/evolutionary (I/E) development group. This category also includes light and agile development. None of these projects were influenced by waterfall development. Those projects that did not use incremental or evolutionary development, but used waterfall development, are placed in the waterfall (W) category.

This classification is undeniably coarse, but the distinction is meaningful in the context of investigating approaches to estimation and the project outcome. Incremental/evolutionary models and waterfall models are often differentiated between when project estimation is at issue, because incremental and evolutionary development allows several estimation revisions [16, 19], which should, in theory, result in the provision of more accurate estimates. We wanted to investigate whether or not this claim is true.

In order to calculate estimation accuracy, the magnitude of relative error, MRE is used [28]. This is calculated as:

$$MRE = \frac{|x - y|}{x}, \quad \text{x = actual effort and y = estimated effort.} \tag{1}$$

Estimation bias (labeled MRE-bias) is calculated with a similar equation. The only difference is that the absolute value is not used in order to reveal an eventual direction of inaccuracy.

$$MRE - bias = \frac{(x - y)}{x}, \quad \text{x = actual effort and y = estimated effort.} \tag{2}$$

If a project had several estimation revisions, we used the revision from the planning estimate as a basis for this analysis. The planning estimate is typically made at the stage in the project where the project is broken down into activities and resources are assigned to the project.

In order to investigate whether the development model influenced the completeness of delivered functionality, we interviewed the project managers about the delivered functionality on the finished products. They were asked about the extent to which the delivered functionality met the original specification on which the estimates were based. Since we did not have access to the opinions of the users of the products, there is a potential problem regarding the objectivity of answers. In

addition, there may be a hindsight bias. However, we have no reason to believe that such factors would affect one of the survey groups differently from the other.

6 Results

Thirteen projects were classified as using an incremental or evolutionary model, or a combination of these. These were placed in the I/E group. The other nine projects followed the waterfall development model. These were placed in the W group. A complete record of the data is presented in Table 1.

Effort is measured in man-hours, and schedule in calendar days. Schedule information for project 4 was not available. The projects are sorted according to effort estimation MRE-bias.

Table 1. Estimated and Actual use of Effort and Schedule

		Effort				Schedule			
Project	Model	Actual	Estimated	MRE	Bias	Actual	Estimated	MRE	Bias
1	I/E	825.0	3750.0	3.55	-3.55	152	138	0.09	0.09
2	I/E	955.0	1410.0	0.48	-0.48	64	74	0.16	-0.16
3	I/E	466.5	533.5	0.14	-0.14	104	97	0.07	0.07
4	I/E	1000.0	1125.0	0.13	-0.13	N/A	N/A	N/A	N/A
5	I/E	319.0	330.0	0.03	-0.03	235	235	0.00	0.00
6	I/E	1242.0	1249.0	0.01	-0.01	106	92	0.13	0.13
7	I/E	1512.0	1500.0	0.01	0.01	70	70	0.00	0.00
8	W	1150.0	1077.0	0.06	0.06	49	42	0.14	0.14
9	W	562.5	487.5	0.13	0.13	106	103	0.03	0.03
10	W	14000.0	12000.0	0.14	0.14	640	619	0.03	0.03
11	I/E	342.0	292.0	0.15	0.15	113	113	0.00	0.00
13	I/E	2732.0	2265.0	0.17	0.17	245	210	0.14	0.14
14	W	5170.0	4227.0	0.18	0.18	98	98	0.00	0.00
15	I/E	1000.0	705.0	0.30	0.30	70	60	0.14	0.14
12	I/E	3454.0	2340.0	0.32	0.32	245	140	0.43	0.43
16	I/E	907.0	570.0	0.37	0.37	116	109	0.06	0.06
17	I/E	1200.0	750.0	0.38	0.38	457	117	0.74	0.74
18	W	1000.0	560.0	0.44	0.44	140	84	0.40	0.40
19	W	600.0	300.0	0.50	0.50	156	156	0.00	0.00
20	W	1400.0	700.0	0.50	0.50	224	182	0.19	0.19
21	W	1932.5	914.0	0.53	0.53	217	196	0.10	0.10
22	W	9000.0	4000.0	0.56	0.56	335	293	0.13	0.13
Average		2307.7	1867.5	0.41	0.02	188	154	0.14	0.13
Median		1075.0	995.5	0.24	0.16	140	113	0.10	0.09

6.1 Differences in Estimation Approach

All projects, except one, relied on expert estimation. The exception used a combination of expert and a use-case based model. The average effort used to estimate the projects was 30 work hours for the I/E projects, and 46 work hours for the W projects. There is also only a slight difference in the number of estimation revisions. Out of the thirteen I/E projects, one had two revisions, five had one

revision, while eight had none revisions. Out of the nine W projects, one had two revisions, four had one revision, and four had none revisions.

The groups had similar approaches to bottom-up/top-down estimation, independent of development model. All the projects involved bottom-up estimation.

The use of WBS was not as widespread. Only nine of the projects used a predefined WBS. There was, however, no systematic difference in the use of WBS between the two main survey groups.

Experience databases were used for three of the I/E projects, and for none of the W projects. Checklists on the other hand, were used for one of the I/E projects, and six of the W projects.

There were no differences in the involvement of the estimators. All project estimators participated in project development themselves. Combination of estimates via group interaction was typical in both groups.

These project properties, and other comments provided by the project managers, indicate that there are no systematic differences in how projects are estimated that are related to the development model used. The approach to estimation seems to depend on the project team involved.

6.2 Effort Estimation Accuracy

In order to investigate a possible difference in estimation accuracy between the survey groups, a statistical t-test can be applied. However, to use the t-test properly, the underlying distributions should be approximately normal. An Anderson-Darlington normality test [29], which resulted in $p = 0.0$, and a visual inspection, excludes normality for the MRE of our dataset. However, this has generally negligible effects on the validity of both Type I and Type II error calculations [30]. The only exceptions are when there are substantially unequal variances and substantially unequal sample sizes. In our case, the sample sizes were unequal (9 and 13), and the difference in variances was large (0.13 vs. 0.98).

Due to the questions raised as to the appropriateness of the t-test for the dataset, we chose instead to use the more robust non-parametric statistical Kruskal-Wallis [31] test on the median MRE values, which were 0.17 for the I/E group, and 0.44 for the W group. We provide the actual p-values, as suggested by Wonnacott and Wonnacott [32], instead of pre-defining a significance level for rejection. The Kruskal-Wallis test resulted in a p-value of 0.22, so although there was a large difference in median MRE-values, we cannot on basis on the data alone reject the hypothesis of no difference.

6.3 Effort Estimation Bias

The same properties regarding normality, sample size and standard deviation apply to the MRE-bias as to the MRE. A Kruskal-Wallis test was therefore appropriate. Performed on the median MRE-biases (I/E = 0.01 and W = 0.44), it resulted in a p-value of 0.02, so we can reject the hypothesis of no difference in median MRE-bias.

To measure the magnitude of the observed accuracy difference in MRE-bias, we included Cohen's size of effect measure (d) [30]. The size of effect (d) was calculated as: d = (mean value W group– mean value I/E group) / pooled standard

deviation amongst groups W and I/E. The result was d=0.65, which is considered a medium effect, since d is between 0.5 and 0.8 [30].

6.4 Schedule Estimation Accuracy

Due to the properties of the data, as explained in the previous subsection, we performed a Kruskal-Wallis [31] test on the median MRE of schedule estimation accuracy (I/E median MRE = 0.11 and W median MRE = 0.10). This resulted in a p-value of 0.75.

6.5 Schedule Estimation Bias

Since only one project finished ahead of schedule, the results for schedule MRE-bias are virtually identical to those for schedule MRE.

6.6 Differences in Product Functionality

The results indicate no large difference related to delivered functionality between the groups. Eighteen out of the 22 managers reported a delivered level of functionality between 95% and 110%. The average for I/E projects was 109% and the corresponding figure for the W projects was 105%.

7 Discussion

Our results suggest that the particular development model used may affect estimation bias. The results from the survey indicate that projects that are developed with an incremental or evolutionary model may be less prone to effort overruns than projects that implement waterfall development models. The lack of significant difference in estimation accuracy between the groups may, we believe, be a result of the low number of observations, i.e., low power of the study. We therefore need more observations to test the difference in estimation accuracy.

The use of estimation approach seemed to be independent of development model used, and we found only minor differences in the number of estimation revisions, choice of estimation method or time spent on effort estimation between different development models. Neither was there any difference in the use of top-down vs. bottom-up estimation or use of supportive procedures such as WBS, experience databases, checklists or group based estimates.

Most surprising is the observation that the number of estimation revisions did not seem to increase in incremental or evolutionary projects. Proponents of incremental and evolutionary development models specify several estimation revisions as the main reason for reduced overruns when these models are applied.

There were no observations of difference in degree of unspecified functionality delivered between the two groups. This implies that differences in schedule and effort

overruns for these projects are not due to differences in the amount of delivered functionality.

The schedule estimation accuracy and bias of the projects in this survey did not seem to be affected by choice of development model. Neither did it seem to be closely related to effort estimation accuracy or bias, as many projects that finished under budget still had overruns related to schedule.

To summarize, it seems that the particular development method used affect effort estimation bias, but not accuracy or bias of schedule estimation. The underlying reasons for these observations are not, however, easy to discern, since neither the estimation approach, nor the degree of delivered functionality could explain the observed differences.

The following sub-sections investigate other possible differences between projects in the groups that may explain our results, and possible threats to validity.

7.1 Differences in Company Maturity Level

A possible explanation of the differences is that more mature companies employ incremental or evolutionary development models, and that it is the maturity, and not the development model itself, that results in reduced estimation bias. This is difficult to assess, since Norwegian companies are rarely evaluated with respect to standards such as CMM [33] or ISO 9001 [34]. However, none of the companies relied only on one type of development model. Most often they had a repertoire of several models, and many of the companies submitted both I/E and W projects. The choice of model on a particular project depended on the project manager and project characteristics.

7.2 Differences in Project Size

Another factor that could explain the differences between our survey groups is differences in project size, because large projects may have larger relative effort overruns. The median size for the W group was 1400, while it was 1000 for the I/E group. However, a Kruskal-Wallis analysis of project size did not indicate a systematic size variation ($p=0.13$). It is also true that the W group had the largest project (14000 hours), but in fact, this project "only" had an MRE of 0.17, both below the median MRE (0.18) and average MRE (0.41) for all projects.

7.3 Threats to Validity

The most obvious threat to validity is the sample size, which is small when applying statistical inference-based analysis methods. However, we believed that it was more important in an exploratory survey to investigate how projects actually were estimated and executed.

Another threat to validity is that a non-random sample was used. The main reason for this is that it was essential to gather information from a broad range of companies.

Those who develop for customers, in-house developers and software houses were included.

However, the abovementioned threats notwithstanding, the results are similar to those reported by other surveys on effort estimation presented in a recent review [1]. In our survey, 73% of the projects faced effort overruns, which is similar to the results of comparable surveys [35-37]. The estimation accuracy is also similar to that reported in previous surveys [35, 38, 39]. This is an indication that the sample, even though not random, is probably not biased in any particular direction.

8 Conclusion

Due to the limited amount of previous research in the area, it is difficult to explain those of our findings that support the claim that using an evolutionary or incremental model may reduce effort overruns [3, 15-20, 40]. Especially interesting is the observation that there were no additional estimation revisions in incremental and evolutionary projects. Both for incremental and evolutionary development, this is claimed to be important for the improvement of estimation accuracy.

Another claim that was not borne out by our research was that differential usage of incremental and evolutionary development models, and waterfall development models, lead to differences in delivered functionality in the projects.

Acknowledgements. This research was funded by the Research Council of Norway under the project INCO. Thanks to Dag Sjøberg and Erik Arisholm for valuable comments.

References

1. Moløkken, K. and M. Jørgensen. *A Review of Surveys on Software Effort Estimation.* in *2003 ACM-IEEE International Symposium on Empirical Software Engineering (ISESE 2003).* 2003. Frascati - Monte Porzio Catone (RM), ITALY: IEEE.
2. Jørgensen, M., *A Review of Studies on Expert Estimation of Software Development Effort.* To appear in Journal of Systems and Software (an early version of this paper can be downloaded from: http://www.simula.no/publication_one.php?publication_id=444), In press.
3. Boehm, B., et al., *Software Estimation with COCOMO II.* 2000: Prentice-Hall.
4. Matson, J.E., B.E. Barrett, and J.M. Mellichamp, *Software development cost estimation using function points.* IEEE Transactions on Software Engineering, 1994. **20**(4): p. 275-287.
5. Miyazaki, Y., et al., *Robust regression for developing software estimation models.* Journal of Systems and Software, 1994. **27**(1): p. 3-16.
6. Woodward, H., *Project Management Institute practice standard for work breakdown structures.* 2001, Newton Square: Project Management Institute, Inc.
7. Tausworthe, R.C., *The Work Breakdown Structure in Software Project Management.* Journal of Systems and Software, 1980. **1**: p. 181-186.
8. Jørgensen, M. and K. Moløkken. *A Preliminary Checklist for Software Cost Management.* in *Accepted for QSIC 2003.* 2003.

9. Shepperd, M. and U. Passing. *An Experiment on Software Project Size and Effort Estimation.* in *2003 ACM-IEEE International Symposium on Empirical Software Engineering (ISESE 2003).* 2003. Frascati - Monte Porzio Catone (RM), ITALY: IEEE.
10. Engelkamp, S., S. Hartkopf, and P. Brossler. *Project experience database: a report based on first practical experience.* in *International Conference on Product Focused Software Process Improvement.* 2000. Oulu, Finland.
11. Linstone, H.A. and M. Turoff, *The Delphi Method: Techniques and Applications.* 1975, London: Addison-Wesley.
12. Taff, L.M., J.W. Borcering, and W.R. Hudgins, *Estimeetings: Development estimates and a front end process for a large project.* IEEE Transactions on software engineering, 1991. **17**(8): p. 839-849.
13. Moløkken, K. and M. Jørgensen. *Software Effort Estimation: Unstructured Group Discussion as a Method to Reduce Individual Biases.* in *The 15th Annual Workshop of the Psychology of Programming Interest Group (PPIG 2003).* 2003. Keele, UK.
14. Moløkken, K. and M. Jørgensen, *Group Processes in Software Effort Estimation.* To appear in Journal of Empirical Software Engineering - 2004, In press.
15. May, E.L. and B.A. Zimmer, *The Evolutionary Development Model for Software.* Hewlett-Packard Journal, 1996. **47**(4): p. 39-45.
16. Graham, D.R., *Incremental Development and Delivery for Large Software Systems,* in *Software Engineering for Large Software Systems,* B.A. Kitchenham, Editor. 1990, Elsevier.
17. Cockburn, A., *In Search of Methodology,* in *Object Magazine.* 1994. p. 52-56.
18. Cockburn, A., *Surviving Object-Oriented Projects.* 1998: Addison-Wesley.
19. Gilb, T., *Principles of Software Engineering Management.* 1988: Addison-Wesley Publishing Company.
20. Mills, H.D., *The Management of Software Engineering, Part 1: Principles of Software Engineering.* IBM Systems Journal, 1980. **19**(4): p. 414-420.
21. Hendrix, T.D. and M.P. Schenider, *NASA's TReK Project: A Case Study in Using the Spiral Model of Software Development.* Communications of the ACM, 2002. **45**(4).
22. Royce, W. *TRW's Ada Process Model for Incremental Development of Large Software Systems.* in *12th International Conference on Software Engineering (ICSE'12).* 1990. Los Alamitos, CA: IEEE.
23. Lien, A.C. and E. Arisholm. *Evolutionary Development of Web-applications - Lessons learned.* in *European Software Process Improvement Conference (EuroSPI'2001).* 2001. Limerick Institute of Technology, Ireland.
24. Crnkovic, I., *Component-Based Software Engineering - New Challenges in Software Development.* Software Focus, 2001. **2**(4): p. 127-133.
25. Royce, W. *Managing the development of large software systems: Concepts and techniques.* in *Proceedings of IEEE WESTCON.* 1970. Los Angeles.
26. Abrahamson, P., et al., *Agile software development methods. Review and analysis.* 2002, VTT Publication. p. 107.
27. Cozby, P.C., *Methods in behavioral research.* 5th ed. 1993, Mountain View: Mayfield Publishing Company.
28. Conte, S.D., H.E. Dunsmore, and V.Y. Shen, *Software Engineering Metrics and Models.* 1986, Menlo Park: Benjamin-Cummings.
29. Christensen, R., *Analysis of variance, design and regression. Applied statistical methods.* 1998: Chapman & Hall/Crc.
30. Cohen, J., *Statistical power analysis for the behavioral sciences.* 1969, New York: Academic Press, Inc.
31. Siegel, S. and N.J. Castellan, *Non-parametric Statistics for the Behavioral Sciences.* 2nd Edition ed. 1988: McGraw Hill College Div.
32. Wonnacott, T.H. and R.J. Wonnacott, *Introductory statistics.* 5th ed. 1990: John Wiley & Sons.

33. Paulk, M.C., et al., *Capability Maturity Model for Software (Ver 1.1).* 1993, Software Engineering Institute: Pittsburgh.
34. Beaumont, L.R., *ISO 9001, The Standard Interpretation.* 3rd ed. 2002: Iso Easy.
35. Jenkins, A.M., J.D. Naumann, and J.C. Wetherbe, *Empirical Investigation of Systems Development Practices and Results.* Information & Management, 1984. **7**: p. 73-82.
36. Lederer, A.L., et al., *Information System Cost Estimating: A Management Perspective.* MIS Quarterly, 1990. **14**(2): p. 159-178.
37. Heemstra, F.J., *Software cost estimation.* Information and Software Technology, 1992. **34**(10): p. 627-639.
38. Phan, D., *Information Systems Project Management: an Integrated Resource Planning Perspective Model,* in *Department of Management and Information Systems.* 1990, Arizona: Tucson.
39. Bergeron, F. and J.-Y. St-Arnaud, *Estimation of Information Systems Development Efforts: A Pilot Study.* Information & Management, 1992. **22**: p. 239-254.
40. Bassin, K., S. Biyani, and Santhanam, *Metrics to evaluate vendor-developed software based on test case execution results.* IBM Systems Journal, 2002. **41**(1).

Managing Software Process Improvement (SPI) through Statistical Process Control (SPC)

Teresa Baldassarre, Nicola Boffoli, Danilo Caivano, and Giuseppe Visaggio

Department of Informatics- University of Bari- Via E. Orabona 4- 70126-Bari-Italy
{baldassarre, boffoli, caivano, visaggio}@di.uniba.it

Abstract. Measurement based software process improvement is nowadays a mandatory activity. This implies continuous process monitoring in order to predict its behavior, highlight its performance variations and, if necessary, quickly react to them. Process variations are due to common causes or assignable ones. The former are part of the process itself while the latter are due to exceptional events that result in an unstable process behavior and thus in less predictability. Statistical Process Control (SPC) is a statistical based approach able to determine whether a process is stable or not by discriminating between the presence of common cause variation and assignable cause variation. It is a well-established technique, which has shown to be effective in manufacturing processes but not yet in software process contexts. Here experience in using SPC is not mature yet. Therefore a clear understanding of the SPC outcomes still lacks. Although many authors have used it in software, they have not considered the primary differences between manufacturing and software process characteristics. Due to such differences the authors sustain that SPC cannot be adopted "as is" but must be tailored. In this sense, we propose an SPC-based approach that reinterprets SPC, and applies it from a Software Process point of view. The paper validates the approach on industrial project data and shows how it can be successfully used as a decision support tool in software process improvement.

1 Introduction

Software process is human intensive and dominated by cognitive activities. Each input into and the output from a software process are different for each process execution. The predominant human factor implies differences in process performances and thus multiple outputs. The phenomena known as "Process Diversity" [1], implies difficulty in predicting, monitoring and improving a software process. Nevertheless, Software Process Improvement (SPI) is nowadays a mandatory activity. To face these problems the software engineering community stresses the use of measurement based approaches such as QIP/GQM [2] and time series analysis: the QIP/GQM approach is usually used to determine what improvement is needed; the time series analyses allow monitoring process performances in order to decide when to improve it and verify the effectiveness of the improvement made. There is a well-established technique for time series analysis that has shown to be effective in manufacturing but not yet in software process contexts. This technique is known as Statistical Process Control (SPC) [3], [4]. It was originally developed by Shewhart in

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 30–46, 2004.

the 1920s and then used in many other contexts. It uses several "control charts" together with their indicators to: establish operational limits for acceptable process variation; monitor and evaluate process performances evolution in time. Process performance variations are mainly due to: *common cause variations* (the result of normal interactions of people, machines, environment, techniques used and so on); *assignable cause variations* (arise from events that are not part of the process and make it unstable).

SPC is a statistical based approach that determines whether a process is stable or not by discriminating between common cause variation and assignable cause variation. A process is said to be "stable" or "under control", if it is affected by common causes only. Each control chart evaluates process performance by comparing it with a measure of its central tendency, an upper and lower limit of admissible performance variations. The interest of using SPC in software is highlighted by many contributions in literature: applications in inspections and review [5, 6, 7, 8], testing [8, 9, 10, 11], maintenance [12, 15], personal software process [32], and other problems [13, 14]. They present successful outcomes in using SPC in the context of software. However they don't exhibit a clear understanding on the meaning of control charts in the context of software process application. Furthermore many indicators that may be extracted from a control chart are often not used. Software processes differ deeply from manufacturing ones [33]. The differences are summarized in Fig. 1.

Software Process	Manufacturing Process
Human intensive, cognitive activity	Machine intensive
Input and Output are different for each process execution	Input and Output are always the same for each process execution
High variation in process performances due to human factors	Low variation in process performance
Risks always present in all phases	Most Risks concentrated in design phase than in production
Product conformance to requirement specifications is difficult to obtain	Product conformance to requirement specifications is provable

Fig. 1. Software process vs manufacturing process

A cause is the predominance of cognitive activities that encourage multiple outputs and the possibility of alternative ways for doing the same thing. Another cause is the influence of the person's performances on process.

The input into and the output from the software process are different for each instance and consequently, the process instability risk is constantly present.

Furthermore, each innovation determines a process destabilization and less predictability. This side effect influences software process until the innovation is fully integrated within it and thus becomes part of organization practices. This is often called "maturity effect" [16]. In manufacturing process this effect is not so strong. In fact, when a new machine is introduced within a product line it will most likely work well.

Another important issue that differs deeply between the two types of processes is the ease to verify the adherence of a end product to its specifications: verify that the dimensions of a piston meet the customer specification is simpler than verifying that software works correctly, that it is maintainable or robust, as the customer wants.

The large number of software process attributes and variables (people, tools, skills, experience, application domain, development language and technique etc.) suggests

that, in order to address process stability, more indicators in software than in manufacturing processes are needed. This assertion contrasts with current literature and thus the purpose of our work is to answer to the following research questions:

- Which are the most suitable indicators for software process control, among those used by SPC?
- How can such suitable indicators be interpreted to support software engineers in managing software process instability?
- How can SPC be used for efficiently supporting software process monitoring and stability investigation?

For this aim, we propose an approach that: starting from stability tests known in literature, selects the most suitable ones for software processes *(tests set);* reinterprets them from a software process perspective *(tests interpretation);* defines an investigation process for monitoring process performance and supporting decisions in SPI activities *(investigation process).*

The paper briefly presents the SPC approach and its peculiarities. Then the proposed approach is described. The approach validation is carried out on industrial wide project data. Finally discussion of results and conclusions follow.

2 Statistical Process Control

A process is an organization of man, machine, and methods into work activities to produce desired outputs [4]. It can be described by measurable characteristics that vary in time due to common or assignable cause variations. If the variation in process performances is only due to common causes, the process is said to be stable and its behavior is predictable within a certain error range. Process performances are tracked overtime on a control chart, some control limits are determined and if one or more of the values fall outside these limits, an assignable cause is assumed to be present (i.e. the process is unstable). A control chart usually adopts an indicator of the process performances central tendency (CL) and upper and lower control limits (UCLs and LCLs) to discriminate between assignable and common cause variations. In software processes, due to the scarceness of data and since measurements often occur only as individual values, the most used control charts are the XmR i.e. individual and moving range charts (Fig. 2) [8][17][18][19].

In the X chart each point represents a single value of the measurable characteristic under observation. The Central Line (CL_x) expresses the process central tendency and is calculated as the average of the available values. The control limits are set at $3sigma_x$ around the mean, i.e. $3sigma_x$ around the CLx, where $sigma_x$ is the estimated standard deviation of the observed sample of values. In the mR chart each point represents a moving range. The Central Line (CL_{mR}), is computed as the average of the moving ranges. A moving range is the absolute difference between a successive pair of observations. The control limits are set at: $3sigma_{mR}$ upon the CL_{mR} for what concerns the UCL_{mR} and to 0 for LCL_{mR}. $Sigma_{mR}$ is the estimated standard deviation of the moving ranges sample. Sigma is calculated by using a set of factors tabulated by statisticians (for more details refer to [20]). Sigma is based on statistical reasoning, simulations carried out and upon the heuristic experience that: “it works”. Other well

known charts are Xbar and R charts. They differ from XmR charts in that they consider sample data instead of single observations. All the considerations apply to all charts, although we will explicitly refer to XmR charts. From here on, we will refer to X or Xbar and R or mR charts indifferently.

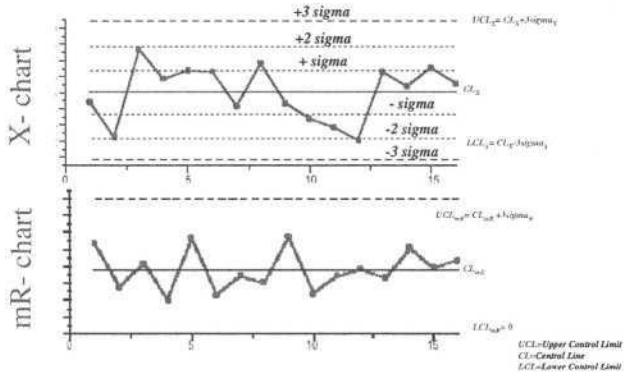


Fig. 2. Example of Individual and moving ranges charts (XmR charts)

3 The Proposed Approach

The approach here proposed is made up by:

- *Test Set:* a selection of a test set, among those presented in SPC literature, along with their rearrangement in logical classes;
- *Tests Interpretation:* the interpretation from the software process point of view, of every test in each class;
- *Investigation Process:* a process for guiding software process monitoring and stability investigation.

Each of the above points, as they are presented, answer the three research questions listed in the introduction.

Test Set

In software process, one should look for systematic patterns of points instead of single point exceptions, because such patterns emphasize that the process performance has shifted or is shifting. This surely leads to more insightful remarks and suggestions. There is a set of tests for such patterns referred to as “run rules” or “run tests”(see [21], [22], [23], [24], [25]) that aren’t well known (or used) in the software engineering community. As sigma, the run rules are based on “statistical” reasoning. For example, the probability of any observation in an X control chart falling above the

CL is at a glance equal to 0.5[1]. Thus, the probability that two consecutive observations will fall above the CL is equal to 0.5 times 0.5 = 0.25. Accordingly, the probability that 9 consecutive observations (or a run of 9 points) will fall on the same side of the CL is equal to 0.5^9 = .00195. Note that this is approximately the probability with which an observation can be expected to fall outside the 3-times sigma limits. Therefore, one could look for 9 consecutive observations on the same side of the CL as another indication of an out-of-control condition. Duncan [34] provides details concerning the "statistical" interpretation of the other tests presented in this paragraph. In order to facilitate the test execution, the chart area is conventionally divided in three zones: *Zone A* is defined as the area between 2 and 3 times *sigma* above and below the center line; *Zone B* is defined as the area between 1 and 2 times *sigma,* and *Zone C* is defined as the area between the center line and 1 times *sigma.* For the execution of the zone based tests, the distribution of the values in the charts need to be assumed as symmetrical around the mean. This is not the case for S, R, mR and other charts. In general, all zone based tests are not applicable to these charts (see Fig. 3 for applicability). Although this is a shared opinion, someone [20] states that these tests help process monitoring. Furthermore, according to [26], managers are more likely to want warning signals to be pointed out, rather than missing them, even if it means risking for false alarms.

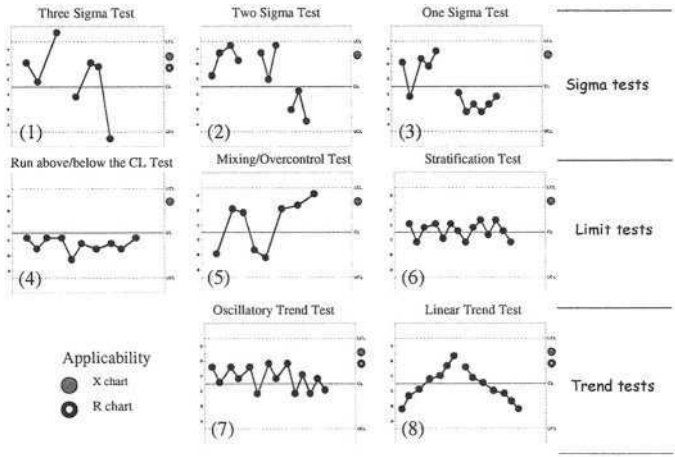


Fig. 3. Tests set

The proposed approach points out which SPC tests may be applied to which control charts. It presents, interprets and organizes tests in order to manage software process. Although in software engineering community only "a point falling outside control limits" test is usually used for testing process stability, we are of the opinion that the SPC-based software process monitoring should be based on the above tests

[1] Provided (1) that the process is in control (i.e., that the center line value is equal to the population mean), (2) that consecutive sample means are independent, and (3) that the distribution of means follows the normal distribution.

that we have rearranged in three conceptual classes according to the type of information they provide (Fig. 3). When one or more of these tests is positive, it is reasonable to believe that the process may no longer be in control, i.e. an assignable cause is assumed to be present. In the following, a brief description of each test is given, together with their applicability to X and R charts. For completeness and clearness it is the case to point out that the first 4 tests among those presented are also referred to as "detection rule" and are the most used tests [20, 27].

Sigma tests

The *three sigma test* can be applied to both, X and R charts. The *One* and *Two sigma tests* are Zone Tests and thus they should not be applied to R chart due to its lack of symmetry around the mean.

1. *Three Sigma Test* (Extreme Points Test): this test watches for a single point beyond a control limit. It signals the presence of an out-of -control condition, i.e. an assignable cause.
2. *Two Sigma Test:* This test watches for two out of three points in a row in Zone A or beyond. The existence of two of any three successive points that fall on the same side of, and more than two sigma units away from, the central line, signals the presence of an out-of -control condition. This test provides an "early warning" of a process shift.
3. *One Sigma Test:* This test watches for four out of five subgroups in a row in Zone B or beyond. The existence of four of any five successive points that fall on the same side of, and more than one sigma unit away from, the central line, signals the presence of an out-of -control condition. Like the previous test, this test may be considered to be an "early warning indicator" of a potential shift in process performance.

The *three sigma test* is the most (and often the "only") used test in software engineering literature.

Limit tests

All the tests included in this class use chart Zones and thus they are applicable to the X charts only.

4. *Run above or below the Centerline Test:* This test watches for 7, 8 or 9 consecutive observations above or below the centerline. The presence of such a run indicates that the evidence is strong and that the process mean or variability has shifted from the centerline.
5. *Mixing/Overcontrol Test:* Also called the Avoidance of Zone C Test. This test watches for eight observations in a row on both sides of the centerline avoiding Zone C. The rule is: eight successive points on either side of the centerline avoiding Zone C signals an out-of-control condition.
6. *Stratification Test:* Also known as the Reduced Variability Test. This test watches for fifteen observations in a row in Zone C, above and below the centerline. When 15 successive points on the X chart fall in Zone C, to either side of the centerline, an out-of control condition is signaled.

Trend tests

This class of tests point out a trend resulting in a process performance shift. Neither the chart centerline nor the zones come into play for these tests and thus they may be applied to both X and R charts.

7. *Oscillatory Trend Test:* it watches for fourteen alternating up or down observations in a row. When 14 successive points oscillate up and down, a systematic trend in the process is signaled.
8. *Linear Trend Test:* it watches for six observations in a row steadily increasing or decreasing. It fails when there is a systematic increasing or decreasing trend in the process.

Tests Interpretation

Sigma tests

They provide an "early" alarm indicator that must stimulate searching possible assignable causes and, if the case, their identification and further elimination. *One* and *Two sigma tests* point out a potential anomalous "trend" that "may" undertake assignable causes. In general, due to the high variance in software process especially when we manage individual rather than sample data, the faults highlighted by these tests could be numerous but less meaningful than in manufacturing contexts. For example, in a manufacturing process a party of poor quality raw material may be a potential assignable cause that must be investigated and removed. In software process a possible assignable cause may be an excessive computer crash due to a malfunctioning peripheral, but also to a headache of the developer. Different considerations could be made if the point on the chart represents a group of observations, such as the productivity of a development team. In this case the peaks accountable to a single developer's behavior are smoothened. Therefore the point on the charts may express a general behavior determined by an assignable cause.

Similar considerations can be made on the use of *Three sigma test,* based on a single observation that falls outside limits, rather than *One* or *Two sigma tests,* that refer to a sequence of observations and thus to a "potential behavioral trend".

Limit tests

This class of tests point out an occurred shift in process performance. They highlight the need to recalculate the control limits because the ones used express past process behavior with respect to the actual ones, and are inadequate. These tests either point out a performance shift with respect to the CL, or limits that are too tiny or larger than required. In software process monitoring and improvement we represent a measurable characteristic that expresses human related activity outcomes (time spent, productivity, defect found during inspection etc.) on a control chart. Thus, while a single point falling outside control limits can be interpreted as the result of a random cause, a "sequence" of points means that something has changed within the process.

Run above or below the Centerline Test: It watches for 9 points on one side of the central line. If this pattern is detected, then there is strong evidence that the software

process performance has changed in better or worse. The longer the sequence is, the stronger the evidence is.

Mixing/Over Control Test: a failure of this test could mean over control (hyper-adjustment) of the process. In software process this test failure highlights that the process is becoming less predictable than in the past. Typically this occurs immediately after an induced improvement before the improvement is fully acquired by the developers or organization.

Stratification Test: a failure of this test can arise from a change (decrease) in process variability that has not been properly accounted for in the X chart control limits. From the software process point of view this is a typical process behavior when a maturity effect is identified. Introduction of a new technology in a software process is usually followed by an unstable period until developers become more confidant and performance variability decreases (supposing a valid improvement). Substantially although in SPC theory this test highlights the presence of an assignable cause, in software process the interpretation of this test may be positive: the process is becoming more stable and predicable than in the past.

Trend tests

While the previous test classes point out the presence of an occurred shift, this one highlights an ongoing or just occurred phenomena that is resulting in an ongoing shift that needs to be investigated. Typically, a failure in this test class can be the result of both spontaneous or induced process improvement initiatives.

Oscillatory Trend Test: When this test is positive, two systematically alternating causes are producing different results. For example, we may monitor the productivity of two alternating developer teams, or monitor the quality for two different (alternating) shifts. As a consequence the measurable characteristic observed must be investigated in a more straightforward way in order to isolate the two causes. Probably when this test fails we are observing the wrong characteristic or the right one measured in a wrong way.

Linear Trend Test: It fails when there is a systematic increasing or decreasing trend in the process. This behavior is common and frequent in software processes. It is the result of an induced process improvement, such as the introduction of a new technology, or a spontaneous one, such as the maturation effect. This test gives insightful remarks when it fails on R chart and it is interpreted jointly between X and R charts. For example:

- If R chart shows a decreasing trend as in Fig. 4.d, a possible interpretation is that the process is going asymptotically towards a new stability point: better as in Fig. 4.b or worse than actual Fig. 4.a. (supposing that greater the observation value, the better the interpretation). If this is the case, this test failure should be followed by a limit test failure (typically test 4) on the X chart. Another situation is represented in Fig. 4.c i.e. a process is going towards a more stable situation around the central line, after a strong period of destabilization.

- If R chart shows a increasing trend, as in Fig. 5.d, then the process is becoming unstable, its performances are changing in a turbulent manner and it is far from reaching a new point of stability (see as in Fig. 5. a), b), c)). Typically this test failure occurs together with test 5 failure on X chart.

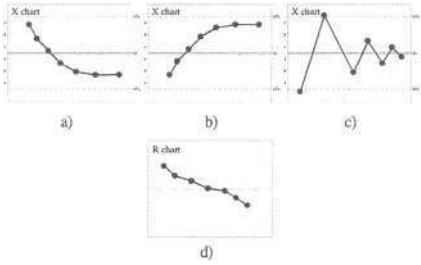


Fig. 4. Decreasing *linear trend test* interpretation

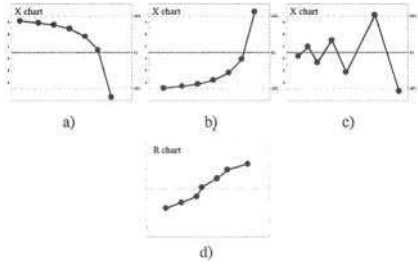


Fig. 5. Increasing *linear trend test* interpretation

Investigation Process

For successful software process improvement a software process must be constantly monitored and evaluated in order to determine its stability. This allows, on one hand, to0 quickly react with improvement initiatives in case process performances slow down, on the other hand, to verify the validity of the improvements made. The approach here proposed is able to constantly monitor process performance and point out its variation either it be spontaneous or induced by improvement initiatives.

After determining a process performance reference set, each time that a new observation is plotted on the chart, the execution of the test set must be carried out. If a failure is detected its interpretation, from a software process point of view, is required. After identifying the assignable causes that make a process unstable they must be managed. Finally, the adequateness of the reference set must be evaluated. More precisely, the steps to follow in process monitoring and stability investigation can be synthesized as follows:

Step 1: Determination of the Measurement Object. The process to evaluate is selected and the measurable characteristics that describe process performance are identified. Types of appropriate control charts [27] are selected.

Step 2: Determination of the Reference Set. The determination of a *"reference set"* is a mandatory activity in order to correctly monitor and evaluate software process performance evolution over time. It is a set of observations of the measurable characteristic of interest. This set must express the "normal" behavior of the process, i.e. the process performance supposing that only common causes variation affect it. Therefore, firstly process performance must be measured over a period of time to establish the CL and limits of variation for normal process performance. Following, the measurement data obtained must be plotted on the control charts and the test included in the test set must be executed. If no test fails, the process has been stable at

least for the period or events covered by the measurements. This first set of observations will be the *"reference set"*. If a test fails, then the process is not stable and thus, it must be investigated. The assignable causes, if present, must be removed from the process and the control limits and the central line recalculated. This is done until a reference set is determined.

Step 3: Process Monitoring and Stability Evaluation. After having determined a *reference set,* measurement continues and data are plotted on the charts. Control limits and central line, must be kept the same as in the *reference set.* For each new observation plotted on the chart, the tests must be executed. If a tests fails, then the process must be investigated in order to find out the assignable cause. If no assignable causes are identified the test failure can be considered as a false positive. In this case no corrective actions are needed and further observations can be collected. If an assignable cause is found and the result is detrimental, we must fix the process so it doesn't happen again. If the result is beneficial, we must try to make the cause part of the process. During the evaluation of the test set outcomes, the interpretation from the software process point of view previously given in the "tests interpretation" paragraph must be kept in mind and consequent actions must be enacted.

Step 4: Reference Set Evaluation. This step is carried out each time an assignable cause is detected. The consequent actions are classified and described in the following, coherently with the identified test classes. If a false positive is detected, no examination of the reference set is needed. Otherwise:

a. if a test included in *sigma tests class* has failed, the process performance:

   - should be the same as in the past if assignable causes have been detected and further removed. If this is not the case, further observations are needed to explicit the new process performance;
   - is probably changing if the assignable cause was made part of the process. In this case the collection of further observations is needed.

   In both cases the *reference set* still remain the same.

b. if a test included in *trend tests class* has failed, there should be an ongoing change or phenomena. More precisely, if a Linear Trend test fails it is not advisable to determine a new *reference set.* Additional observations are needed to determine reliable limits for the process because the actual observations express a change in act and thus they are not suitable to be part of a *reference set.* When *Oscillatory Trend Test* fails, the different sources of variations must be identified, separated and tracked on different charts. When this test fails, the investigation process must be restarted from step1 in order to correctly identify the measurable characteristics to monitor.

c. if a test included in *limit tests class* has failed, the control limits should always be recalculated in order to determine a new *reference set* that expresses the new process performance. The candidate points to be included in the *reference set* are those responsible for the test failure.

4 Approach Validation

The proposed approach has been validated using a combination of validation models, legacy data and simulation [28]. Legacy data are related to a previously completed project whose existing data represent the sequence of events that occurred during project execution. On the other hand, simulation is used to hypothesize how the same real environment would have reacted to the introduction of the new approach. In other words we have simulated the use of our approach on a legacy data set concerning the execution of a renewal project where the induced process improvements are known.

The aim of this empirical investigation is to verify the:

- effectiveness of the *tests set* in discovering at least the induced improvements;
- suitability of the *tests interpretation* given to the events occurred;
- effectiveness of the *Investigation Process* in supporting a software engineer in software process monitoring, understanding and decision making;

As side effects we can also explore if any spontaneous improvements that we don't know, occurred during project execution.

The legacy software system used for technology validation is an aged banking application that needed to be rejuvenated because of its poor quality. The renewal process is made up of two sub processes: reverse engineering and restoration. Greater detail on these two types of processes can be found in [29] [30]. A total of 289 programs were subjected only to reverse engineering and 349 to restoration. All restored programs were first reverse engineered and then restored but, from here on, for clearness we refer to them as "restored programs". A team of four developers renewed the entire legacy system. Several metrics were collected during project execution [31]. For the aims of this work we will consider the following data: NLOC, number of lines of code per program; EFFORT for each program, in man hours.

The data were collected daily by the developers and checked weekly. Data accuracy was monitored during the project because the data were also used to check the quality improvements made and their costs. To control the project risks, the project manager executed the following improvement initiative (induced improvements) in different time points T1, T2 and T3:

1. From the start of the project and during T1, both process characterizations were improved and the decision model for determining which process to carry out for each program was refined;
2. In T2, the tools for the two processes were adopted; in some cases simple tools that were not available on the market had to be developed;
3. In T3, improvements had to be made in formalizing of the reading procedures to check the programs transformed during and after process execution.

The operations carried out on the processes made them unstable. A first useful analysis is the trend of the developers' performance in reverse engineering and restoration, shown in Fig. 6.

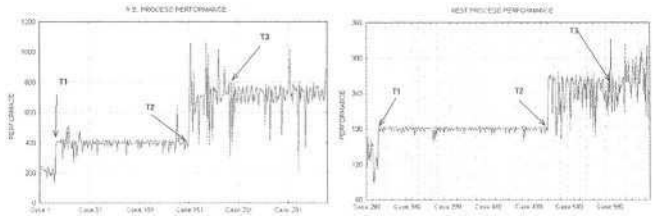


Fig. 6. Reverse engineering and restoration process performance in LOC/Hour

The x-axis refers to the time sequence the programs were subject to either reverse engineering or restoration. The y-axis reports the performances expressed in LOC/hour for each renewed program. Each program to be renewed was subjected to reverse engineering or restoration, according to the project manager's decision. The trend shown in the graphs occurred in the same period for both types of process and was due to continuous improvement of each one. As it can be seen in the graphs, the process performances vary deeply during project execution. We simulated the use of the approach proposed in this paper by applying the step of the *investigation process* as presented in the following. Here, for space reason, only the application of the approach on the reverse engineering process execution will be presented according to the events occurred during the approach simulation, as in Fig. 7.

Recognition of First Induced Improvement (Made in T1). Initially, we determined the measurement object that was the developer performance in renewal process execution, expressed in LOC/Hour. Then we determined the reference set: it was made of the first 15 data points that appeared to be stable in that there were no test failures highlighted. Keeping the control limits and central line fixed, we progressively added data points on the chart until the first *tests set* failure occurred. It was a *three sigma test* failure. According to the *tests interpretation* given for the *sigma tests class* it is an early alarm and therefore may be a false positive. So, we continued adding data points on the charts, according to the *investigation process,* until the first *limits test* failure was detected (Fig. 8). It was a *Run above or below the CL Test* failure. This happened after *two* and *one sigma tests* failures. Thus in this case *sigma tests class* failures have predicted an actual anomalous trend due to the first induced improvement (T1) made to the process (see Fig. 7).

Discovery of Spontaneous Improvement. We determined a new *reference set* made of the 9 data points involved in *Run above or below the CL Test* failure. Data points were progressively added until a new limit tests failure occurred. It was a *Stratification Test* failure (from datapoint 59 to datapoint 73) Fig. 9 which points out a maturity effect after the improvement in T1 (Fig. 7). A two sigma test failure is also highlighted on X chart (Fig. 9) and many other *three sigma test* failures on R. All these failure are false positives in that they don't result in a performance shift.

Recognition of the Second Induced Improvement (Made in T2). According to the *investigation process,* a new reference set was determined from the data points highlighted by *stratification test failures* (59-73). All worked well up to data point 150. Here, several *sigma test* failures on X and mR charts preannounce the second

induced improvement T2 (Fig. 7) that became explicit when a *mixing/over control test* failure occurred on X chart from data point 150 to data point 158 (Fig. 10). From data point 74 until 150 also several sigma test failures occurred on both X and mR chart. They can be considered as false positives in that they don't result in process performance permanent shift until data point 158.

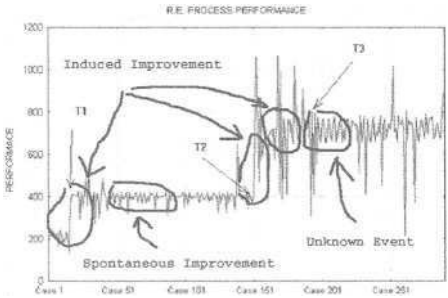


Fig. 7. Spontaneous and induced process improvement

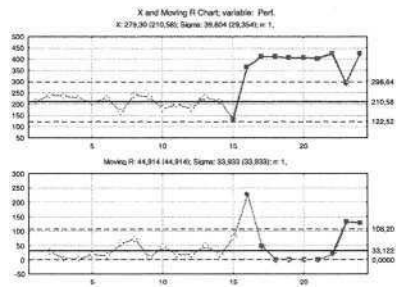


Fig. 8. Limits test failure (datapoints 16-24)

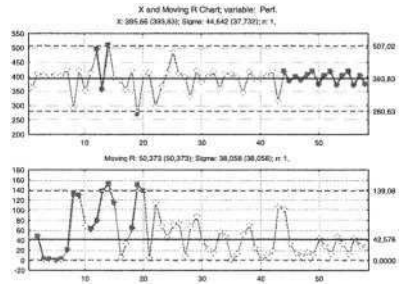


Fig. 9. A spontaneous improvement after T1 pointed out by Stratification test (data points 59-73).

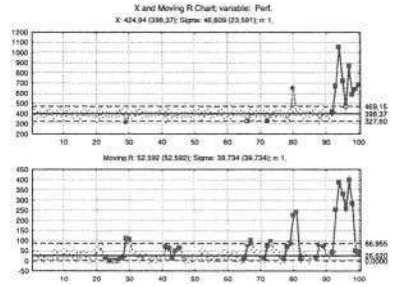


Fig. 10. Mixing/over control test failure (data point 150-158)

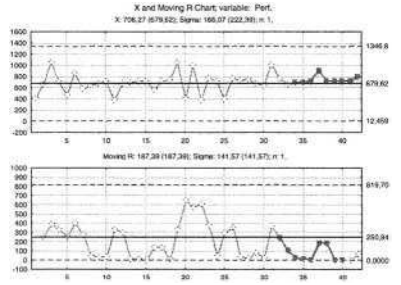


Fig. 11. Run above/below the CL failure (data point 183-191)

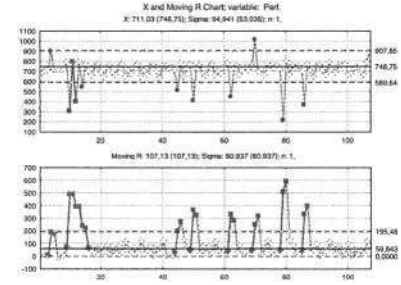


Fig. 12. Linear trend test failure on mR chart (data points 192-199)

Recognition of the Third Improvement (Made in T3). The fourth reference set was determined based on data points 150-158. Starting from data point 159 until 190 there aren't test failures. At the next point, a *run above/below the CL* test fails (Fig. 11). This failure highlights T3, i.e. the third induced improvement (Fig. 7). Induced improvements in this case don't determine turbulent performance variation. No *sigma tests* failure precede the *limit tests* failure. This means that developers adapted to the improvement immediately. Usually this happens by a turbulent change in process performance, highlighted first by several sigma tests failures, then by a *limit/trend tests* failure and finally by the occurrence of a further *limits tests* failures as the result of a maturity effect that prove the innovation acquisition by the developer.

Discovery of an Unknown Event. Finally the fifth *reference set* was determined. It was made of data point 183-191 and was used until the end of the project (data points 192-290). Although many *sigma tests* failures occurred on X and mR charts (Fig. 12), they were false positives. In Fig. 12 note that between data points 192 and 199 a series of *sigma test* failures occur on X chart that show a slow down trend in process performance and a higher variability than expected.

This behavior suggests that an assignable cause is present and solicits its search. Since we are carrying out a post mortem data analysis we are unable to determine the assignable cause behind this behavior. On the same data points interval on mR chart a *trend tests* failure is detected. More precisely, a *linear trend* fails to highlight a decreasing trend. According to the *tests interpretation* given above, this failure highlights an ongoing phenomena. The process is going asymptotically towards a new stability point. Usually this type of test failure is followed by a *limits test* failure when the new point of stability is reached and differs from the previous one. In this case the two stability points (the old and new one) seem to be the same because, after the variations occurred, the performances return to be randomly distributed around the CL and within the limits of the same *reference set.* This means that something strange has happened, although the performance still remains the same.

5 Discussion

The validation, points out that the proposed approach seems to be effective in managing software process improvement.

- The approach was able, by using *tests set,* to point out both, the induced improvements T1,T2,T3 carried out on the process by the project manager, and the spontaneous ones.
- The events occurred during project execution were interpreted consistently according to the *tests interpretation* given.
- The investigation process was as effective as monitoring and decision support tool. It allowed to decide when a new reference set was needed according to process performance shift occurred during the project.

It also confirms the occurrence of several false positives in *sigma tests* failures and thus the correctness of the *sigma tests* interpretation given. In fact, during the validation there were:

- failures that highlighted further shifts in process performance. This happened every time an induced improvement was made;
- failures not resulting in a process performance shift or known assignable causes;

The mR chart was useful in interpreting process variation. For example when the improvement in T2 was made it confirmed the outcomes of X chart (Fig. 10). Furthermore, after the improvement T3, it pointed out the occurrence of an ongoing phenomenon that we are not aware of. The interpretation of the test failure on mR chart allowed waiting for the attainment of a new point of stability avoiding us to determine a new (probably misleading) reference set.

These results have been confirmed during the simulation on the restoration process, which has not been considered here for space reasons.

The proposed approach stresses the use of "run rules" instead of single point failure. Run rules point out trends/systematic patterns that, in cognitive and human centered activities, are more meaningful. Investigation of assignable causes is cheaper and less risky in software processes than in manufacturing ones. In the latter, the production must be stopped and the assignable cause removed in order to minimize the future production losses. This typically implies shutdown followed by start up activities that require extra time and costs. In software contexts the production is not necessarily interrupted. There aren't artifacts that probably will need to be thrown out. They can be subject to rework and be further improved. This primary difference implies less risks and costs in waiting for patterns detection and thus avoids stopping the software development cycle as soon as a single point falls outside the control limits.

6 Conclusion and Future Works

In this work SPC has been applied to carry out time series analysis on data collected during software process execution. SPC is a statistical technique traditionally used for manufacturing process control. Only recently it has been extended to software processes. By analyzing the contributions in literature, we have noticed that software engineering researchers have applied less SPC indicators to software than those used in manufacturing contexts. This contrasts with the fact that software processes are influenced by more state variables than manufacturing ones. Our conjecture is that more indicators are needed to control software rather than manufacturing processes. This conjecture has motivated this work in answering the following three research questions:

- Which are the most suitable indicators for software process control, among those used by SPC?

- How can such suitable indicators be interpreted to support software engineers in managing software process instability?
- How can SPC be used for efficiently supporting software process monitoring and stability investigation?

The answer to the first question has led to the hypothesis of using many of the indicators unknown within the software engineering literature and usually used for controlling manufacturing processes. Such considerations are coherent with the fact that software engineering requires more indicators, due to its higher process diversity. The second answer associates an objective interpretation to each indicator, according to a software engineering point of view. Finally, the third answer identifies an investigation process for software processes, according to the indicators presented in this work and to their meaning.

The method has been simulated through a legacy data study of a large industrial project carried out some time ago. It represents a first validation. The results can be briefly summarized as follows: the proposed approach would have identified all the induced improvements to the processes and their effects on process improvements, just as they occurred during the actual execution of the software application.
Results encourage investigation of the method, followed by its empirical validation. Therefore the authors will carry out further studies and experimentations in order to improve the method according to the obtained results.

References

1. IEEE Software.: Process Diversity. July – August 2000.
2. Basili V.R., Caldiera G., Rombach H.D.: Goal Question Metric Paradigm. Encyclopedia of Software Engineering,Vol.1, John Wiley & Sons, 1994.
3. Shewhart W.A.: The Economic Control of Quality of Manufactured Product. D. Van Nostrand Company, New York, 1931, reprinted by ASQC Quality Press, Milwaukee, Wisconsin, 1980.
4. Shewhart W.A.: Statistical Method from the Viewpoint of Quality Control. Dover Publications, Mineola, New York, 1939, republished 1986.
5. Ebenau R.G.: Predictive Quality Control with Software Inspections. Crosstalk, June 1994.
6. Florac W.A., Carleton A.D., and Bernard J.R.: Statistical Process Control: Analyzing a Space Shuttle Onboard Software Process. IEEE Software, July/Aug. 2000.
7. Florence A.: CMM Level 4 Quantitative Analysis and Defect Prevention. Crosstalk, Feb. 2001.
8. Weller E.F.: Practical Applications of Statistical Process Control. IEEE Software, May/June 2000.
9. Card D., Berg R.A.: An Industrial Engineering Approach to Software Development. J. Systems and Software, vol. 10, pp. 159-168, 1989.
10. Card D., Glass R.L.: Measuring Software Design Quality. Prentice Hall, 1990.
11. Jalote P.: CMM in Practice: Processes for Executing Software Projects at Infosys. Addison-Wesley, 1999.
12. Weller E.: Applying Statistical Process Control to Software Maintenance. Proc. Applications of Software Measurement, 1995.
13. Card D.: Statistical Process Control for Software. IEEE Software, pp. 95-97, May 1994.

14. Florac W.A., Carleton A.D.: Measuring the Software Process: Statistical Process Control for Software Process Improvement. Addison-Wesley, 1999.
15. Weller E.: Applying Quantitative Methods to Software Maintenance. ASQ Software Quality Professional, vol.3, no.1, Dec 2000.
16. Wohlin C., Runeson P., Höst M., Ohlsson M. C., Regnell B., Wesslen A.: Experimentation in Software Engineering: An Introduction. Kluwer Academic Publishers, 2000.
17. Gardiner J.S., Montgomery D.C.: Using Statistical Control Chart for Software Quality Control. Quality and Reliability Eng. Int'l, vol. 3, pp. 40-43, 1987.
18. Zultner R.E.: What Do Our Metrics Mean?. Cuttler IT J., vol. 12. no. 4, pp. 11-19, Apr. 1999.
19. Radice R.: Statistical Process Control in Level 4 an 5 Organization Worldwide. Proc. 12th Ann. Software Technology Conf., 2000.
20. Wheeler D.J., Chambers D.S.: Understanding Statistical Process Control. 2nd ed., SPC Press, 1992.
21. AT&T: Statistical quality control handbook. Indianapolis, AT&T Technologies, 1956.
22. Nelson L.: The Shewhart control chart-tests for special causes. J. of Quality Technology, 15, 1984.
23. Nelson L.: Interpreting Shewart X-bar contol charts. J. of Quality Technology, 17, 114-116, 1985.
24. Grant E.L., Leavenworth R.S.: Statistical quality control. 5th Ed.,New York,McGraw-Hill,1980.
25. Shirland L.E.: Statistical quality control with microcomputer applications. New York,Wiley, 1993.
26. Jalote P.: Optimum Control Limits for Employing Statistical Process Control in Software Process. IEEE Transaction on Software Engineering, vol. 28, no.12, pp. 1126-1134, December 2002.
27. Florac W.A., Park R.E., Carleton A.D.,: Practical Software Measurement: Measuring for Process Management and Improvement. Carnagie Mellon University, 1997.
28. Zelkowitz M.V., Wallace D.R.: Experimiental Model for Validating Technology. IEEE Computer, May 1998.
29. Abbattista F., Fatone G.M.G., Lanubile F., Visaggio G.: Analyzing the application of a reverse engineering process to a real situation. Proceedings of the 3rd IEEE Workshop on Program Comprehension, Washington, D.C.,1994, pp.62-71.
30. Fiore P., Lanubile F., Visaggio G.: Analyzing empirical data from a reverse engineering process. Proceedings of the 2nd IEEE Working Conference on Reverse Engineering, Toronto, Ontario, Canada, 1995,pp.106-114.
31. Baldassarre M.T., Caivano D., Visaggio G.: Software Renewal Projects Estimation Using Dynamic Calibration. Proceedings of International Conference on Software Maintenance -ICSM 2003-, Amsterdam, Holland, September 2003.
32. Paulk M.C.: Applying SPC to the Personal Software Process. Proc. 10th Intl. Conf. Software Quality, October 2001
33. Lantzy M.A.: Application of Statistical Process Control to the Software Process. Proc. 9th Washington Ada Symposium on Ada: Empowering Software Users and Developers, July 1992
34. Duncan A.J.: Quality Control and Industrial Statistics. R.D. IRWIN 5th edition, 1986.

Towards Hypotheses on Creativity in Software Development

Mingyang Gu and Xin Tong

Department of Computer and Information Science, Norwegian University of Science and Technology, Sem Sælands vei 7-9, N-7491, Trondheim, Norway
{mingyang, tongxin}@idi.ntnu.no

Abstract. Though more and more researchers have realized the importance of creativity in software development, there are few empirical studies reported on this topic. In this paper, we present an exploratory empirical research in which several issues on creativity in software development are studied, that is, which development phases are perceived to include more creative work, whether or not UML-based documentation can make developers perceive more time is devoted to creative work, whether or not more creative work can accelerate the software development speed, and whether developers more prefer to do the creative work. Based on result analyses, we proposed four hypotheses to direct the future research in this field and discussed the challenge that 'since developers do not like to participate in those improving activities (quality assuring activities), how can we keep and improve software quality effectively and efficiently?'

1 Introduction

Currently in order to improve industry competing capability and exploit new software application fields, more and more attention is put on the research about creativity in software development which aims to produce novel and high-quality software products.

Robert L. Glass compared two aspects in software development, creativity and discipline, and pointed out software construction was primarily a complex problem-solving activity, which required creativity ultimately [1]. Issues of unpredictability and improvisation in small software organizations were discussed in [2], in which the author pointed out that small software organizations should focus on the relationship between discipline and creativity in software development. In Bring Design to Software [3], Terry Winograd argued that software development was more like the art creation and we should take human characteristics, including creativity, into consideration. Recently, much attention was paid to the human-computer interface and computer-assistant tools in order to promote human creativity [3].

Though the term, creativity, has existed for a long time and is used widely in many fields now, there is not a clear definition about creativity. As summarized by J. Daniel Couger, there are more than 100 definitions about creativity [5]. Some define it as characters of a person [6], some define it as characters of one product or outcome [7], and others define it as processes [7] through which creative people can produce

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 47–61, 2004.

creative results. In this paper, we adopt the definition that creativity is the process, 'that results in novel and useful products [3]'.

Cognitive psychology has provided us theories to explain the generating process of creativity and proposed relevant methods to motivate human creativity. According to the Cognitive Network Model of Creativity [8], people always like to use familiar solutions to resolve the problems they encounter. The way to motivate creativity is to stimulate the potential unfamiliar knowledge in one's mind or connect the concerned knowledge of different people to generate creative solutions.

In business research, people have done much work about creativity. Greg and Anne presented the research status about how employee's creativity-relevant personal characteristics and organizational context characteristics influenced the employee's creative performance, and carried out an empirical research which provided positive evidence that 'the employees who had appropriate creativity-relevant characteristics, worked on complex, challenging jobs, and were supervised in a supportive, non-controlling fashion, produced more creative work [9].'

Since human creativity is thought as the source to resolve complex problem [1] or create innovative products [10], one possibility to improve the software development process is to design a process which can stimulate developers' creativity furthest. However, the problem is more complex than this: 'On the one hand, too much emphasis on individual creativity and independence might create problems in controlling development costs, quality, and long-term maintenance. On the other hand, too much structure and control might stifle creativity, innovation, and ability to change.'[11] So here comes a challenge that 'how should we combine the creativity and discipline in software development?'

Kristian Rautiainen has provided a successful case study about combining flexibility and control in a small company's software product development process [12]. We believe it is useful for generating more general solution to this challenge if we clarify the following questions that whether there are any definite patterns for the creativity to generate or distribute in software development, whether the principles about creativity applied in business research are still effective in software engineering, and what methods or processes can be used to stimulate the developers' creativity to produce novel and high-quality software products, and at the same time can control development cost, quality and long-term maintenance.

But till now, most of the studies about creativity in software development are qualitative or prescriptive, which lack of practical evidences to support industries to make any decision to balance the two aspects in software development: creativity and discipline. In order to start empirical research, we identified four motivational questions on creativity in software development, designed and carried out an exploratory research. The identified motivational questions are listed as following:

- Which phases in software development are perceived to include more creative work compared with discipline-based work?
- Can UML-based documentation increase or decrease developers' perception about the amount of creative work compared with discipline-based work?
- Can more creative work accelerate or decelerate development speed?
- Do developers have distinct inclinations to creative work or discipline-based work?

In this research, we define creative work as the work (process) that are perceived by students to be helpful to generate novel and useful product, for example, the work

being helpful to find new, efficient ideas and methods to resolve the encountered problems in software development. On the contrary, we define discipline-based work as the work demanded by the development process that are perceived by students to be mandatory routines, for example, the complex documents and trivial tests, which are perceived to be useless to produce novel products. The difference between them is that creative work can motivate the generation of something new.

The rest of the paper is structured in the following way: In section 2, we describe the research method used in this research; we reported research context and data collecting methods in section 3 and section 4; we presented the results in section 5; in section 6, the interpretations and implications of the results are analyzed, and four hypotheses are formalized; in the end, we present limitations and future works in section 6, and draw conclusions in section 7.

2 Research Method

This research is based on an educational project in a software architecture course, in which we defined requirements, development phases, mandatory documentation templates, and we divided students into 12 developing groups and asked each group to implement same tasks. And we collected process data during the process of software development and result data through analyzing the final products of each group.

In this research, we could get 'multiple instances of an observation in order to support statistical validity of the results' [13]. At this point, this research could be called as controlled experiment. However, unlike the classical experiment, this research began with identifying several motivational questions instead of formalizing several hypotheses because we could not find any related empirical studies in this field (creativity in software development) to formalize hypotheses, and we ended in four hypotheses as final contributions to guide future research in this field instead of evidence to support or oppose the formalized hypotheses.

Though Martin Host [14] and J. Carver [15] have pointed out that the difference between students and novice professionals is being blurred, we identified one important problem with students experiment that students are lack of process experience compared with professionals [16], and designed a small warming-up project (KWIC, KeyWord-In-Context) to let students to be familiar with the field of software architecture, and remind them of the skills of programming using Java before carrying out the formal project (Mobile Robot) based on which this research is carried out.

3 Research Context

This empirical research is based on an educational project (Mobile Robot) in a software architecture course, in which we also designed and carried out a warming-up project (KWIC) to mitigate students' shortness of experience.

3.1 Software Architecture Course

Software architecture deals with the transition from requirement specification to software design. This phase produces one or more architecture designs, and these designs are assessed to explore defects and risks in them. Through software architecture phase, defects can be found in high level and early stage [17], [18].

This course, in which we carried out this research, was carried out in the spring of 2003 in Norwegian University of Science and Technology (NTNU). It spanned over 13 weeks, and in each week there were 3 hours in class and 9 extra hours for reading and working. Undergraduate students in the 4th year and graduate students in the major of software engineering could select this course. In this term, there were 68 students taking part in this course including 4 PhD students.

In this course, we selected two projects, KWIC and Mobile Robot from the standard example problems in the software architecture community [19]. Both of these two projects were demanded to be implemented using Java, and we used BSCW system to share documents and products besides lectures and face-to-face meetings.

3.2 The Warming-up Project (KWIC)

KWIC (KeyWord-In-Context) was a small project selected with the goal to let students to be familiar with the field of software architecture and remind them of the knowledge of programming using Java.

KWIC project was done by students individually and was divided into two phases: producing phase and evaluating phase. In producing phase, students were demanded to complete the tasks of requirement specification, architecture design, and implementation. In the evaluating phase, students evaluated all the results produced in the producing phase: requirement specification, architecture design and final implementation.

KWIC project lasted three weeks. In the first two weeks, students were asked to complete the producing phase. And then they exchanged documents and programs to evaluate between every two or three students in the third week.

3.3 The Formal Project (Mobile Robot)

Compared to KWIC project, Mobile Robot project was a larger project that was divided into five phases: requirement specification, architecture design, architecture assessment, implementation (programming and testing), and postmortem analysis. We provided documentation templates for each phase to guide students to complete the tasks in that phase. And in requirement specification phase and architecture design phase, students can choose to use or not to use UML diagrams to describe their requirements, and architecture views. We selected the 'WSU Java Khepera Simulator' [20] as the simulating environment for the Khepera Robot [21].

In Mobile Robot project, students were organized into 12 groups randomly as developing units and exchanged semi-finished products between each two groups (illustrated in Fig. 1). Taking the group 1 and group 2 as an example: after requirement specification phase, group 1 and group 2 exchanged their requirement specification documents and continued to complete architecture design task based on

the requirement specification documents they got. The same exchange actions happened again after architecture design phase and architecture assessment phase. Finally after implementation phase, the two groups completed the post-mortem analysis together. This process setting simulated the practical software development in which students acted as different roles interactively: customer, developer, and architect.

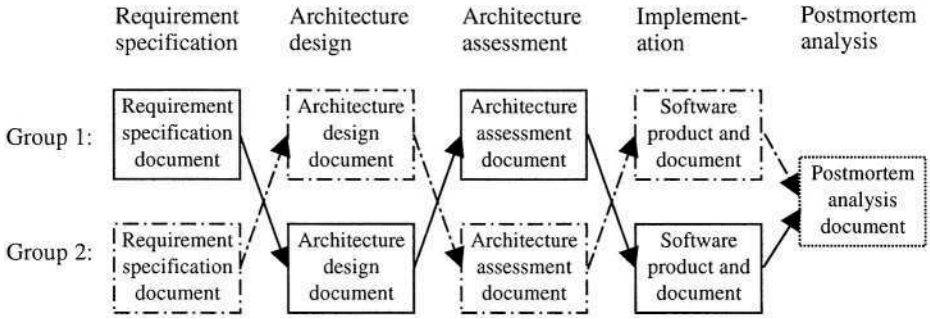


Fig. 1. The process setting to exchange semi-finished product in each pair groups in Mobile Robot project
Note: The dash-dotted units and the real line units present two product lines with different phases to be completed by different group. Arrow means the direction to exchange semi-finished products

Mobile Robot project lasted 6 weeks. Each phase used one week except for implementation phase which used two weeks.

4 Data Collecting Method

In order to study the first three motivational questions, 'which phases in software development are perceived to include more creative work compared with discipline-based work', 'can UML-based documentation increase or decrease students' perception about the amount of creative work compared with discipline-based work', 'can more creative work accelerate or decelerate development speed', we need to measure the amount of creative work and discipline-based work in each phase. In this research, we used creative time (used to do creative work) or discipline-based time (used to do discipline-based work) to quantify the amount of creative work or discipline-based work. In addition, 'other time' is used to do the work which can not be classified as creative work or discipline-based work by students. All these three types of time are come from students' subjective perception. We put the following data collecting questions into documentation templates for each phase:

- Creative time
- Discipline-based time
- Other time

In addition, we also asked students to provide the necessary explanation to each type of time, that was, what they really did using their reported time.

In order to study the last motivational question that 'do developers have distinct inclinations to creative work or discipline-based work', we designed a questionnaire to collect students' subjective data at the end of this course. 36 students contributed to the results (there were 36 students attending the class when we handed out the questionnaire).

All these materials (templates and questionnaire) and results are available on the website (http://www.idi.ntnu.no/~mingyang/research/results/spr2003exp).

5 Results

5.1 The Creative Time and Discipline-Based Time in Each Phase

In order to express the relationship of creative work and discipline-based work, we calculate two indicators to each student in each phase: Percentage of Creative Time (PCT) and Percentage of Discipline-based Time (PDT) using Formula 1, 2:

$$PCT = \frac{CT}{CT + DT} \tag{1}$$

$$PDT = \frac{DT}{CT + DT} \tag{2}$$

Note: CT and DT in the formulas mean creative time and discipline-based time respectively to each student in each phase

We use relative value (percentage) instead of absolute value of creative time and discipline-based time because of the following two considerations:

- In each phase, the amounts of used time differ widely from student to student, so if we calculate statistical results based on absolute data, some valuable clues hidden in the data from many short-time-used students will be obscured by the data from several long-time-used students.
- Every practical project has its limited delivery time, so the meaningful and important thing is how to divide and keep the balance of creative time and discipline-based time, not absolute time in each category.

Box-and-whisker plot (box-plot) is used to express the data about this issue. Box-and-whisker plot is based on the statistics of median, quartiles and extreme values. In this chart, box is used to represent the inter-quartile range which contains the 50% of values. The whiskers are lines that extend from the box to the highest and lowest values, excluding outliers, which are expressed using small cycles. A line across the box indicates the median.

Since the sum of PCT and PDT equals to 1, we need only focus on one of them in the result analysis. In this paper, we choose PCT as indicator to show the relationship between creative work and discipline-based work.

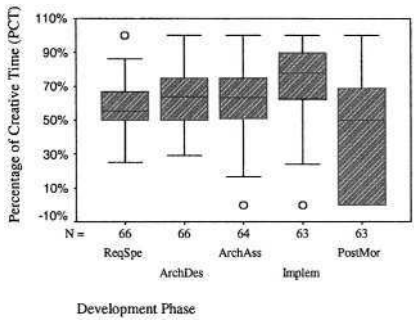


Fig. 2. Box-and-whisker plot about the values of PCT (Percentage of Creative Time) to each development phase

As shown in Fig 2, according to the values of PCT, development phases can be classified into three categories: the category with highest values of PCT only includes implementation phase; the category with lowest values of PCT just includes postmortem analysis phase; and requirement specification phase, architecture design phase and architecture assessment phase are in the same middle category.

The results tell us that compared with discipline-based work, implementation phase is perceived by students to include most creative work, post-mortem phase is perceived to include least creative work , and the amount of creative work in requirement specification phase, architecture design phase and architecture assessment phase are perceived at the same middle level.

5.2 The Influence of UML-Based Documentation on the Amount of Creative Time in Software Development

In requirement specification phase, there are 4 groups use use-case diagrams to describe their requirements, while others use free-text-based document. And in architecture design phase, 11 out of 12 groups use UML-based diagrams to describe architecture views, and the only group who dose not use UML-based diagrams defines their own diagrams to describe architecture views.

In order to illustrate whether UML-based documentation influence students' perception about how much time is used to do creative work compared with that used to do discipline-based work, we calculate the values of PCT according to Formula 1 to each student and each phase of requirement specification and architecture design, and then we classify all students into two categories in each phase according to whether or not the student uses UML-based diagram in documentation in that phase. The results are reported in Fig. 3 using box-and-whisker plot.

The figure shows us that the values of PCT for students who use UML-based diagram in their documents are higher than those of the students who do not use UML-based diagram in their documentation, especially in the requirement specification phase, which means that UML-based documentation can cause the students to perceive more time is devoted to creative work.

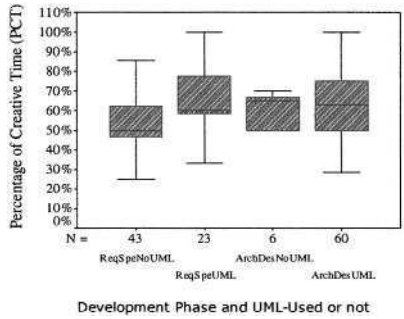


Fig. 3. Box-and-whisker plot about the compassion of PCT values between UML-Used students and none UML-Used students in requirement specification phase and architecture design phase

5.3 The Relationship between Total Used Creative Development Time and the Total Used Development Time

In order to illustrate whether the amount of creative work has distinct influence on development speed, we calculate the following indicators to each group:

$$TCT = \sum_{first\_4\_phases} \sum_{all\_students\_in\_group} CT \tag{3}$$

$$TDT = \sum_{first\_4\_phases} \sum_{all\_students\_in\_group} DT \tag{4}$$

$$TOT = \sum_{first\_4\_phases} \sum_{all\_students\_in\_group} OT \tag{5}$$

$$TT = TCT + TDT + TOT \tag{6}$$

$$TPCT = \frac{TCT}{TCT + TDT} \tag{7}$$

Note: TCT, TDT, TOT, TT, and TPCT mean Total used Creative development Time, Total used Discipline-based development Time, Total used Other development Time, Total used development Time and Total Percentage of Creative Time in the first four phases (requirement specification phase, architecture design phase, architecture assessment phase and implementation phase) to each group.

The reciprocal of TT (Total used development Time) can be seen as the development speed. And the relative amount of creative time is expressed by Total Percentage of Creative Time (TPCT) calculated using Formula 7.

So in Fig 4, we illustrate the changing trend of TT following the increase of TPCT to show whether there is distinct relationship between the amount of creative work and development speed.

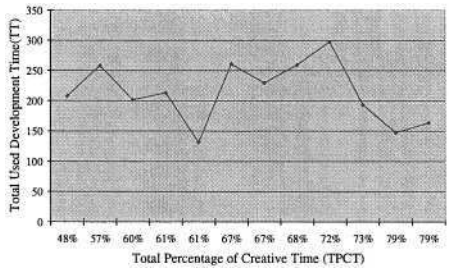


Fig. 4. Relationship between Total used development Time (TT) and Total Percentage of Creative Time (TPCT)

Fig. 4 shows us a intuitive impression that following the increase of TPCT (from 48% to 79%) there is no distinct changing trend for TT, which seems to tell us that the total used creative development time has no influence on the development speed (the correlation coefficient between TPCT and TT is only -0.16).

5.4 The Students' Perception about Creativity in Software Development and Their Preference to Different Development Phases

The data in this sub-section come from the questionnaire. Here, we use the data about students' perception about what activities should be classified as creative work and what activities should be classified as discipline-based work (question 1), and their personal preferences to each phase (question 2).

Question 1 in Questionnaire (with partial options)
Do you think the following activities belong to discipline-based work or creative work? (Discipline-based/Creative)

Completing document with demanded format	D/C
Architecture design	D/C
Architecture assessment (ATAM)	D/C
Programming	D/C
Testing	D/C

Question 2 in Questionnaire
What software engineering phases do you like and what phases do you dislike? (Multiple choices)

The ones you like:_____	The ones you dislike:	
①. Requirement specification.	②. Architecture design.	③. Architecture Assessment.
④. Implementation.	⑤. Testing.	⑥. Maintaining.
⑦. Post Mortem.		

The results of question 1 are shown in Fig. 5, from which we can see that more students perceive architecture design (97%) and programming (81%) as creative work, while fewer students classify completing documents with demanded format (33%), architecture assessment (44%) and testing (19%) as creative work.

Question 2's results are presented in Fig. 6, which gives us the impression that most students like requirement specification phase (72%), architecture design phase (91%) and implementation phase (81%).

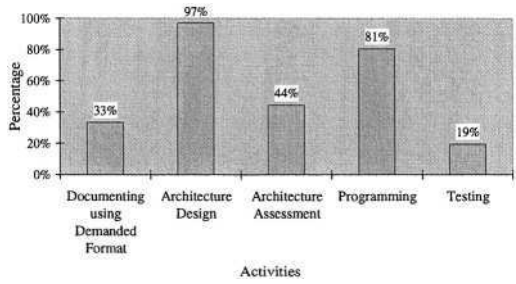


Fig. 5. Percentage of students who perceive each activity as creative work

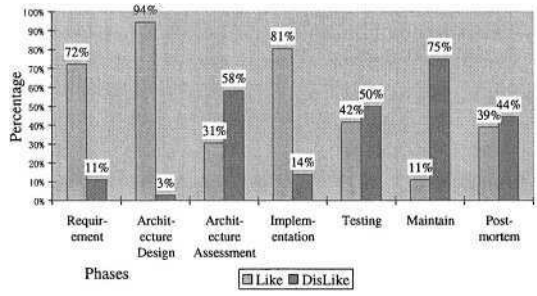


Fig. 6. Percentage of students who like or dislike each phase in software development

6 Results Analysis and Hypotheses Formalization

From the results presented in the previous section, we can see the statistical trend to each issue. In this section, we will provide the interpretation for these results, discuss their implications, and propose hypotheses for the future research.

6.1 The Relationship between Creative Work and Discipline-Based Work in Each Phase

From Fig 2, we can find that compared with the time used to do discipline-based work, students perceive most time is devoted to creative work in implementation phase; in post-mortem phase, most time is used to do discipline-based work; and the time used to do creative work are at the middle level in requirement specification phase, architecture design phase and architecture assessment phase.

We can formalize this finding into the following hypothesis:

- H1: In software development, there is most creative work in implementation phase and least creative work in post-mortem analysis phase.

One possible interpretation for this hypothesis is that students feel the real meaningful thing is implementing software product, and all the documents and

diagrams during development process are done for the teaching staff, especially the post-mortem analysis work which can never help to generate software product, though the documents during the requirement specification phase, architecture design phase and architecture assessment phase can help them clarify their thoughts about software design (which is supported by the result of question 1 in questionnaire partially).

6.2 The Relationship between UML-Based Documentation and the Amount of Creative Work

Fig. 3 gives us a trend that both in the requirement specification phase and architecture design phase, the students who used UML diagrams in their documentation perceived that more time was devoted to creative work than those who did not use UML diagrams in their documents.

So a new hypothesis can be formalized:

- H2: UML-based documentation can promote students to do more creative work in requirement specification phase and architecture design phase.

The justification for this hypothesis is that UML diagrams provide students a set of tools to present their ideas about requirement specification and architecture design, which reduces the time used to express their ideas using text, or searching and learning other tools to describe them.

6.3 The Influence of the Amount of Creative Work on Development Speed

Fig. 4 illustrates that relative amount of creative work in software development has no distinct influence on the development speed, that is, if we improve the relative amount of creative work, we can not accelerate or decelerate the development speed.

So we formalize a new hypothesis based on this analysis:

- H3: More creative work compared with discipline-based work can not accelerate or decelerate development speed.

6.4 The Developers' Perception on Creativity in Software Development and Their Preference to Different Development Phases

According to Fig. 5, if we classify the activities in question 1 into two categories: producing activities, including architecture design and programming, and improving activities (quality assuring activities), including completing document with demanded format, architecture assessment and testing, the results tell us that students perceive there are more creative work in producing activities than in improving activities.

Such result is accordant with the data shown in Fig. 2 to some degree. The architecture design phase (producing architecture design) and implementation phase (producing the final software) include much producing activities and are perceived by students to contain more creative work. On the contrary, architecture assessment and

post-mortem phase include more improving activities and are perceived to contain less creative work.

Furthermore, from Fig. 6 we can see that more students like the phases (requirement specification, architecture design, implementation) that include more producing activities than those (architecture assessment, testing, maintain and post-mortem) that include more improving activities.

Based on the results of question 1 and question 2, we can conclude that the phases students more like are the phases which they perceive include more creative work. That is to say, they like the requirement specification, architecture design and implementation phases because these phases contain more creative work, and they do not like architecture assessment, testing, maintain and post-mortem phases because these phases include too much discipline-based work.

A new hypothesis can be formalized here that:

– H4: Students prefer the phases which include more creative work.

Previous discussions give us a clue that people tend to create something (producing activities), and do not like to test, improve or maintain something (improving activities). This clue gives us a new perspective to explain the emergence and flourish of OSS (Open Source Software) in which participators enjoy designing and programming, and push the testing and evaluating tasks to users of software products.

Generally, people always intend to pay much attention and energy into what they like to do, and always can do it much better than what they do not. And we also know that standardized documentation, architecture assessment, testing, maintaining, and post-mortem analysis play important role to control and improve software quality. So here comes a challenge 'since developers do not like to participate in these improving activities (quality assuring activities), how can we keep and improve the quality of the software product effectively and efficiently?'

In software engineering community, there have been mechanisms to resolve this challenge. One traditional method is to set up a series of mandatory, detailed and easily manipulative evaluating and testing rules or routines, and to provide a set of restrict documents or template to keep or improve software quality.

The other method to solve this challenge is trying to reduce possible errors during producing activities so that the workload of improving activities can be reduced. For example, in XP (eXtreme Programming), there are three rules concerning this topic: the first is 'pair programming', which can motivate pair programmers to find out or avoid as many errors as possible in the process of implementation; the second rule is 'testing first', which asks developers to pay much attention to testing before or during the implementation, the last rule is 'refactoring', which can find errors whenever or wherever possible in whole project life cycle.

OSS is another mechanism to solve this challenge. In our opinion, OSS can be seen to generate a work division in software development. It divides all the tasks of software development into two parts: designing plus implementing (producing activities) and evaluating plus testing (improving activities). The designing and implementing tasks are charged by developers, and the evaluating and testing tasks are managed by users. Developers complete their favorite producing activities to create new designs and programs, while users make use of the software product for their own benefits and carry out testing and evaluating tasks. So both developers and

users 'enjoy' their work to complete both producing activities and improving activities.

7 Limitations and Future Works

As discussed in the research method section, we designed and carried out a warming-up project, KWIC, to mitigate the inherent shortcoming about student-based experiment that students are lack of process experience compared with professionals. There are other limitations about this experiment.

Since this research is based on an educational project, we have to take into account the pedagogical goals of this course or the benefits of students and teaching staff [14], [22], some of which are inconsistent with the research goals. Concretely speaking, we put much energy and time into architecture phase, that is, we used two weeks to do architecture related work in the total 6 weeks in order to let students experience the process of architecture design and architecture assessment. And we combine the system design, programming and testing into one phase named implementation in order to avoid overmuch burden on students, but the Fig. 5 tells us that students feel there is more creative work in design (architecture design) and programming, while there is more discipline-based work in testing. So we can get more detailed data if we separate the implementation phase into three phases: detailed design, system implementation and system test.

Though we have explained to students that all data about the creative time and discipline-based time are used for an academic research, and have nothing to do with their own scores in this course, students may incline to report more time on each item, more creative time or less discipline-based time than reality to show that they are working hard and efficiently, because they know the staff of this course and the research executants are the same people and believe this submitted data about research topic have some influences on the evaluations of their schoolwork.

As the future works, we will design and carry out experiments to test these formalized hypotheses, in which above identified limitations will be taken into account. Furthermore, we are going to introduce two other parameters: developer's experience and software quality into new experiments. As to the developer's experience, we plan to observe whether the developer's experience has distinct influence on the amount of creative work or discipline-based work. As to the software quality, we plan to observe whether there are distinct relationship between the amount of creative work and the different aspects of software quality, such as performance, functionality, reliability, and usability.

8 Conclusions

In this paper, we presented an exploratory empirical research on creativity in software development to study the following four identified motivational questions:

- Q1: Which phases in software development are perceived to include more creative work compared with discipline-based work?

- Q2: Can UML-based documentation increase or decrease students' perception about the amount of creative work compared with discipline-based work?
- Q3: Can more creative work accelerate or decelerate development speed?
- Q4: Do students have distinct inclinations to creative work or discipline-based work?

Based on the result analysis, we formalized four hypotheses according to each of the above motivational question:

- H1: In software development, there is most creative work in implementation phase and least creative work in post-mortem analysis phase.
- H2: UML-based documentation can promote students to do more creative work in requirement specification phase and architecture design phase.
- H3: More creative work compared with discipline-based work can not accelerate or decelerate development speed.
- H4: Students prefer the phases which include more creative work.

In addition, we provided a new perspective to explain the emergence and flourish of OSS, in which the participators enjoy producing activities and push improving activities to the users of the products. We also brought forward one challenge that 'since developers do not like to participate in these improving activities (quality assuring activities), how can we keep and improve the quality of the software product effectively and efficiently?', and discussed three mechanisms to resolve this challenge. In the end, we discussed the limitations of this research and planed our future work.

References

1. Glass, Robert L. (1995). "Software Creativity", Prentice Hall.
2. Tore Dybå, (2000). "Improvisation in Small Software Organizations." IEEE SOFTWARE 17(5): 82 - 87.
3. Winograd, Terry (1996). Bring Design to Software', Addison Wesley, Addison Wesley.
4. (2002). SPECIAL ISSUE: Creativity and interface. Communications of the ACM. New York, NY, USA, ACM Press. Volume 45: 89 - 120.
5. Couger, J. Daniel (1989). Ensuring Creative Approaches in Information System Design, Center for Research on Creativity and Innovation.
6. Gough, Harrison G. (1979). "A creative personality scale for the adjective checklist." Journal of Personality and Social Psychology: 1398-1405.
7. Amabile, T.M. (1988). A model of creativity and innovation in organizations. Research in organizational behavior 10: 123-167.
8. Eric L. Santanen, Robert O. Briggs and Gert-Jan de Vreede (2000). The Cognitive Network Model of Creativity: a New Causal Model of Creativity and a New Brainstorming Technique. International Conference on System Science, Hawaii.
9. Gerg R. Oldham, Anne Cummings (1996). "Employee Creativity: Personal and Contextual Factors at Work." Academy of Management Journal 39(3): 607 - 634.
10. Humphrey, Watts S. (1987). Managing for Innovation: Leading Technical People, Prentice Hall Trade.
11. Cusumano, M. (1991). Japan's Software Factories, Oxford University Press.

12. Kristian Rautiainen, Lauri Vuornos, Casper Lassenius (2003). An experience in Combining Flexibility and Control in a Small Company's Software Product Development Process. International symposium on empirical software engineering, Rome.
13. Marvin. V. Zelkowitz, Dolores R. Wallace (1998). "Experimental Models for Validating Technology." IEEE Computer 31(5): 23-31.
14. Martin Höst, Björn Regnell and Claes Wohlin (2000). "Using Students as Subjects - A Comparative Study of Students and Professionals in Lead-Time Impact Assessment." Empirical Software Engineering 5(3): 201 - 214.
15. J. Carver, L. Jaccheri, S. Morasca, F. Shull (2003). Using Empirical Studies during Software Courses,. Empirical Methods and Studies in Software Engineering _ experiences from ESERNET, Springer: 81-103.
16. Jeff Carver, Forrest Shull, Victor Basili (2003). Observational Studies to Accelerate Process Experience in Classroom Studies An evaluation. International symposium on empirical software engineering, Rome.
17. Bosch Jan (2000). Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach, Addison-Wesley.
18. Len Bass, Paul Clements, Rick Kazman (1998). Software Architecture in Practice, Addison-Wesley.
19. Mary Shaw, David Garlan, Robert Allen, Dan Klein, John Ockerbloom, Curtis Scott and Marco Schumacher (1995). Candidate Model Problems in Software Architecture. Discussion draft 1.3 in circulation for development of community consensus.
20. Wright State University, "WSU Java Khepera Simulator Home Page", http://ehrg.cs.wright.edu/ksim/ksim.html, 2003
21. K. Team, "Khepera Page", http://www.k-team.com, 2003
22. J. Carver, L. Jaccheri, S. Moraso and F. Shull (2003). Issues Using Students in Empirical Studies in Software Engineering Education. IEEE International Software Metrics Symposium, Sydney, Australia.

Using Software Inspection as a Catalyst for SPI in a Small Company

Lasse Harjumaa[1], Ilkka Tervonen[1], and Pekka Vuorio[2]

[1] Department of Information Processing Science,
University of Oulu
P.O. Box 3000,
90014 OULUN YLIOPISTO
{lasse.harjumaa, ilkka.tervonen}@oulu.fi
[2] Buscom Oy, Elektroniikkatie 4
90570 OULU
pekka.vuorio@buscom.fi

Abstract. Process improvement activities in small and medium size enterprises (SME) are challenging due to small number of personnel and projects, people have to perform in a variety of roles. Assigning process improvement activities to the overburdened personnel may be seen as a threat to ongoing projects. Both management and staff should become convinced of the benefits of the improvement actions before taking the first step of the process. Even in this situation the software inspection provides a tempting starting point for process improvement. It is a focused and well defined subprocess and enables high return on investment benefits even in short period use. Our experiment in a small software company confirms that software inspection provides the justified starting point for process improvement. By means of the inspection maturity model the company recognises the weak points in their review practice and inspection patterns help in discovery of improvement actions.

1 Introduction

Several methods and tools exist for determining and improving the quality assurance function in software organizations. The most widely used are CMMI, ISO9001 and SPICE. However, small companies have difficulties in applying these models in their full extent [1]. SPI is still possible, but it requires simplification of the improvement models. [2] In small companies, all SPI activities have to be weighted differently: the goal is not to achieve certain level of maturity, but to concentrate on the real quality and business goals of the company. [3]

Software inspection is amongst the most effective methods when evaluating the return on investment of different quality assurance techniques [4, 5]. Inspections are useful from several viewpoints: In addition to clean-up and prevention of defects, they serve as training method [6] and valuable data source for knowledge capturing and management [4, 6]. The inspection process even provides means for process measurement and management [7, 8], and can enhance team collaboration and communication [4].

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 62–75, 2004.

The original software inspection process, as described by Fagan [9], or its later variations are very rigorous. There may be occasions when a customized or slightly restricted version of the process is more applicable. For example, the inspection team may be geographically scattered, or the organization may have limited resources for running an exhaustive inspection process. Even though these more flexible forms of inspection exclude certain details of the traditional processes, they also are capable of producing excellent results by the means of product and process quality. Furthermore, inspections can be carried out to all types of software artifacts, including requirements and design documents. [10, 11]

Being such an effective method, software inspection provides a feasible starting point for small companies for improving their software production processes. The goals of process improvement in small organizations are probably different from those in large ones. While the main focus in large companies is to achieve better efficiency and reliability at lower cost, small companies aim at controlling their growth and fighting chaos.

[12] suggests that implementing a limited set of well-defined quality assurance processes is the most beneficial SPI approach for a small company. A process should be a tool for producing the software, not a purpose in itself. Processes should also be tailored for the needs of the organization.

From inspection process improvement viewpoint none of the general or testing tailored models is enough oriented to inspection evaluation. The really justified improvement suggestions are also rare, which comes from the laborious data collection, analysis and interviews required for them. Due to these reasons we in this paper introduce a light capability model tailored especially to inspection process evaluation that looks for weak points through indicators and base practices. The model is called i3GO (improved inspection initiative group in Oulu) capability model, and it provides a skeleton for semi-formal interviews and thus help in the planning and implementation of evaluation.

The main idea of the tailored capability model comes from BOOTSTRAP [13], i.e. we look for the base practices of the inspection process and evaluate the grade of each practice in the company/division/project (whether it is in use totally or partially). The evaluation is based on indicators, which can be enabling or verifying ones. The idea is that, if we find indicators of a base practice, we then have some justifications for assuming the existence of that base practice. If some practices are at a low level or missing, they should be targets for inspection process improvement, usually based on discussions with company staff. We also have generated a preliminary set of improvement patterns by means of which companies can find improvement activities more easily.

We will first present some rationales behind the model, and then introduce the software inspection process with the indicators and base practices. We will then explain the usage of the model and improvement patterns in more detail and go on to present some further research ideas.

2 Background of the Model

There are a number of process capability determination models in the area of software engineering, including tailored testing models such as the Testing Maturity Model

(TMM) [14] and Testability Maturity Model [15]. From an inspection process improvement viewpoint none of the general or testing models is sufficiently oriented towards inspection evaluation, although BOOTSTRAP and SPICE [16] allow focused process assessment and improvement, i.e. they allow one to choose the process to be improved (e.g. review/inspection. Proper reporting of inspection improvement initiatives has also been rare, although there are some good examples e.g. at Bull NH Information Systems [8] and Hewlett Packard [17], which focus on the analysis of current status.

Most software development organizations in Finland are relatively small, or at least they are divided into rather autonomous units. To stay competitive, it is important that they upgrade and continuously update their software production processes. However, the most popular software process improvement models have a number of deficiencies where the small companies are concerned – full-sized improvement projects are far too costly and time-consuming. [18, 19, 20]

Software process improvement models and tools should be adapted for the needs of SMEs. [21], for example, suggests a computerized tool, which aims at speeding up the improvement and reducing the amount of necessary resources. Another approach is to focus the SPI on the most crucial parts of the development. By taking one clearly defined practice, the software inspection, into the focus of the improvement, several advantages may be gained:

- An organization can "practice" the SPI in manageable scale. Later, a full-sized SPI programme can be launched.
- Benefits of the inspection on product quality are considerable and measurable. This facilitates management and personnel commitment to the SPI effort.
- Inspection has positive side-effects on the process: improved communication and knowledge transfer.
- Single quality practice – especially one defined as rigorously as inspection – is easier to understand and manage when introducing substantial changes to it.

Software inspections are gaining increased acceptance in the software engineering industry. Even though industry experience shows that inspections reduce cycle time, lower costs, increases process visibility, improves programmers' capability and, of course, improve quality by preventing defects [9, 6, 22], the inspection process has not gained widespread usage according to the Software Engineering Institute [23]. To promote the utilization of inspections, we suggest inspection improvement patterns for easy installation and usage of the process.

The use of abridged and focused assessments to launch SPI has been successfully experimented and reported in [24], Focused assessments and SPI activities can also be applied to a single project within a large organization to find out the maturity of a particular development effort. Focused interviews can be used to determine the current maturity [24]. For the best results, these interviews should be carried out in a short time frame, have participants that possess adequately broad views on the development process and concentrate on achieving consensual opinion about the current state of the project or organization.

According to [25], an improvement model aimed at small companies should focus on the software processes that are most important to it, and provide fast return on

investment. Inspections can provide that. Richardson also states that an improvement model have to be flexible and easy-to-use. [25]

Human factors are emphasized in small organizations. They are more dependent on individuals, as developers usually become experts on certain domains. Expert knowledge is rarely documented and thorough documentation is probably not required at all in an immature process. Due to small number of employees and projects, people have to perform in a variety of roles. Assigning additional SPI activities to the personnel may be seen as a threat to ongoing projects. Furthermore, development processes and tools will more likely change more often than in large organizations. Processes have to be adaptable to the specific requirements set by each customer. [26]

Due to the limited resources and divergent environment, the model is not allowed to inhibit the creativity and dynamics of projects by enforcing arbitrary or artificial procedures, but provide the essential quality operations to help the projects to concentrate on the most important details [26]. To sustain the improvement, the SPI assessment and improvement activities should also be repeatable at a reasonable effort.

Generic process improvement models typically describe the ideal outcome of the improvement actions in adequate detail, but do not provide exact guidance on how to proceed in the improvement. For example, [27] report that even though persons involved in SPI understand what need to be improved, they require more specific guidelines about how to conduct the improvement. Instead of discussing the processes and assessment items at a generic level, the improvement model should deal with concrete and well-known issues that are relevant to the company.

We introduce a pattern approach to direct the implementation of the inspection improvement activities. Patterns describe general solutions for frequent problems. In software engineering, design patterns for object-oriented development have been widely used and recognized as a valuable add-on in program design and implementation. [28] In addition, patterns have been applied to software processes. For example, [29] represents pattern approach for software process improvement to overcome some typical difficulties. Patterns have also been applied to capture and refine the software engineering knowledge into reusable form [30].

Patterns also allow adaptation of the SPI implementation for each organization. Such tailorability is a significant factor in successful improvement and reduces natural resistance against change [31]. Patterns are concrete and manageable descriptions of improvement actions needed in various situations.

Finally, the inspection improvement model should take into consideration generic, full-scale process improvement models. The i3GO model attempts to support the company in its course to a greater level of maturity, and if the company is interested in outside certification, such as ISO, the i3GO improvement model must not hinder, but promote that. Compliance can be achieved by using similar terminology and concepts as in generic models. Furthermore, the inspection process is not a separate function in a development cycle. Inspections cannot be truly capable, if there are not other quality assurance activities in use in the organization. For example, metrics that are gathered in inspections have to be defined, managed and reported. Thus, the improvement model should encourage and prepare the organization to improve other development and management processes as well.

3 The i3GO Model

The i3GO model is based on the ideas and structure of the Bootstrap methodology [13]. The model forms the basis for the software inspection process assessment by defining the process as precisely as possible, yet allowing company-specific tailoring of the process.

Inspection is traditionally defined in terms of steps such as entry, planning, kickoff meeting, individual inspection, logging (inspection) meeting, edit, follow-up, exit and release [6, 9]. The structure of the ideal process of inspection which we use as a reference model in evaluation is based on these steps.

Although the ideal process is defined in the BOOTSTRAP model as a set of base practices, our interpretation of these differs in that we also include organisational and supporting activities among them. The discovery of base practices has been guided by six specifically defined goals:

1. to identify defects in an artifact,
2. to estimate the quality of an artifact,
3. to improve product quality,
4. to provide data for process improvement,
5. to provide the means for knowledge transfer, and
6. to improve the effectiveness of the development process.

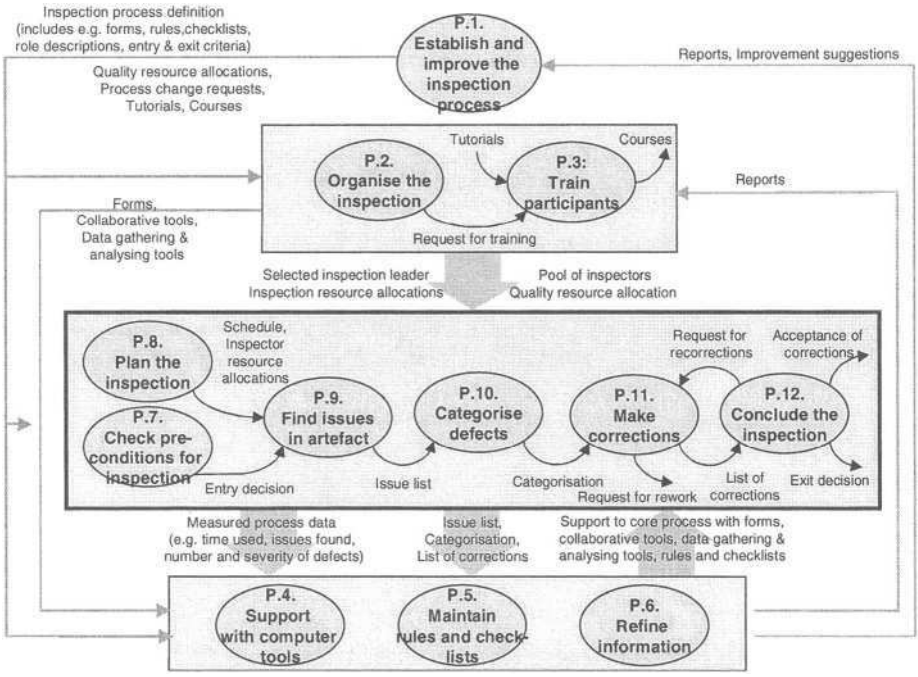


Fig. 1. The i3GO base practices and indicators.

These practices can be classified into three groups (cf. Figure 1): organisational activities (at the top), that ensure continuous improvement and efficient organisation of the inspection process, the core set of activities (the middle bar), that are the essence of any implementation of the inspection process (as defined in inspection books, e.g. [6]), and supporting activities (at the bottom), that help in carrying out an instance of the inspection process.

The organizational activities are: P.1. Establish and improve the inspection process, P.2. Organise the inspection and P.3. Train participants. Work products, reports and other material related to these activities are represented in the figure by arrows. The core activities are P.7. Check the preconditions for inspection, P.8. Plan the inspection, P.9. Find issues in the artifact, P. 10. Categorise defects, P.11. Make corrections and P. 12. Conclude the inspection. Finally, the supporting activities are P.4. Support with computer tools, P.5. Maintain rules and checklists and P.6. Refine information.

Figure 1 also depicts data flows between activities. According to the BOOTSTRAP model, the data flows are used for evaluating the existence of a base practice, and we thus call them enabling and verifying indicators. The capability model has been experimented in five software organizations, and it has been refined based on the usage experiences. Experiments are described in more detail in [11].

Base practices and indicators form the skeleton of the capability model. For example, the base practice P.9, Find issues in the artifact, is one of the core activities and serves all six goals of the model. Table 1 lists the indicators related to the practice.

Table 1. Enabling and verifying indicators.

Possible enabling indicators	Possible verifying indicators
Forms	Measured process data
Rules	Issue list
Checklists	
Schedule	
Inspection resource allocation	

The indicators are evaluated and the existence (and grade) of the base practices can be analysed and justified in the light of the results. For example, “issue list” points to the potential defects in the artifact. List should be sent to the author, and each issue in the list should include at least information about its location, severity and checklist reference. If any crucial information is missing, the grade will be lower. The grading depends on the needs of the organization. The process has not to be “ideal” in every occasion. In a small company, some practices may be rather informal, but they are still adequate.

According to our experiences in the industry, company representatives were keen to discuss the potential improvement actions already during the assessment meetings. For this purpose, the outcome of the assessment is recorded into a spreadsheet-based tool, which calculates and represents the result of the evaluation instantly.

Furthermore, concrete guidelines for updating the process have to be at hand immediately after the assessment. These are provided by means of improvement

patterns, which describe the most typical improvement strategies. The next chapter discusses the assessment and improvement parts of the model in more detail.
Assessments and Improvement with the i3GO Model

The inspection process improvement with i3GO model starts with an assessment, which is carried out during an interview session. It immediately shows the results of the evaluation. After that, overall goals for the improvement are set, and finally a set of actions are performed to gain the improvement. The goal-setting and actual improvement procedures are aided by the improvement patterns.

3.1 Determining the Capability

Capability determination is based on checking of the enabling and verifying indicators. An enabling indicator confirms that the preconditions for a specific base practice are met, and the absence of such an enabling indicator strongly suggests that the corresponding base practice does not exist. A verifying indicator confirms that a specific base practice has produced appropriate and adequate results, and its existence suggests that the corresponding base practice may exist, but does not guarantee this.

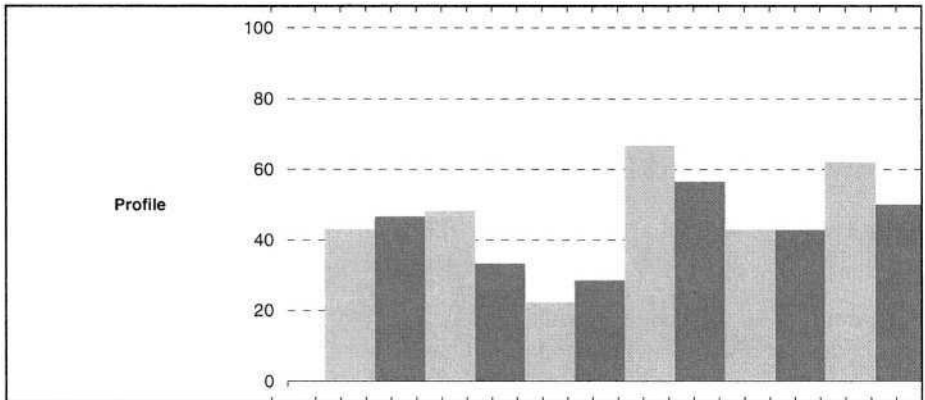
The matrix for capability evaluation is presented in Figure 2. The first version of the model included 35 indicators, but according to our experience in four cases we removed some indicators and added new relationships. Now there are 29 indicators to be walked through during an evaluation. The goal was to avoid overlapping indicators but to retain an essential set of indicators for determining the capability of base practices. The enabling and verifying types of indicators are described with letters E and V, correspondingly.

The existence of an indicator is evaluated on a five-grade scale:

na	not relevant to the organisation
0	not in use at all or only rarely
1	partially, exists to some degree
2	largely, exists fairly well
3	fully, always in existence

As we can see from the matrix, one indicator can affect a number of base practices. In the evaluation session we walk through all the indicators. They are grouped according to core, supporting and organisational base practices, the first 10 indicators applying to the core set of base practices, indicators 11-20 being related to all groups of base practices and indicators 21-29 being mainly for organisational base practices. This order of indicators is justified because it is easier to start the assessment session by evaluating core activities and to go on to supporting and finally organisational activities. This is understandable, because indicators such as pool of inspectors, quality resource allocation and measured process data require a deep understanding of the company's inspection process and this information is typically gathered from discussions with the company staff during evaluation session.

The result of a capability evaluation performed in an IT company is also presented in Figure 2. The diagram at the top of the matrix shows an estimate of each base practice in graphical form, and at the bottom of the matrix the estimate is represented in numeric form. The company has about 70 employees in Finland and our assessment revealed weak points in the inspection process at the whole company level.



	%	43	47	48	33	22	29	67	56	43	43	62	50
Indicator	**Score**	**P.1**	**P.2**	**P.3**	**P.4**	**P.5**	**P.6**	**P.7**	**P.8**	**P.9**	**P.10**	**P.11**	**P.12**
1) Entry decision	2							V 4		E 2			
2) Entry criteria	1	V 2						E 1					
3) Schedule	3								V 6	E 3			
4) Inspection resource allocations	1		V 2						V 2	E 1			
5) Request for rework on another a	1											V 2	
6) List of corrections	3						E 3					V 6	E 3
7) Acceptance of corrections	1												V 2
8) Request for recorrection	0												V 0
9) Exit decision	3												V 6
10) Exit criteria	1	V 2											E 1
11) Forms	2	V 4	EV 6		EV 6					E 2	E 2		
12) Tutorials	2	V 4		E 2							E 2		
13) Courses	1	V 2		V 2						E 1			
14) Collaborative tools	1		V 2		V 1				E 1	E 1			
15) Data gathering & anal. tools	0		V 0		EV 0	E 0	E 0			V 0	E 0		
16) Rules	2	V 4				EV 6			E 2	E 2	E 2	E 2	
17) Role descriptions	1	V 2		E 1		E 1			EV 3				
18) Checklists	0	V 0				EV 0			E 0	E 0			
19) Issue list	3						E 3			V 6	E 3	E 3	E 3
20) Categorisation	0					E 0	E 0				V 0	E 0	E 0
21) Pool of inspectors	2		V 4	V 4					E 2				
22) Selected inspection leader	3		V 6					E 3	E 3				
23) Request for training	1		V 2	E 1									
24) Quality resource allocations	1	V 2	V 2										
25) Reports	0	E 0	E 0	E 0	V 0	E 0	V 0						
26) Measured process data	0					E 0	E 0			V 0			
27) Improvement suggestions	1	E 1											
28) Process change requests	1	V 2	E 1			E 1							
29) Inspection process definition	3	V 6	E 3	E 3	E 3				E 3				
	sum	31	28	13	10	8	6	8	22	18	9	13	15
	max	72	60	27	30	36	21	12	39	42	21	21	30

Fig. 2. Capability determination matrix.

In the capability evaluation we start by estimating the grades of existence for each indicator. This estimated value is then multiplied by the weight attached to the indicator type (enabling = 1, verifying = 2) and placed in the matrix. The fact that verifying indicators carry twice the weight of enabling ones is justified on the grounds that they really guarantee that something has been done, whereas enabling indicators merely make certain activities possible. Finally the capability value is calculated for each base practice by means of sum and max values depicted at the bottom of the matrix. The values are also depicted in the form of a diagram. As a result of the

evaluation, a profile describing the capability of each activity is created, and put together these estimates give overall view of the process as a whole.

In this company, the staff is mostly young people in their first established post. The company has recently started to improve its quality assurance, however, and systematic inspecting has been introduced as a new practice. We have evaluated the company twice, the second occasion giving a more accurate result, because the interviewees had a more accurate picture of the inspection process, although both evaluations pointed to similar weak points. The core set of base practices is fairly well established, but supporting practices are almost entirely absent. The most important improvement suggestions should be focused first on core practices and after that on supporting ones, i.e. the company should start with adequate definition of the inspection process and appropriate training in this process. The definition of checklists and guides for defect categorisation would help to improve base practices P.5, P.9 and P.10 in particular, and the training would affect P.3. The company does not collect measurement data regularly nor does it refine such data into reports for the management, which means that they will have to define the metrics for inspection improvements and the tools and data required for that purpose at a later stage.

3.2 Improvement with Patterns

According to the experiments and evaluations that have been made with the i3 capability model, inadequacies in software inspection processes are often similar in distinct organizations. In addition, elements and features for the inspection improvement patterns have been extracted from related inspection literature and research.

Improvement patterns for software inspection process are introduced for straightforward implementation of improvement actions. Pattern approach provides a set of practical guidelines for actually executing the improvement actions, which are often missing in the full-scale assessment and improvement models.

After the evaluation, diverse approaches can be taken to actually achieve the process improvement. First, one can go through the indicator list once more, placing emphasis on the indicators with low scores. It is useful to have an expert to aid in this procedure, as the meaning and implications of an individual indicator may go further than the name of the indicator implies. For example, indicator “tools” have to be interpreted in a different way depending on the situation. Sometimes a comprehensive groupware package is needed, sometimes paper forms and templates are sufficient.

Even though the indicators are rated according to their importance for the organization, defining improvement steps requires in-depth understanding of the inspection process. Inspection improvement patterns are pre-defined lists of guides and procedures of the most typical action points to achieve process improvement. Each pattern defines an unambiguous goal for the inspection process upgrading. The purpose of the pattern catalog is to aid the assessor and the company representatives to focus the improvement activities on the most crucial parts of the process. Typically the weakest activities are addressed first.

When utilizing the patterns, an organization sets an overall goal for the process improvement effort. The goal can be based on the assessment report or common sense. If particular problems are uncovered during the assessment interviews, these

can direct the goal setting. The most straightforward method to determine suitable objectives is to read through the symptoms in the improvement pattern descriptions.

After determining the main goal, the most suitable patterns for the situation can be selected from the pattern catalog. Currently, there are seven patterns in the catalog. For each pattern, the following characteristics are defined:

Purpose of the pattern describes how the inspection process is enhanced, if the pattern is used.
Symptoms section presents a variety of problems and difficulties. If these have been encountered during inspections, the pattern may be suitable for the situation.
Possible reasons for the problems or inefficiency are listed.
Activities related to the pattern are listed.
Action list represents a strategy to solve the problems. The procedures are derived from activities and indicators in the i3 model.

Here is an example pattern, composed in the company, whose results were shown in previous section:

Purpose
The purpose of this pattern is to stabilize the inspection process in the organization.

Symptoms
- The inspection process has been introduced just recently.
- The motivation and purpose of inspections is unclear for participants.
- Not much information about the efficiency of the inspection process is available.

Possible reasons
The process description is not up-to-date.
- Checklists are not adequately made use of.
- Defects are not classified and analysed.
- There are no tools to manage the inspection information.

Related activities
- P.1. Establish and improve the inspection process.
- P.2. Organize inspection.
- P.3. Train participants.
- P.4. Support with computer tools.
- P.5. Maintain rules and checklists.
- P.6. Refine information.
- P.10. Categorize defects.

Action list
- Allocate adequate resources for process improvement.
- Make a schedule for the improvement.
- Evaluate the process.
- Obtain feedback and improvement ideas from the personnel.
- Consult an expert.

- Define defect classification categories.
- Create checklists for different roles.
- Update existing checklists.
- Define metrics to be gathered.
- Establish a procedure for recording metrics.
- Establish a procedure for analysing and reporting inspection data.
- Establish a procedure for recording feedback and improvement suggestions.
- Ensure that inspection process description is up-to-date in the quality handbook.
- Name an owner for the process.
- Define relations to other processes.
- Promote inspections.
- Write instructions for the inspection process.
- Arrange training.
- Evaluate the need for computer tools for the inspection process.
- Employ the tools.
- Arrange training for the users of the tools.

In addition to this company-specific pattern, we have seven generic patterns, which are listed in Table 2. The catalogue is a good foundation for inspection process improvement, although it is not complete. Experiences from future inspection capability assessments and improvement projects should be captured in patterns.

Table 2. Pattern catalogue.

Pattern name	**The main goal of the pattern**
Greed	Aims at finding more defects during inspections.
Early bird	Aims at finding defects at earlier stages of development.
Substance	Aims at finding more serious defects in inspections.
Comfort	Aims at making the inspection process easier to run.
Promotion	Aims at promoting the process so that it is carried out more often and in larger number of projects.
Wisdom	Aims at more understandable, transparent and effective inspection process.
Precision	Aims at making the process more rigorous, thus making it more effective.

According to our experiences, the pattern descriptions need to be accompanied with concrete examples of the execution of the specific improvement actions. Furthermore, it would be useful to summarize the possible consequences of specific actions, as using patterns may initiate changes in other processes as well. For example, inadequacies were found in the descriptions of other development processes during the composing of the inspection training material, and the relevant points were updated as well. As a result, the whole quality system of the company was able to benefit from the modifications.

After a half year of running, the improved process seems to perform well. There have been certain problems concerning data-gathering tools, and the importance of

continuous training and coaching was underestimated first. However, as a consequence of training, metrics program and other improvements, the effort needed to accomplish inspections has been decreased 10-20 percent and discovered defects are more serious than before.

The experiment caused a number of adjustments in the pattern descriptions. Action lists needed to be represented in more convenient order, overlapping parts of separate patterns were removed and pattern descriptions were clarified in general. However, we discovered that the pattern catalogue offers a feasible foundation for the inspection process improvement.

4 Conclusions

We have evaluated the usability and relevance of the capability model by means of five experiments. The model has been updated according to the research results. The main problem in inspection improvement is that the definition of improvement steps requires a deep understanding of the inspection process, in which a certain amount of support is called for. This support can be provided in the form of patterns, which give advices and are suited for novices in the area. Improvement patterns are pre-defined sets of actions that can be taken to upgrade the inspection process. Each pattern has a clear goal and a set of symptoms for detecting an appropriate pattern. When using these patterns, the organization has to determine a general goal for its process improvement. Goals can be based on the assessment report, common sense and feelings arising during the assessment.

A further research topic would be to develop a SPICE-compliant version of the model. The SPICE-based assessment allows evaluation and improvement to be focused on a process (e.g. review/inspection) and thus there is a need for a tailored model. Changing the model to be compliant with generic SPI models causes changes in the number of base practices and indicators to be walked through, as that version starts with a core set of base practices (six activities in the middle bar, Figure 1) and related indicators. The evaluation of adequate education and training and of tool support comes later, when evaluating the process resource attribute at the third (established) level of capability.

The presentation of a new model always requires some justification as to why this new model is necessary. We can sum up the benefits of the present model by evaluating its usefulness in terms of three questions: (1) does the model help to find the most important activities to improve? (2) Does the model help to find the best improvement suggestions? (3) Does the model work in practice?

We can answer positively to all these questions. The idea in i3 model is that, if we find indicators of a base practice, we then have some justifications for assuming the existence of that base practice. If some of them are at a low level or missing, they should be targets for improvement activities. Improvement patterns help in determining most suitable improvement actions. Furthermore, our experiments have discovered weak points in the companies' inspection processes which were also agreed on by the companies.

Finally, a warning is justified. Although we focus on the inspection process, we assume that a company which aims at improvement in this respect has already defined the whole software development process at an appropriate level. SME companies

should not focus all their process improvement measures on inspection, but rather they should improve the whole development process, with inspections as one important part of this.

References

1. Grunbacher, P.: A Software Assessment Process for Small Software Enterprises. In Proceedings of the 23rd EUROMICRO. Conference, Budapest, (1997) 123-128
2. Batista, J., Dias de Figueiredo, A.: SPI in a Very Small Team: a Case with CMM. Software Process - Improvement and Practise, Vol. 5 (2000) 243-250
3. Potter, N.S., Sakry, M.E.: Making Process Improvement Work. A Concise Action Guide for Software Managers and Practitioners. Addison Wesley, Boston (2002)
4. Rico, D.F.: Software Process Improvement (SPI): Modeling Return on Investment (ROI). http://davidfrico.com/dacs02pdf.htm (2002)
5. Conradi, R., Marjara A., Skåtevik, B.: Empirical Study of Inspection and Testing Data at Ericsson, Norway. Proceedings of PROFES'99, Oulu (1999) 263-284
6. Gilb, T., Graham, D.: Software Inspection. Addison-Wesley, Wokingham (1993)
7. O'Neill, D.: Issues in Software Inspection. IEEE Software, Vol 14 (1997) 18-19
8. Weller, E.F.: Lessons learned from Three Years of Inspection Data. IEEE Software, Vol 10 (1993) 38-45
9. Fagan, M.E.: Design and Code Inspections to Reduce Errors in Program Development. IBM Systems Journal, Vol 15. (1976) 182-211
10. Perpich, J.M., Perry, D.E., Porter, A.A., Votta, L.G., Wade, M.W.: Anywhere, anytime code inspections: Using the web to remove inspection bottlenecks in large-scale software development. Proceedings of the 19th International Conference on Software Engineering (1997) 14-21
11. Tervonen, I., Iisakka, J., Harjumaa, L.: A Tailored Capability Model for Inspection Process Improvement. Proceedings of the Second Asia-Pacific Conference on Quality Software, (2001) 275-282
12. Ward, R.P., Fayad, M.E., Laitinen, M.: Software Process Improvement in the Small. Communications of the ACM, Vol 44 (2001) 105-107
13. Kuvaja, P., Similä, J., Krzanik, L., Bicego, A., Koch, G., Saukkonen, S.: Software Process Assessment and Improvement: The BOOTSTRAP Approach. Blackwell Publishers, Oxford (1994)
14. Burnstein I., et al., A Testing Maturity Model for Software Test Process Assessment and Improvement, Software Quality Professional, vol 1 (1999)
15. Gelperin, D., Hayashi A.: How to support better software testing. Application Trends, May (1996) 42-48
16. Emam El K., Drouin J., Melo W.,: SPICE: The Theory and Practice of Software Process Improvement and Capability Determination. IEEE Computer Society (1998)
17. Grady R., van Slack T.: Key Lessons in Achieving Widespread Inspection Use. IEEE Software, Vol 11 (1994) 46-57
18. Zahran, S.: Software Process Improvement, Practical Guidelines for Business Success. Addison-Wesley, UK (1998)
19. Buchman, C.D., Bramble, L.K.: Three-tiered Software Process Assessment Hierarchy. Software Process – Improvement and Practice, Vol 1 (1995) 99-106
20. Horvat, R.V., Rozman, I., Gyorkos, J.: Managing the Complexity of SPI in Small Companies. Software Process – Improvement and Practice, Vol 5 (1995) 45-54
21. Sakamoto, K., Nakakoji, K., Yasunari, T., Niihara. N.: Toward Computational Support for Software Process Improvement Activities. Proceedings of the 20th International Conference on Software Engineering, Kyoto (1998) 22-31

22. Tyran, C.K., George, J.F.: Improving Software Inspections with Group Process Support,.Communications of the ACM, Vol 45. (2002) 87-92
23. O'Neill, D.: National Software Quality Experiment: Results 1992-1996. Proceedings of Quality Week Europe Conference, Brussels (1997) 1-25
24. Wiegers, K.E., Sturzenberger, D.C.: A Modular Software Process Mini-assessment Method. IEEE Software, Vol 17 (2000) 62-29
25. Richardson, I.: SPI Models: What Characteristics are Required for Small Software Development Companies? Software Quality Journal, Vol 10 (2002) 101-114
26. Kelly, D.P., Culleton, B.: Process Improvement for Small Organizations. IEEE Computer, Vol 32 (1999) 41-47
27. Herbsleb, J., Zubrow, D., Goldenson, D., Hayes, W., Paulk, M.: Software Quality and the Capability Maturity Model. Communications of the ACM, Vol 40 (1997) 25-29
28. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
29. Appleton, B.: Patterns for Conducting Process Improvement. PloP'97 Conference, http://www.cmcrossroads.com/bradapp/docs/i-spi/plop97.html (1997)
30. Rising, L.: Patterns: A way to reuse expertise. http://www.agcs.com/supportv2/techpapers/patterns/papers/expertise.htm (1998) (referenced 01.03.2003)
31. Sakamoto, K., Kishida, K., Nakakoji, K.: Cultural Adaptation of the CMM: A Case Study of a Software Engineering Process Group in a Japanese Manufacturing Company. In: Fugetta, A., Wolf, A. (eds.): Software Process. John Wiley & Sons Ltd, West Sussex (1996) 137-154

Comparing Global (Multi-site) SPI Program Activities to SPI Program Models

Atte Kinnula[1] and Marianne Kinnula[2]

[1] Nokia Corporation, P.O.Box 50, 90571 Oulu, Finland
atte.kinnula@nokia.com
[2] University of Oulu, Dept. of Information Processing Sciences,
P.O.Box 3000, 90014 University of Oulu, Finland
marianne.kinnula@oulu.fi

Abstract. Software Process Improvement (SPI) methods have been used for years as means to try to solve the problems in software development. Number of SPI life cycle models exists, and some of them take a wider look to the problems. However, little information exists about how the SPI models apply to global level SPI programs in a multi-site environment. This article takes a historical look into one such case and compares how well IDEAL and ISO 15504-7 models match the actual activities. The results are naturally only indicative, but suggest that the literature models do not readily scale up and that a separate model may be needed to support setting up and guiding the execution of a multi-site SPI program.

1 Introduction

In the past 10 to 15 years Software Process Improvement (SPI) efforts have become rather commonplace means to try to solve, at least partially, the problems in software development. These efforts range from small, local, and focused projects to larger and more sustained programs that tackle bigger goals, some of which may attain a truly global nature spanning over several sites. In many cases global SPI initiatives can be considered to be simply a heightened management focus for process improvement, without actual program structure. The SPI activities are indeed local and the main 'global" activity is setting a high-level goal (such as "attain maturity level 2") and monitoring of progress. Sometimes, however, in multi-site and global companies there may be a need to create a more cohesive SPI effort across the entire company.

How bad the need for coordinated effort is depends on great many things, but one observable factor, based on authors' experience, is the structure of the company's R&D organization. In cases where each site is self-sufficient R&D –wise, the need for a global SPI program may be less than in cases where the R&D is distributed and there is a strong inter-dependence between sites for product development. In latter cases an uneven progress in SPI is undesirable, as the product development becomes a chain where the weakest link (or site) has a significant impact on the net maturity (i.e. reliability, quality), effectively negating the gain from SPI on the sites where progress is faster. As improvement costs money, not only to do but to maintain as well [3], it is quite understandable that from management viewpoint then, the SPI efforts on all the

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 76–91, 2004.

other sites are effectively a waste of money. A more harmonized approach is desirable, and a proper SPI program, at the global level, is a natural attempt to tackle this need.

The literature offers a number of SPI life cycle models, most of which are well suited for local SPI efforts (e.g. Plan-Do-Check-Act [21], Effective change process [10], AMI [20], Pr^{2}imer [13], Iteration cycle [4], CPI-7 [15]), and a couple that try to provide insight for larger, sustained long-term programs that encompass multiple SPI activities, mainly IDEAL [17], ISO 15504-7 [12], and Quality Improvement Paradigm [2], [3]. In addition, industry experience reports are available (e.g. [1], [11], [6], [7], [8], [9], [5], [16]) to describe how these models have been applied. However, these reports also have a focus at local, rather than at the global activities.

Reports describing the SPI programs that span multiple sites are not readily available, nor do the models discuss how, if in any manner, the life cycle models can be applied for global level SPI programs. At best they offer only rudimentary responsibilities such as 'providing resources', 'monitoring progress', 'directing the program' [17], or 'fostering communication and teamwork', 'providing reward and recognition' [12] etc. For the operational manager, charged with the task of setting up and maintaining the global program, these are clearly not sufficient – more tangible guidelines are needed; what is the organization like, what kind of processes run at global level, how they are related to local level activities, how to balance the global need for uniformity vs. local desire for diversity, and so forth. A natural thought is that the SPI cycle models could be used as such – a global SPI program is, after all, an SPI program (although different in scope and scale). But do the models easily scale up, or is the global cycle somehow different?

This article studies one global SPI program that was clearly cyclic in nature and compares its activities against the IDEAL [17] and ISO 15504-7 [12] SPI cycle models, both of which are intended for large and more sustained programs. A third candidate with elements to support a larger program, the QIP [2], [3] was left out for two reasons – the lack of space in this article and due to lesser familiarity to the authors. The purpose is to see if the global level program followed these cycle models or if it was very different from what the models propose.

Since the viewpoint in this study is a global level SPI program, the comparison requires some definition of what actually is a *global level* SPI program and how it differs from a *local level* SPI program. The following definition has been used here:

> A global level SPI program is a program that is working to improve SW process of an organization in a situation where the organization consists of a number of independent and geographically distributed SW development units.

A local level program would then be an SPI program that is working to improve SW process of one (single) SW development unit.

A global SPI program is also considered to include all the SPI activities that take place in an organization. The activities are considered from global viewpoint, i.e. how the task or results reflect across the entire company and how well the activity covers the units within the scope of the program. For this reason e.g. a single process assessment activity in a single site does not constitute a process baseline activity for a global SPI program, any more than an in-depth process analysis of a single process

done by an SPI action constitutes a comprehensive assessment activity for a local SPI program.

The rest of this paper contains the case description including a detailed look at the cycles from global perspective (chapter 2), a comparison of global activities to IDEAL and ISO 15504-7 cycle phases (chapter 3), a comparison of global activities to IDEAL and ISO 15504-7 continuous activities (chapter 4), discussion (chapter 5) and conclusions (chapter 6).

2 Case Description

Nokia Corporation is the world number one manufacturer of mobile terminals, owning close to 40% of the world mobile terminal markets. It is structured into business groups, one of which is Nokia Mobile Phones (NMP). This particular business group is further divided into R&D centers (sites) around the world, responsible for product development, all of which had SW activities ongoing. The product development is done in a distributed fashion, i.e. software from one site is delivered to other sites and to multiple products. In this article we will look at the SPI effort at the NMP level, which can clearly be considered to be a global entity. The material of the case has been gathered over the years from the start of the NMP SPI Program, to its closure. Furthermore, one of the authors of this article was the manager of the said NMP SPI Program for two years, and kept following the progress of the program out of personal interest afterwards. The same author also studied the NMP SPI Program from infrastructure point of view. Those interested can read more of this study and of the case itself from [14].

2.1 The First SPI Cycle

A corporate-level SPI initiative was launched at Nokia in early 1996, with an aim to get more focus on SPI and drive it into sustained activity across the corporation. The business groups of the Nokia Corporation were expected to establish their respective SPI efforts to meet the corporate level targets. [18] Although NMP had had SPI activities and goals already prior to this, they were not in par with the corporate expectations and NMP had to re-evaluate its SPI approach thoroughly.

The operative SPI manager for NMP was committed to the corporate level goals and obtained also support to a new SPI initiative from the top management of NMP. A task force was set off to evaluate if the goals could be achieved with the current SPI set-up or should something be done. The task force studied the current problems and lessons learned from past SPI efforts and, as a result, recommended a new global SPI program for NMP, to facilitate the achievement of the goals. It was also evident that the entire SPI infrastructure in NMP had to be revised in order to meet the corporate level demands. The new NMP SPI Program was initiated in late 1996.

The first task of the new NMP SPI Program was to define a set of global and local level objectives. These were based on both demands from corporate level and needs of NMP. However, it should be noted that the objectives were defined without actual measurement data; as such data covering the entire NMP was not available. Furthermore the global objectives only indicated areas for improvement, based on the

Capability Maturity Model for Software (CMM) [19], and did not define any particular processes or action plans, as the improvement needs varied from site to site. A staged approach to reach the corporate level goals was adopted. In accordance with the task force findings, the first set of objectives were mostly about revising the SPI infrastructure, rather than focusing on the actual software process improvements, and as such fell short of the corporate level initiative targets. This was, however, deemed to be necessary in order to sustain the SPI activity on a long run. In addition, some of the objectives and requirements dealt with the actions to collect and report status data, including process maturity reports from self-assessments, which could be used to evaluate the problems and maturity of the SW Process both at the local and at the global level. A calendar-based checkpoint for closing the first cycle and revising the objectives was agreed. The assumption was that for the second cycle the set of objectives would be in par with the Nokia-level SPI goals.

After the objectives for the first cycle were defined, commitment to them was obtained from the local level. While most of the sites were committed, some sites were unable to resource the activity at this stage and it was agreed that they would work on the resource issue and join the program later on. Those sites that did join the program were given the responsibility to create a local level SPI infrastructure, and form local SPI actions that would work towards the objectives. The NMP SPI Program established the global level infrastructure and defined interfaces and rules for running the entire program. The role of the global level was defined to be that of monitoring, coordination, fostering communication, improving SPI competences, acting as a mediator between external SPI consulting sources and local SPI activities, and providing support for the local level.

It was also agreed that the local level had the responsibility and power to define their specific goals according to local situation – thus the global part of the NMP SPI Program had only loose control over the local SPI programs. While the sites were committed to the global objectives, they also had local needs and these two sometimes had a priority conflict. Such conflicts were handled on a case-by-case basis, and in some cases the local level SPI program delayed its progress towards global objectives in order to answer to the local needs first.

After having committed to the globally agreed objectives, the local level sites initiated their own SPI strategy and action planning phases and continued from there to do measurement and improvement activities, supported by the global SPI entity. They were also required to report their status on a regular basis. In most sites the local level was able to establish an entire SPI program to run the local SPI activities, but some sites used the global level objectives in a quite straightforward fashion and simply executed those as separate SPI projects, without taking first the assessment steps and without an overlying local SPI program.

Throughout the first cycle the activities to strengthen and improve the infrastructure were ongoing in parallel with other SPI activities both at the global and local level.

2.2 At the Checkpoint

At the agreed checkpoint time NMP SPI Program performed a status review, where achievements were evaluated against the objectives and requirements. This was carried out in two areas – first the status of the SPI infrastructure (and hence the SPI

program itself) across NMP was evaluated and lessons from trying out the new approach were collected. The activity was about gathering and communicating the lessons learned in order to see if the program was going into a right direction, and how it could be further improved. The second review was that of evaluating the status of the SW Process in NMP from the status and assessment reports delivered from the sites. These reviews allowed the global level to monitor the use of the new processes and formed the global level baseline findings and recommendations report, which was then communicated across the organization, and to the Nokia level initiative.

After the evaluation, the goals for the next cycle were defined based on the review and evaluation of the objectives, and commitment from the local level to the goals was obtained. As some of the SW process related objectives for the first cycle had not been achieved across NMP, it was realized that the two-step approach was not realistic, and the goals for the second cycle had to be re-thought. With the insight obtained from the global baseline, the next set of goals was based on more realistic understanding of the current status and SPI capability. Nevertheless the progress was good enough to make the new objectives focused on improving the actual SW process, rather than establishing the improvement infrastructure. The measurement objectives for the second cycle were also reviewed but no changes were made.

2.3 The Second SPI Cycle

The second cycle was very similar to the first, with three major differences. First, some of the local SPI projects were not finished during the first cycle and were carried over to the next, so the local and global cycles were not synchronized. Second, the NMP SPI Program initiated certain global SPI projects, managed directly by the global organization. These were based on the needs apparent from the local reports and observations that had been made during the first cycle. These projects were given a thorough planning at the global level, very much like a local level SPI project would be planned. The third difference was the inclusion of several R&D sites that had not been able to resource the SPI activities during the first cycle but were now ready to join the program. For these sites the set of objectives for the second cycle was considered to be too demanding, and a reduced set of objectives, achievable within a year, was defined. In addition some of the sites that had been active from the beginning had had problems with establishing the local infrastructure, and were also given a 'reduced' set of goals. Thus it was acknowledged that sites advance at different pace and in the progress is not "synchronized". This had to be taken into account in planning and managing the global objectives and the NMP SPI Program itself.

At the end of the second cycle the NMP SPI Program went again through the status check and established the objectives for the third cycle. In addition the lessons learned were used to improve the SPI program infrastructure.

2.4 The Third SPI Cycle

The third cycle was initiated but never quite completed, as the changes in NMP organization made it too difficult to maintain a NMP-wide SPI program. Furthermore

the Nokia-level initiative was closed in 2001, as it was concluded that SPI had become business as usual, and the corporate level task force was no longer needed. With the closure of Nokia level initiative also the NMP SPI Program was closed and responsibility for SPI was transferred to the new business units that had been formed between the global level and the site level.

2.5 The Global Cycle

Based on the case data, it is possible to draw a rough SPI process cycle from the global perspective.

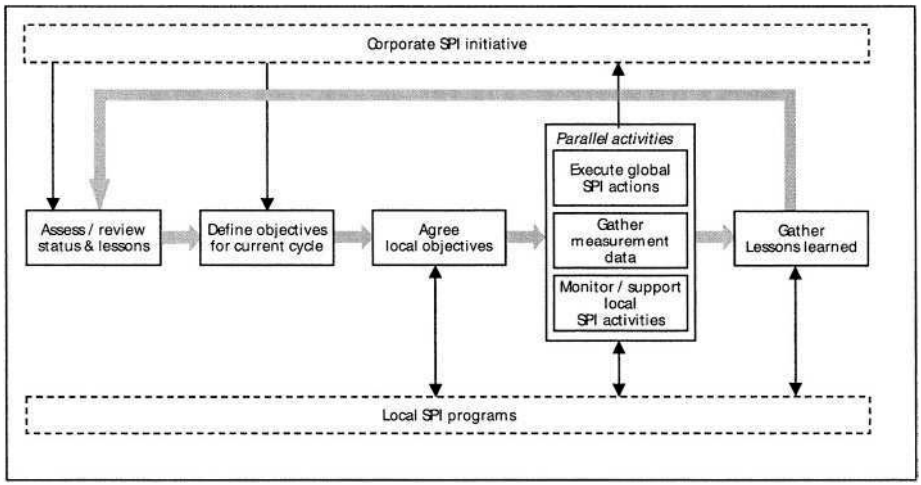


Fig. 1. A global SPI process cycle

The first phase is where the accumulated knowledge of the process maturity and lessons are evaluated, to establish an understanding where the company as such seems to be in respect to overall goals. There was no 'initiation phase' in the case studied, as NMP already had an SPI program in place. The program was thoroughly revised, but this does not constitute initiating a completely new program. In the second phase the accumulated knowledge is turned into concrete plans, influenced (during the first pass) by the corporate initiative. The third phase is where the different sites are consulted and a set of objectives is agreed for the current cycle and incorporated to the local SPI programs. It is important to note that although the picture does not point it out, there were eventually three sets of goals – one for those sites who were not yet part of the program as such (no SPI resources), one for those who had just started, or had not had decent progress during the previous cycle, and one for those who had had good progress. The fourth phase is the 'main' work during the cycle, mainly with the local level but also at global level and included status reports to the corporate initiative. The last phase was to get ideas for improving the SPI processes and was done as a workshop at the end of the cycle, together with local and global representatives.

3 Comparison to Cycle Phases

Both IDEAL and ISO 15504-7 cycle models take an approach that an SPI program, once initiated, works towards its improvement goals. When the goals have been achieved the 'old' SPI program is concluded and a 'new' program is initiated, with new objectives. Both models contain two different types of activities: the actual cycle activities and the continuous activities. In this chapter the NMP SPI program activities are compared to the cycle activities of both models.

3.1 IDEAL Cycle vs. NMP SPI Program

The IDEAL model states up front that the guide is "primarily focused on organization-specific activities" [17, p.8]. Organization in this model is equivalent to a single unit, as opposed to corporation, which the IDEAL model defines to be "made of a number of separate organizations" [17, p.8]. Looking at the Nokia case, at the business group level the 'organization' would be a single software development unit (R&D site) and the business group (e.g. NMP) is equivalent to IDEAL model's 'corporation'.

The IDEAL phases can be summarized as [17, p.6]:

I (Initiating) – Learn about process improvement, commit initial resources, and build process infrastructure. This is carried out only once, and the subsequent SPI programs go from Leveraging phase to Diagnosing phase, providing that the continuous cycle is kept up.
D (Diagnosing) – Establish current levels of process maturity, process descriptions, metrics, etc. Initiate action plan development.
E (Establishing) – Establish goals and priorities, complete action plan
A (Acting) – Research and develop solutions to process problems. Expand successful process improvements to entire organization
L (Leveraging) – Prepare for the next cycle through the IDEAL model. Apply the lessons learned to refine the SPI process

From how IDEAL treats goal setting in the Leveraging phase it seems that the model considers an SPI program to be equivalent to one IDEAL cycle.

Since NMP already had an existing SPI program, it is justified to leave the 'Initiate' phase out of this comparison and concentrate on the remaining four steps of the model.

Comparing the early (1996) activities of the NMP global SPI program to the IDEAL model shows that they correspond closely to the *Leveraging* phase: a new set of SPI goals were established, commitment obtained, the need for new infrastructure identified through studying the current problems and lessons learned from past SPI efforts, and a new program established to meet the goals.

The next activities after initiating the NMP SPI Program bear resemblance to both *Establishing* and *Diagnosing* phases. The setting of objectives and requirements for the local level, and getting local level commitment belong to the former, while some of the requirements dealing with collecting and reporting status data belong to latter. More specifically they cover the first two steps of the Diagnosing phase. However,

the order of the phases is reversed at the global level, and the baselining activity is missing altogether at this point. In the IDEAL model the current state analysis from the Diagnosing phase is expected to be available for defining the objectives but this was not what happened in the case.

The goals were established based on Nokia corporate level targets and data from the 'previous cycle' (i.e. past SPI efforts), as it would be unfeasible to do a comprehensive assessment step in a short period of time across a large organization in order to establish a baseline in the sense IDEAL defines it. Instead a global level SPI program has to resort to establishing the first cycle objectives based on very little data and the objectives for the subsequent cycles are then based on the data gathered during the previous cycle. A major difference between the NMP SPI Program and the IDEAL cycle in establishing the objectives was that the NMP SPI Program goals were at much higher abstraction level and control over local level activities much weaker than what the IDEAL appears to expect. This, however, was a necessity as NMP was already at that time a very large organization with very different sites, and a culture that fosters diversity. It was considered most reasonable to have local level apply their first-hand knowledge of what is needed and what works, and give a broad direction for improvement, rather than imposing a rigid set of targets from the global level.

After the local level SPI strategy and action planning was initiated, the global level SPI program can be considered to have entered the *Acting* phase. The local level also performed measurement and assessment activities. This – from the global SPI program viewpoint – is *Diagnosing* phase step 3, i.e. conducting the baselines, although this activity was, from global perspective, made in a piecemeal fashion, not concurrently across all sites.

Activities to strengthen and improve the infrastructure were also ongoing in parallel with all other activities throughout the cycle. This again resembles the 'Revise Organizational Approach' –step from the *Leveraging* phase.

At the agreed checkpoint the status reviews were made, goals for the next cycle were established and measurement objectives re-evaluated. Here again we see multiple IDEAL cycle phases represented. The SPI infrastructure review corresponds to the *Leveraging* phase, with the exception that it was only concerned with gathering and communicating the lessons learned, not initiating a new SPI program. The SW Process status review in turn corresponds to the *Diagnosing* phase steps 5 and 6, i.e. developing the *global level* findings and recommendations report and communicating them to the organization. The goal setting corresponds to the *Establishing* phase and measurement objective review covers the *Diagnosing* phase steps one and two.

During the second and third cycles the same pattern emerged, although the need for having several goal sets in parallel for sites in different phases of progress is a major difference to the IDEAL cycle, which clearly expects to have one set of goals for each cycle. The existence of global level SPI projects is somewhat parallel for having local level SPI actions ongoing, the difference being that the NMP SPI Program had direct control over them as opposed to very loose control it had over local SPI activities. In addition the local level having "old" SPI activities from the first cycle is again somewhat different from what the IDEAL expects, as the assumption is more towards that of the cycle being not finished until all SPI objectives are met.

3.2 ISO 15504-7 Cycle vs. NMP SPI Program

The ISO 15504-7 model for SW process improvement cycle states up front that a program may have long-ranging improvement goals that are achieved by going through several iterations of the improvement cycle. The model has a total of 8 steps. These are:

1. *Examine organization's needs* to identify drivers for, objectives of, and goals for SPI
2. *Initiate Process Improvement* as a programme of its own right, including planning and resourcing the program in an appropriate manner
3. *Prepare for and conduct a process assessment* in order to identify process strengths and weaknesses of the organization
4. *Analyze assessment output and derive action plan,* including prioritizing improvement areas, defining specific improvement goals and setting targets
5. *Implement the improvements* in order to improve the SW process
6. *Confirm the improvements* against agreed targets
7. *Sustain improvement gains* by institutionalizing the improved process
8. *Monitor the performance* of both the SW process and the SPI program

The ISO 15504-7 model's corresponding lead-in phase (step 1) has less emphasis than IDEAL model's first step on building something from a scratch. As such it fits better into a situation where the objectives of an existing SPI program are revised to the extent that it can be considered to be a new improvement program. This is also consistent with the fact that ISO 15504-7 considers it possible for an improvement program to go through several cycles, and the reviewing and revising of the goals and targets in step 8 is presented with being more 'adjusting' than 'renewing' in nature. Conversely the Leveraging phase in the IDEAL model is more 'renewing' in nature.
For this reason the activities of the NMP SPI program in the beginning of the first cycle can be considered to match the steps 1 and 2 of the ISO 15504-7 cycle. The Nokia level initiative made NMP to revise its goals thoroughly and a set of new goals were established *(step 1).* To facilitate achievement of these goals the NMP SPI Program was initiated (*step 2*). It started to plan the improvement program, form the new approach and revise the SPI infrastructure.

During this time local level requirements were also defined, including requirements to collect and report status data, e.g. process maturity reports from self-assessments, which roughly corresponds to the purpose of ISO 15504-7 *step 4.* The global level requirements were, however, defined before any of the data was available. The global level did not establish specific improvement goals, targets or an action plan as required by the model. Here the NMP SPI Program activities clearly conflict with ISO 15504-7 cycle model, which expects the organization to first conduct an assessment *(step 3)* and then go through an analysis and prioritization phase. The difference to the model is similar to that of the IDEAL model as discussed in the previous chapter, for the same reasons. Substitutes for the global assessment are the measurement and assessment activities at the site level and these, from global point of view, are continuously ongoing across the organization. This is explicitly included in ISO 15504-7 step 8, which means that the step 3, from global SPI program viewpoint, does not exist. Furthermore there is little sense in establishing a detailed action plan at the global level, as expected by the ISO 15504-7 step 4, since

different sites may have different problems. Instead, identifying a broad scope or area for the improvements and allowing the local SPI programs to draw the action plans is a more practical approach.

After the global objectives were established, local level went on planning their respective SPI strategies, action plans and change projects. From global viewpoint this corresponds to the *step 5* of the ISO 15504-7 model.

At the checkpoint the infrastructure and software process status were reviewed, and a new set of objectives for the second cycle was established based on both the review and the evaluation of the achievements. These activities bear resemblance to *step 6, step 8,* and also to some extent *step 4*.

Step 7 is not easily found from the activities of the first cycle of the NMP SPI Program. This step includes deployment of the improvements to all applicable entities, but this did not take place in a globally enforced fashion in NMP. While the general objectives were similar to all sites, the NMP SPI program did not attempt or have the responsibility to "roll out" a change that was developed and piloted in one site to other sites. Based on the previous experiences, such inter-site deployment was considered to be impractical for NMP at the time. As NMP is a very heterogeneous environment, the practices for carrying out a specific activity are not easily transferable to another site. Instead the NMP software process architecture had been structured to make a difference between "What" and "How" levels of process execution, where the former is to ensure process outcome and interface compatibility, and the latter is left for the local level to develop or borrow from other sites. Another activity in step 7 is monitoring the use of the new process, which is something that the global SPI program did do, through the standard measurement and assessment reports. As stated earlier, the pattern was very similar during the second and third cycles. Having several sets of objectives, because of sites advancing at a different pace, and having some SPI activities carried over from the previous cycle does not appear to be an issue from ISO 15504-7 model viewpoint. The model acknowledges that several iterations of the cycle may be needed to achieve the final goals.

3.3 Cycle Comparison Summary

Although the analysis confirms that the global SPI program indeed has similar activities as the two cycle models describe, the sequence and distribution of these activities in case cycle differs from that of the model cycles. In addition some of the activities identified in the models are missing or are not carried out as expected. As a result the projection feels somewhat artificial and it is rather safe to state that the global SPI program presented in this article does not follow the SPI program cycle as defined in either of the models.

4 Comparison to Continuous Activities

Both IDEAL and ISO 15504-7 identify certain activities which are beyond the cycle itself, mostly related to managing of an SPI program, and as such can be considered to be more or less continuous or done on-a-need -basis. In this chapter the continuous activities of both cycle models are compared to the NMP SPI Program activities.

4.1 Continuous Activities from IDEAL vs. NMP SPI Program

The IDEAL model lists six tasks under "Manage SPI program". The overall objective of this 'phase' is to provide oversight to the improvement projects and resolve issues. The tasks are:

- Setting the Stage for Software Process Improvement
- Organizing the SPI program
- Planning the SPI program
- Staffing the SPI program
- Monitoring the SPI program
- Directing the SPI program

In addition the IDEAL model recommends that a program that performs activities in a corporate context has the following responsibilities:

- Establishing infrastructure and links to support and coordinate the organization programs
- Looking outside the corporation for "process reuse"
- Supporting organizational activities through resources, common practices, communication, etc.
- Spreading findings, practices, information, etc., across a wider base within the corporation
- Acting as a focal point for outside process improvement influences, such as those from the Software Engineering Institute (SEI), etc.

There are two ways to view these activities. If the NMP SPI Program is considered to be the program itself, a steering group for the program should then for the most part carry out the continuous activities. In the case studied here the steering group did carry out some of the directing activities, and the Nokia -level initiative, through reports it required from the NMP, carried out the monitoring activities. However, the NMP SPI Program itself was largely responsible for the rest.

Another approach to the continuous activities is to project the NMP SPI Program activities to them, and indeed it would appear to be more fruitful approach. The early events and actions in the beginning of the first cycle are clearly *setting the stage for SPI* – the new requirements from the corporate level, the analysis of problems in the process and infrastructure, etc, all can be considered to be part of this phase. The first activity of the new NMP SPI Program was to *organize* itself, and define the infrastructure (of which organization is a part) for the entire SPI initiative, and this was indeed a continuous activity throughout the cycle. The *staffing* was another important issue, although it was considered to be part of the work on infrastructure (organizing), rather than a separate activity. The activities across the NMP were *monitored* through the reports, and the end-of-cycle checkpoint was used to review the status, across NMP, get lessons learned, and establish a new set of objectives – i.e. to *direct* the overall program that encompassed all SPI activities in NMP, both at global and local level.

As for the recommended activities, improving the infrastructure was indeed a central continuous effort, and the global part of the NMP SPI Program also had a notion that they act as an interface between the external sources for SPI issues, in order to get the latest ideas but also to verify their usefulness and applicability for NMP purposes, which corresponds to the second and last bullet. They also had an explicit responsibility to foster communication and support the local SPI activities, as identified in the third and fourth bullet, although the support role was more that of a consultant than resource pool and the process improvements were collected but not effectively disseminated, as discussed in the earlier chapters.

Evidently the only activity missing is the identifying, planning, and executing the actual SPI actions, which were carried out both at the local and global level. Of course this is the essential part of any SPI program, so the omission is not to be overlooked. The reason is, naturally, that these are covered in the cycle activities of the IDEAL model but in the case study the SPI actions did not necessarily have all the elements of a complete IDEAL cycle – this is most evident in the case of global SPI actions. Hence the assumption that the NMP SPI program would be just the complete set of IDEAL cycle phases and continuous activities does not quite match the case.

4.2 ISO 15504-7 Continuous Activities vs. NMP SPI Program

Like IDEAL model, the ISO 15504-7 considers management of the SPI program a separate task that covers organizing, planning and measuring SPI and reviewing SPI activities, although IDEAL appears to have more extensive list of the management activities. As with IDEAL, the ISO 15504-7 activities to *Manage the Process Improvement* are easy to map to the activities of the NMP SPI Program.

The early activities of the NMP SPI Program were much concerned about the establishing an infrastructure, corresponding to the ISO 15504-7 activity *Organizing for Process Improvement.* As already discussed this was also a continuous action that the global part of the NMP SPI Program was carrying out throughout the cycle, and expected the local level to do the same. After the initial work on infrastructure, the global objectives and the stepwise approach for getting to the corporate level goals were planned, which matches the *Planning for Process Improvement* -activity. *Measuring Process Improvement* was carried out in a 'continuous' fashion, although most of the metrics and assessment reports were filed in only twice a year. Finally the *Reviewing of Process Improvement Activities* was done at the end of the cycle.

Again, the missing parts are the SPI activities and the discussion in the previous chapter (4.1) about this gap applies to the ISO 15504-7 model as well.

4.3 Continuous Activity Comparison Summary

The continuous activities identified in the models fit fairly well with the global SPI program activities, although they omit the SPI actions themselves, for a good reason. However, assuming that the NMP SPI Program was just a complete set of cycle activities and management activities does not quite match the case. The global level SPI activities did not constitute an entire SPI cycle, and in addition some local SPI actions were also merely a group of separate SPI projects rather than a full-fledged

SPI program. Furthermore nothing in the both models' continuous activities suggests a cyclic process, and the case study program was clearly progressing through a cycle.

5 Discussion

From the case study it is apparent that while the global SPI Program activities were progressing through a cycle, with long-term objectives and a step-wise approach to them, there is only a partial match to the cycles presented in the IDEAL or ISO 15504-7. This suggests that the current models cannot be straightforwardly scaled up as such and a different cycle model may be needed.

The most notable difference to the cycle model activities of both IDEAL and ISO 15504-7 is how the baseline data was taken in the case study program. Although as a single deviation this does not appear to be much, it points out a deeper philosophical difference in improvement planning between the global cycle and the literature models. Both models expect the SPI program to perform a thorough baselining phase in order to find out improvement opportunities, which are then used to plan and initiate SPI actions for the current cycle. For a multi-site SPI program conducting such a baseline phase is neither feasible nor desirable for several reasons:

1. Executing a detailed diagnosis simultaneously in all sites is not feasible. This would be a huge task (to coordinate alone) and disruptive for the company. If performed within a tight time frame, there will be some local units to which the timing is poor, e.g. occurs during a critical project phase. This can potentially have a major negative impact on business and certainly has a negative effect on SW practitioners' attitude towards SW Process Engineering.
2. The company (if it has enough sites) would simply run into a problem of getting enough training capacity and qualified lead assessors.
3. If the baselines are carried out sequentially (over a period of time), it is likely that when the final baselines have been conducted, the findings from the first ones are no longer relevant since actions have been already done to change the situation.
4. In a multi-site organization the sites are likely to be very heterogeneous, which means that they may also have different problems. Making a diagnosis and, based on that, establishing same SPI actions across all sites may not be possible.
5. In a multi-site organization, micromanaging all the SPI actions is not feasible for the global SPI program. A more sensible approach is to establish local level SPI programs that establish and manage the local SPI actions. For that reason there is less need for the global level SPI program to have a detailed understanding of the exact strengths and weaknesses of each site.

A global level SPI program uses the best current understanding of the overall status in the company (from the measurement data available, rather than through a specific global baselining phase or action) as a source of the global level objectives. The data is collected in parallel with SPI activities and used to support planning of the next improvement cycle, rather than to plan activities of the current cycle – a clear difference to the way assessment data is treated in both models. While data collection can be considered to be a form of baselining, it is spread out in time as opposed to being conducted in a way that provides a snapshot status of the organization's SW

process. In a global SPI program the baselining phase is thus either missing or is split up into 'Collecting data' done in parallel with the SPI activities, and 'Analyzing data' done prior to proposing objectives for the new global SPI cycle.

While the cycle itself does not appear to scale up effortlessly, the global SPI Program activities can quite easily be mapped to "continuous activities" identified in both models. However, the literature models consider these activities to be non-cyclic (continuous) or ad hoc in nature, and for that reason place them outside the cycle itself. Yet, as stated, the NMP SPI Program was clearly progressing through a cycle.

We argue that the 'continuous activities' are, in fact, projections of a global level cycle, viewed from the local level. Although the models identify them as "continuous", this does not exclude the existence of a higher-level cycle. It should be kept in mind that the models are primarily intended for a local SPI program [17], and hence reflect the local view to the matter.

The final interesting finding is the issue of control over SPI activities. While the models assume that the SPI program has direct and tight control over all SPI activities being executed – a reasonable assumption for a local level initiative – the global SPI program appears to have more complex control structure. The program goals in the case study indicated the areas for improvement and thus a direction that the SPI program in its entirety should be progressing. However, the individual SPI activities (local level actions) were semi-autonomous within the global framework and had even the power to break away from it if this was seen justifiable. This would correspond a situation where an individual SPI action would inform the SPI program that it is focusing on different things than where the SPI program initially focused it on, because it finds those other issues more important. This is an important difference between a global and a more 'traditional' SPI program. The global SPI program has only moderate control over the local SPI programs and while the local SPI actions have limited initiative of their own, they still can act on their own volition. The reason for this is again in the heterogeneous nature of a global organization. Apart from the global needs the local SPI program is also responsible for answering the local needs and there are cases where the two have a priority conflict. In the case organization it was considered to be a "lesser evil" to have the local SPI organizations have the right to focus on local needs at the expense of global needs. The general idea of the SPI initiative was to make it business as usual. Centrally enforced actions had a risk of causing resentment and stifling local enthusiasm for SPI. Although this may be due to the company culture, the finding can also imply that the nature of management activities of the global SPI program may be more guiding than controlling.

In our opinion the evidence from the case study points out that while the current models have elements that are similar to what occurs in the global level, the models can not be used to establish and guide a global level SPI program. However, this is not to say that the current models are flawed in any manner. Instead it is apparent that the models are tailored and suitable for running local SPI programs and already have elements – the continuous activities - that work to bridge the local level SPI cycle and global level activities. Rather than trying to modify the existing models to cater also global needs a more sensible approach is to devise a different model for the global SPI program and establish more precisely defined interfaces between the two levels. The rough model presented on section 2.5 is a step towards a global SPI program cycle model, but clearly falls short in great many aspects and must not be mistaken for a proper model. More studies of such programs and activities are needed before the key steps and their actual nature emerge as a pattern.

6 Conclusion

In this article we have studied one cyclic global SPI program and compared its activities to the two most well-known SPI program life cycle models – IDEAL and ISO 15504-7 – to see if the global SPI program followed one cycle model or the other.

The findings suggest that the current models, as far as the cycle itself goes, do not scale up in a straightforward fashion, as they have been tuned for local (single-site) SPI programs. While all activities can be found if both cycle and continuous activities are taken into account, the phase structure does not match the global level cycle, nor is there a provision for multiple hierarchical levels (which are an obvious part of a global program).

However, we feel that there is no use to change the existing models as such, since they seem to apply well for the purpose they've been defined for. Instead the models should be expanded to contain a hierarchical structure, a global level cycle model, and the interaction between the global and local cycles. Although in this paper a very rough global cycle has been presented, it is clearly an oversimplified one and more case studies are needed to define a proper one. In addition the QIP model should be evaluated against industry cases to see if it has a better match to the processes of a global SPI program.

References

1. Basili, V., Green, S.: Software Process Evolution at SEL. IEEE Software, July (1994) 58-66.
2. Basili, V., McGarry, F.: The Experience Factory: How to Build and Run One. Tutorial TF01, 20th International Conference on Software Engineering, Kyoto, Japan (1998)
3. Basili, V., Weiss, D.: A Methodology for Collecting Valid Software Engineering Data. IEEE Transactions on Software Engineering, November (1984) 728-738
4. Culver-Lozo, K.: Software Process Iteration on Large Projects: Challenges, Strategies and Experiences. Software Process – Improvement and Practice 1 (1995) 35-45
5. Diaz, M., Sligo, J.: How Software Process Improvement Helped Motorola. IEEE Software, September/October (1997) 75-81
6. Dion, R.: Elements of a Process-Improvement Program. IEEE Software, July (1992) 83-85
7. Dion, R.: Process Improvement and the Corporate Balance Sheet. IEEE Software, July (1993) 28-35
8. Haley, T.: Software Process Improvement at Raytheon. IEEE Software, November (1996) 33-41
9. Hollenbach, C., Young, R., Plufgard, A., Smith, D. (1997) Combining Quality and Software Improvement. Communications of the ACM 6, Vol. 40 (1997) 41-45
10. Humphrey, W.: Managing the Software Process. Addison-Wesley, Reading, Massachusetts (1990)
11. Humphrey W., Snyder, T., Willis R.: Software Process Improvement at Hughes Aircraft. IEEE Software, July (1991) 11-23
12. ISO/IEC: 15504-7 Information technology – Software process assessment – Part 7: Guide for use in process improvement. ISO/IEC TR15504-7: 1998(E) (1998)

13. Karjalainen, J., Mäkäräinen, M., Komi-Sirviö, S., Seppänen, V.: Practical process improvement for embedded real-time software. Quality Engineering 4, Vol. 8 (1996) 565-573
14. Kinnula, A.: Software Process Engineering in a Multi-Site Environment – An architectural design of a software process engineering system. PhD thesis, Univ. Oulu, Dept. Information Processing Sciences. Acta Universitatis Ouluensis A333. Oulu Unmiversity Press, Oulu (1999) URL: http://herkules.oulu.fi/isbn9514253035/
15. Kinnula, A.: Software process engineering systems: models and industry cases. Acta Universitatis Ouluensis A372. Oulu University Press, Oulu (2001) URL: http://hercules.oulu.fi/isbn9514265084/
16. Laporte, C., Papicco, N.: Software and Systems Engineering Process Development and Integration at Oerlikon Aerospace. Software Process Newsletter 11 (1998) 10-17
17. McFeeley, R.: IDEAL℠ – A User's Guide to Software Process Improvement. CMU/SEI-96-HB-001, Software Engineering Institute, February (1996)
18. NSPI Launch Letter by Matti Alahuhta, and corresponding slide sets (Nokia internal documents) (1996)
19. Paulk, M., Weber, C., Garcia, S., Chrissis, M., Bush, M.: Capability Maturity Model for Software, version 1.1. SEI-93-TR-024, Software Engineering Institute, February (1993)
20. Pulford, K., Kuntzmann-Combelles, A., Shirlaw, S.: A quantitative approach to Software Management: The ami Handbook. Addison-Wesley, Wokingham, England (1996)
21. Shewhart, W. A.: Statistical Method From the Viewpoint Of Quality Control, Dover Publications, New York (1986) reprint of the 1939 original

Starting SPI from Software Configuration Management: A Fast Approach for an Organization to Realize the Benefits of SPI

Kunihiko Ikeda and Yasuyuki Akamatsu

OMRON Corporation, Control Technology Laboratory, Software Engineering Group, 9-1 Kizugawadai, Kizu-cho, Soraku-gun, Kyoto, 619-0283, Japan
{kunihiko, aka}@ari.ncl.omron.co.jp
www.omron.com/

Abstract. For an organization that has just begun the SPI initiatives, making developers understand the benefits of SPI and gain active participation are critical success factors. Meanwhile, within the level CMM® 2 KPAs, the SCM KPA has a broad range of directly involved stakeholders. Therefore, as an SEPG, the authors have decided to implement the SCM process firsthand to such an organization developing a series of embedded software systems. The following activities are keys for successful SCM implementation: a) Creating an SCM deployment plan from easy, single project implementation to complex, multiple projects implementation. b) Establishing systematically the SCM Web where all stakeholders share plans, processes, and FAQs. c) Administering questionnaires to stakeholders, analyzing responses and improving SCM quickly and frequently. As a result, the developers conformed to the processes and no SCM related problems were found throughout testing.

1 Introduction

For an organization that has just begun the SPI (Software Process Improvement) initiatives, making developers understand the benefits of SPI and gain active participation are critical success factors. The authors assumed that developers who had successful experiences of SPI during the early stages of SPI activities would eagerly and actively participate in latter stages. Within the level 2 Key Process Areas (KPA) in the SW-CMM®[1], Software Configuration Management (SCM) involves not only project leaders but also developers. Therefore, the SEPG (Software Engineering Process Group) members and the project leader decided to focus on adopting SCM to a certain project. The SEPG member also played the roles of an SCM administrator, where he developed an SCM plan, organized SCM support staff and implemented SCM processes within SCM support tools. The adoption succeeded without any serious confusion of development. Through the adoption, the developers understood the benefit of SPI and began to provide positive suggestions about SPI, which were not seen before the adoption.

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 92–104, 2004.

2 Background of Organization

The organization develops various series of embedded software for a certain domain.

In the domain, requirements to software differ by customers and software for each customer has to be customized over long period of time. The organization has to develop and maintain complicated version evolution of software products and components. The organization has begun SPI (Software Process Improvement) activities based on SW-CMM® not very long ago. Most of developers have little or no experience of SPI success.

The authors decided to focus a certain project in the domain. The project was the reconstruction of embedded software, i.e. the development of the common software architecture. The aim of the project was the cost reduction of the customization for customer's requirement by reusing the common software architecture. The project used an iterative software lifecycle model based on object-oriented software analysis, design and programming using UML and C++.

3 Adopting Software Configuration Management

In order to meet the project's objective to enhance the reuse of common software architecture, it was critical to adopt SCM process to identify, manage and control each version and each series of software products. In this section, procedures to adopt the SCM process are descried in detail.

3.1 Planning

The authors planned the step-by-step adoption of software configuration management process in order to avoid the risk of confusion of the project by adopting new technologies that were unfamiliar to the developers. The plan is summarized in Table 1. Orientations were held at the beginning of each development phase. During these orientation sessions, the software artifacts developed in the former phase and the procedures used in the next phase were explained to the developers.

Table 1. Incremental plan for SCM adoption

Step	Operations
1	Planning, version management of specification documents, basic training
2	Version management of design models and test cases, version branch and merge
3	Version management of source code, makefiles and unit-testing scripts
4	Defect information management
5	Decomposition and relocation of repository, multi-cite SCM management

3.2 SCM Tool Selection

To appropriately manage complex series of software and effectively implement SCM process to the organization which had little experience in it, the authors considered that powerful SCM tool supports with the following requirements were essential:

- Version control tool must have abilities
  - to manage multiple and concurrent version branches,
  - to integrate with other tools to provide seamless user operations.
- Change control (Defect tracking) tool must have abilities
  - to control process execution,
  - to customize its GUIs and functionalities easily,
  - to notify specified stakeholders by e-mail,
  - to integrate the functioning with version control tools.

Additionally, their tool vendors were required to provide technical support and training courses with expertise in SCM tools and SCM process itself.

3.3 Organizational Structure

Fig. 1 shows the organizational structure and overall flow of software configuration management. Besides the basic practices of SCM, the process paid special attention to provide services and support to better implement the process; an SCM administrator managed Q and A sessions with developers and tool vendors, and tool engineers of a tool vendor offered SCM training courses and on-site/off-site consultation.

3.4 Environment

The authors adopted sophisticated SCM tools, IBM Rational ClearCase® and ClearQuest® to construct computer-aided SCM environment (Fig. 2), which was built on Windows 2000 domain network. A software repository was implemented on a ClearCase® server PC. A defect information database was implemented on a SQL Server 2000® sever PC. Developers were able to access the repository or the database from ClearCase® or ClearQuest® client PCs.

3.5 Version Management

In order to assure the correctness of the version management of team development, the versions shared by the project members and the versions used only by an each developer were distinguished from each other. The authors developed a process to utilize integrated branch type and private branch type features of ClearCase® (Fig. 3).

The process defined as follows:

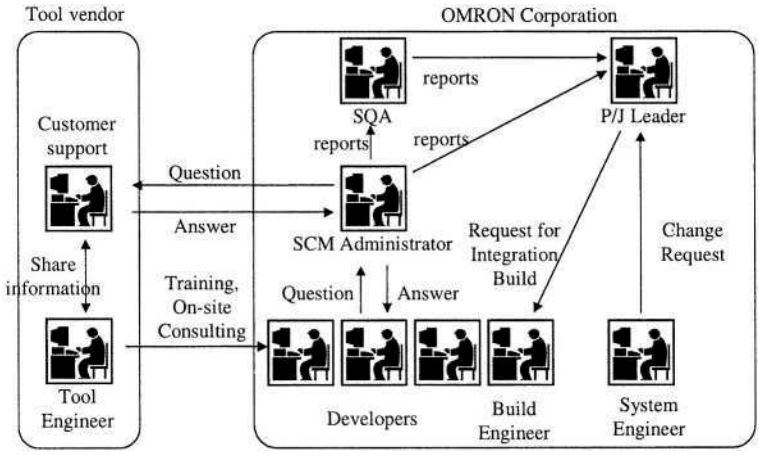


Fig. 1. Organizational structure from an SCM point of view

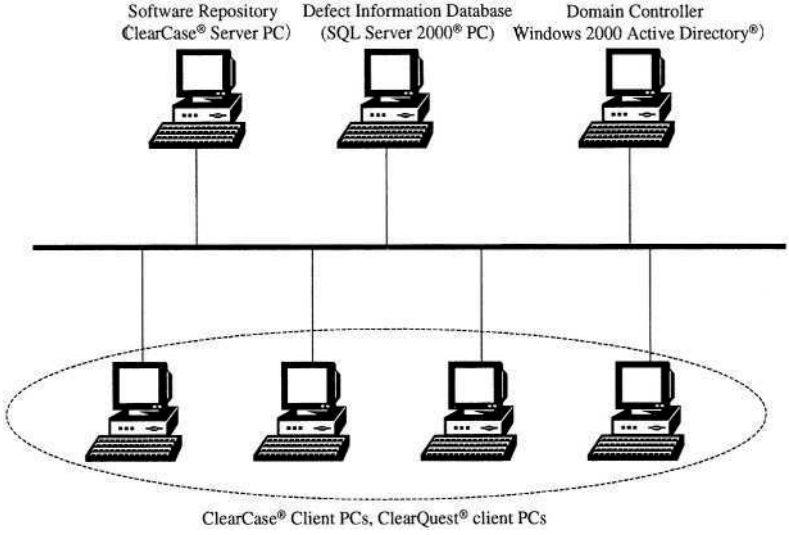


Fig. 2. Computer-aided SCM environment

1. To initiate an individual work, a developer generates a private branch by checking out from an integrated branch.
2. After modification of the checked out element is complete, the developer merges this modified version to the latest version of the integrated branch.
3. The SCM administrator sets a label to the version
4. The latest version is designated as the baseline of the development and the next release to the developers.

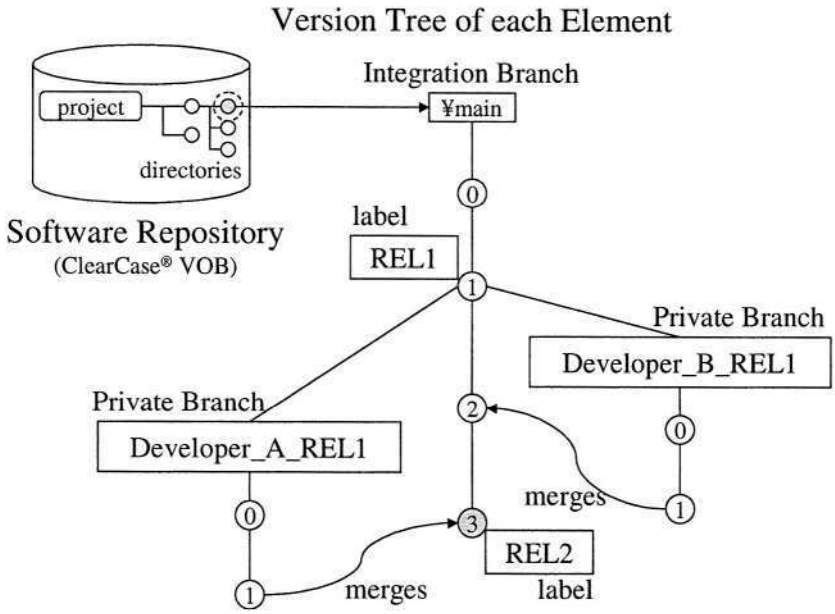


Fig. 3. Version branch policy

4 Adopting Change Control Process

In this chapter, the change control process and its implementation are described.

4.1 Workflow

In Fig. 4, the workflow of change control in functional testing phase is defined. In this figure, in case that a tester detects a defect, he/she reports the phenomenon of the defect to CCB. CCB reviews the report and assigns an appropriate developer to fix the defect. The developer analyzes the defect and fixes artifacts by checking out and modifying appropriate items. After the completion of the fix, the developer updates (i.e. checks in) the artifact. The tester retests the updated version of artifacts and

informs the result of the retesting to CCB. CCB reviews the result and judges whether the task is completed or not. If the result is fine, the task is closed. If not, the task is rejected and the developer analyzes and debugs the defect again.

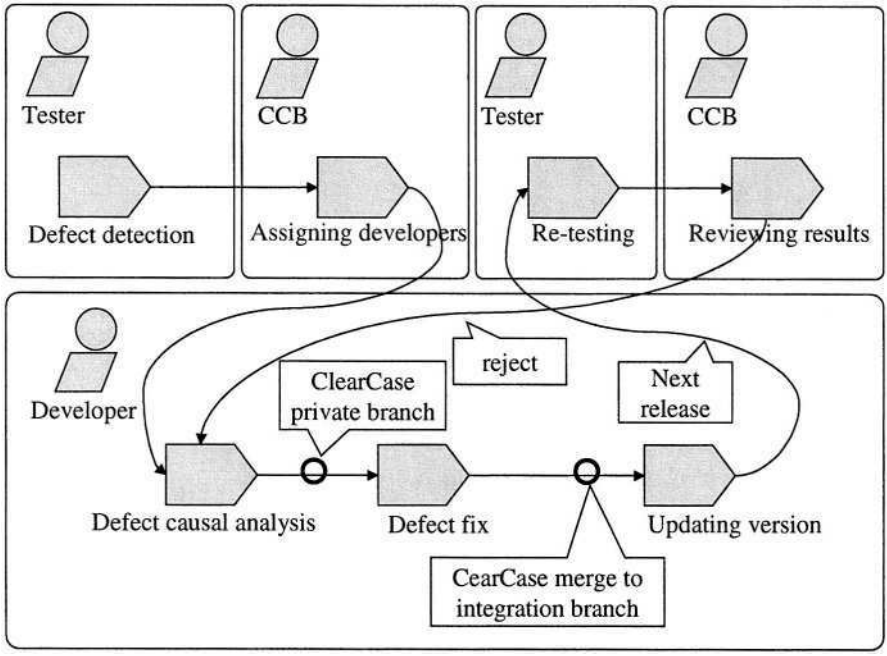


Fig. 4. Workflow of change control

4.2 State Transition Model

From the viewpoint of each defect, the workflow is equal to the state transition model of a defect record (Fig. 5). The model was implemented on the schema of ClearQuest® that made developers perfectly keep the workflow. ClearQuest® sent an e-mail of the defect information to the stakeholders at the both time of submitting a defect and assigning a developer to the defect.

4.3 Graphical User Interface and User Guide

In order to increase the accuracy of defect information recorded by testers or developers, graphical user interface and user guide were developed. In order to record defect information in timely manner according to state transition model, several input forms were developed. The user guide provided definitions and descriptions of each input fields in each input forms which helped and facilitated developers and testers to record defect information accurately. Fig. 6 is a sample of the input form.

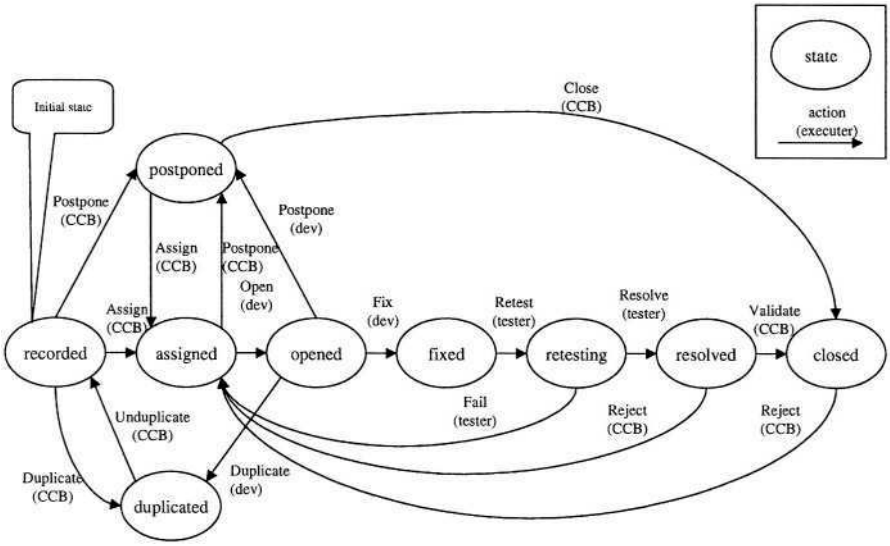


Fig. 5. State transition model of a defect record

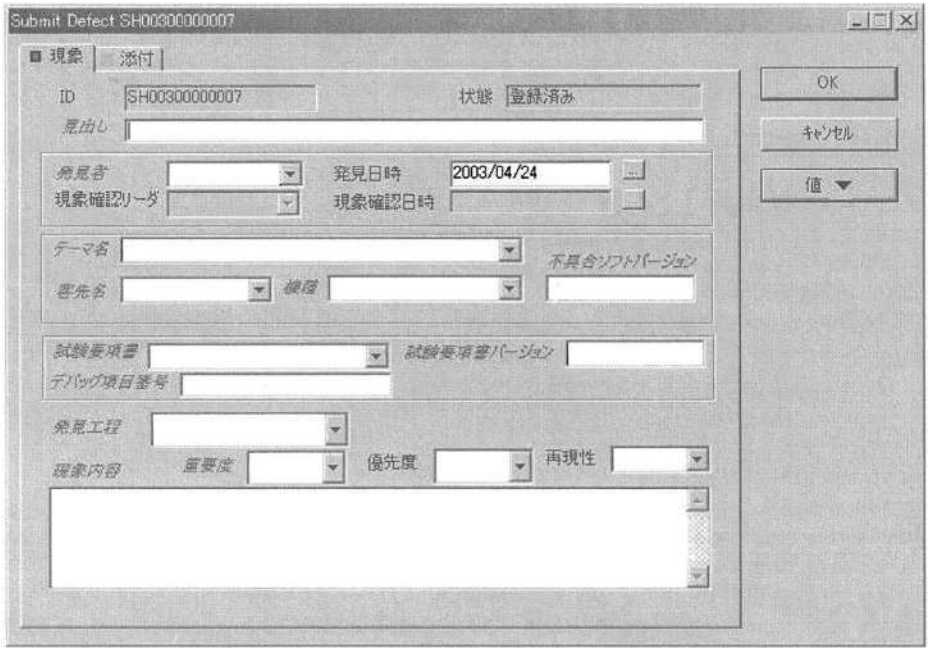


Fig. 6. Defect information input form (in Japanese)

4.4 Regression Testing

In order to reduce the cost of executing regression testing, the authors adopt CppUnit testing framework and developed "AllUnitTest", a set of scripts that covered all packages of the software products. AllUnitTest was executed in functional testing phase. Fig. 7 is the workflow of the regression testing using AllUnitTest.

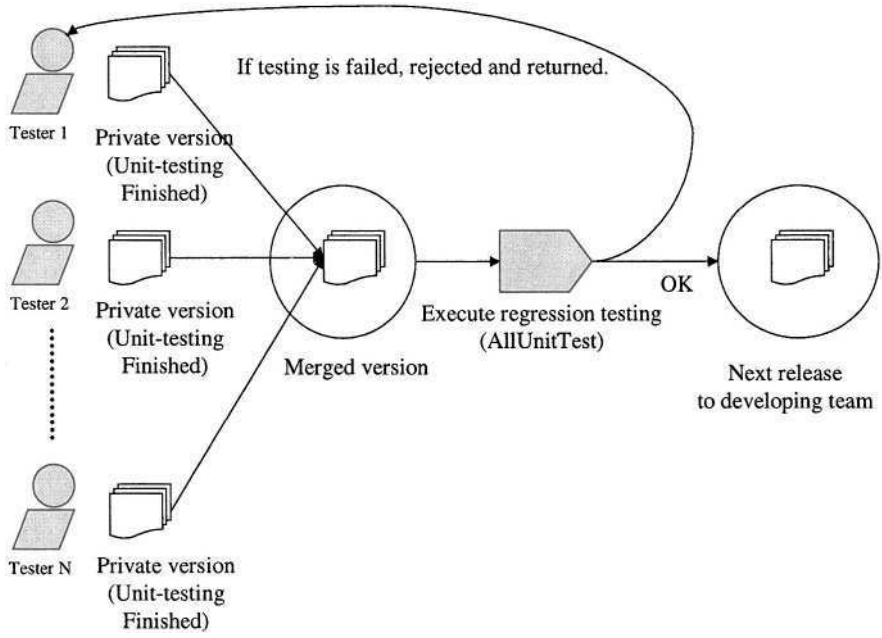


Fig. 7. Workflow of regression testing

4.5 Progress Management

Everyday, both the number of total and completed test cases of each functional unit and status of each defect were reported to the project leader. A thirty-minute team meeting was held every morning. During the meeting, developers reported their status of defect fixes and issues that needed discussion within the project. In cases where resolutions of issues could not be made immediately, additional time was spent for further considerations. The defect listing reports generated by ClearQuest® contributed to the cost reduction of preparing data for the meeting.

4.6 Visualization of Defect Data

Defect data inputted by developers and testers was calculated and analyzed from various points of view. The results of analysis were offered in various kinds of graph

or chart (Fig. 8). For example, percentage of each defect cause type was calculated and visualized in a bar chart.

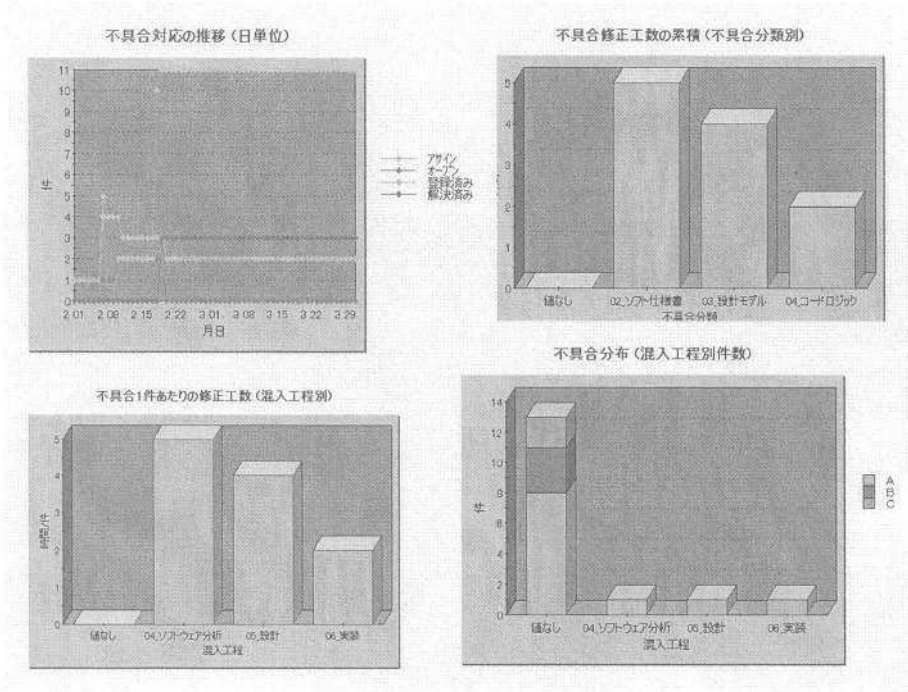


Fig. 8. Sample graph and chart of defect data analysis (in Japanese)

5 Technical Transfer

Approximately half of developers did not have any experiences of software development projects using any SCM tools. Moreover, they did not have systematic knowledge of SCM. So, the authors trained developers to understand the concept of SCM and the operation of SCM tools at the beginning of the project.

5.1 Training Courses

SCM training courses were planned, developed and executed to help developers understand and to perform SCM activities appropriately (Table 2).

Table 2. SCM training courses

No.	Course name	Description	Time	Trainer	Trainee	
					SCM administrator	Developers
1	Introduction of SCM	Concept and Process	3 hours	SCM administrator	-	must
2	ClearCase® Basic	Tool operation for users	2 days	Tool Vendor	must	must
3	ClearCase® Administration	Tool operation for administrators	2 days	Tool Vendor	must	-
4	Introduction of Change Control	Concept and Process	2 hours	SCM administrator	-	must
5	ClearQuest® Basic	Tool operation for users	1 day	Tool Vendor	must	must
6	ClearQuest® Administration	Tool operation for administrators	3 days	Tool Vendor	must	-

5.2 SCM Homepage

In order to share information about SCM with all project members, the authors developed "SCM homepage", SCM knowledge database that contained SCM concept, basic policies, procedures for users and administrators, FAQs, repository configuration map, training resources and so on (Fig. 9). An SCM administrator maintained the SCM homepage. For example, if procedures of SCM changed, an SCM administrator updated the procedure documents on the SCM homepage.

5.3 Questionnaires

In order to understand the effectiveness and developer's satisfaction of the SCM process and tools, the authors administered questionnaires to stakeholders once a month, analyzed the responses and improved the SCM process quickly and frequently. The questionnaire contained about 20-30 questions and required about 20-30 minutes to answer. It asked the evaluation of operating process of SCM tools, SCM training courses, work effort of SCM activities and so on. The result of analysis was documented as an analysis report and presented to all stakeholders.

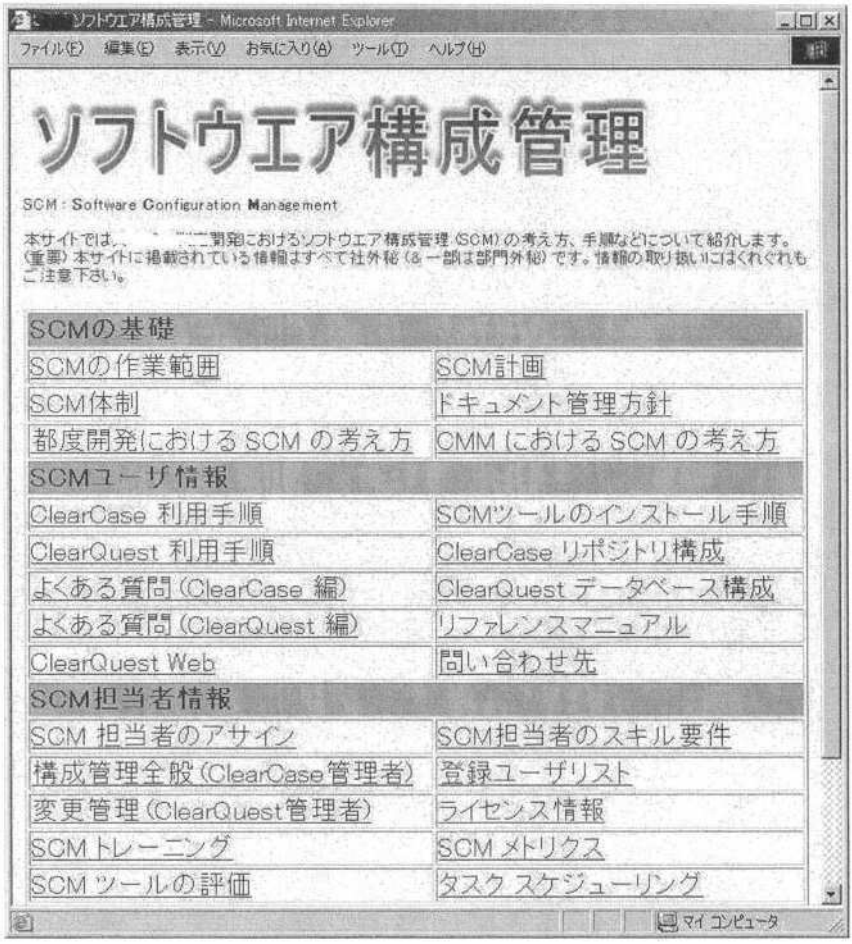


Fig. 9. Top page of SCM homepage (in Japanese)

6 Discussions

In this section, the effectiveness of starting SPI from software configuration management is discussed.

6.1 The Comparison of KPAs in SW-CMM® Level 2

In order to assess the effectiveness of starting SPI from SCM, the SCM KPA was compared with other KPAs in SW-CMM® level 2. The method used for the comparison was to see who directly participates in executing the SCM processes of the project. The authors found that the SCM KPA had broader range of directly involved stakeholders than other KPAs in SW-CMM® level 2 (Table 3). Many kinds

of stakeholders had opportunities to have successful experiences of SCM during the early stages of the project and to understand the benefit of SPI.

Table 3. The comparison of KPAs in SW-CMM® level 2

KPA	Stakeholders directly involved in the project
Requirements Management (RM)	Project leader, Developers
Software Project Planning (SPP)	Project leader
Software Project Tracking and Oversight (SPTO)	Project leader
Software Subcontract Management (SSM)	Project leader
Software Quality Assurance (SQA)	SQA members
Software Configuration Management (SCM)	SCM administrator, Project leader, Developers, Testers

6.2 Benefits of SCM

The authors measured defect records reported in functional testing phase of the project and validated that there was no defect caused by any failure of SCM activities. This fact indicates that the SCM process and tools contributed to the improvement of reliability of the software products. The authors have conducted further survey with developers of the organization that some problems occurred in another project of the same domain.

The authors assessed the answers to the questionnaires and found that the testers recorded all defects that detected in testing phase. This fact indicates that all detected defects were surely tracked from the detection to the closing without deviating from the state transition model of a defect record. It contributed to well-regulated project management.

6.3 Defect Data Collection

Many kinds of defect data were recorded into Defect information database through ClearQuest® input forms. Graphical user interface and user guides are considered to contribute to the cost reduction of recording defect data and the friendly cooperation of developers and testers. On-demand visualization services of defect data are also considered to be a contributing factor to promote the defect data collection.

7 Conclusions

In this paper, the effectiveness of starting SPI from software configuration management is discussed. The authors found that this approach gain active participation of developers. It is very important for an organization at the beginning of

SPI to involve all stakeholders and make them understand the benefit of SPI through successful experiences of SPI. Among KPAs of SW-CMM®, SCM satisfies it. The authors plan in the near future to study the effective KPA order of SW-CMM® and CMMI[SM] [2] based SPI.

References

1. Paulk, M.C., Weber, C.V., et al.: Key Practices of the Capability Maturity Model® Version 1.1. CMU/SEI-93-TR-25, Software Engineering Institute (1993)
2. Capability Maturity Model® Integration (CMMI[SM]), Version 1.1. CMU/SEI-2002-TR-004, Software Engineering Institute (2002)

Evaluating the Calmness of Ubiquitous Applications

Jukka Riekki[1], Pekka Isomursu[2], and Minna Isomursu[3]

[1] Department of Electrical Engineering, P.O.BOX 4500,
90014 University of Oulu, Finland
Jukka.Riekki@ee.oulu.fi
[2] Nokia Corporation, P.O.BOX 50, 90571 Oulu, Finland
Pekka.Isomursu@nokia.com
[3] Department of Information Processing Science, P.O.BOX 3000,
90014 University of Oulu, Finland
Minna.Isomursu@oulu.fi

Abstract. Evaluating the characteristics of ubiquitous applications is still in its infancy although this area is rapidly gaining importance as such applications are becoming common. We present a framework for evaluating the degree of calmness in ubiquitous applications. Calmness is a characteristic of the system that is experienced by the user in the usage situation, hence our evaluation framework is targeted at evaluating the technology in real usage context. We first identify the characteristics of calmness from the user's perspective and then use them to define the dimensions for our evaluation framework. As the framework is subjective in nature, calmness is not presented as numerical values but as graphics. The suggested presentation gives an overall understanding of the application characteristics at a glance. Further, it facilitates understanding the effects of different design and implementation choices and comparing different applications. We illustrate the framework by evaluating an application being currently constructed. This application will be evaluated in real usage context at the beginning of year 2004. We also compare related work with ours.

1 Introduction

This paper defines a way to evaluate a characteristic called *calmness* in ubiquitous systems. Calmness is a user-centric measure that describes how a system appears to the user; it is a user experience that occurs when technology is used in a specific usage context. When a system is calm, it does not unnecessarily interfere with the user but quietly supports her by providing the required service, exactly when and where needed and as unobtrusively as possible.

Ubiquitous computing is expected to have a tremendous effect on our lives. Computers will be everywhere around us, connected to each other and serving us. However, to become a success, the ubiquitous applications have to be calm. Who wants to have hundreds of computers around, offering cryptic user interfaces, constantly requiring attention, and bombarding us with irrelevant information? From the user's

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 105–119, 2004.

perspective, calmness is essential and should be understood better in order to develop good applications. However, it should be noted that calmness is just one view to the applications' quality.

As calmness will be such an essential topic in the forthcoming years, we hope to open a dialogue that would lead to a thorough understanding of this characteristic. We have a large research project ongoing to study the technology and user interfaces of ubiquitous applications. As a part of this, we have created and analyzed application scenarios for ubiquitous computing. The evaluation framework was defined for evaluating how clearly these applications show the desirable characteristics of calmness.

The framework can be used similarly in other research projects that tackle ubiquitous computing. It allows one to analyze the effects that different design decisions have on the calmness of the application. Furthermore, different applications can be compared.

Our research is part of the CAPNET project (Context-Aware Pervasive Networking). CAPNET is a nationally-funded three-year project that started at the beginning of 2002. The aim of the project is to create technologies for new interactive ubiquitous services that adapt to the resources at hand, according to user profile and usage environment, allowing tailored communication between local environment and mobile terminal users.

2 Defining Calmness

In order to evaluate the degree of calmness, we first need to define the characteristics of calmness in a ubiquitous application. These characteristics are identified in this section. The next section presents our evaluation framework that is based on these characteristics.

According to the dictionary [1], 'calm' means "free from agitation, excitement, or disturbance." Therefore, disturbing is opposite to calm. Weiser & Brown [2] state that calm technology is a new approach to fitting ubiquitous computing to our lives. Weiser defined a ubiquitous system as calm in his visionary paper [3], although he did not use the term 'calm.' Instead, he stated that "computers will be used unconsciously to accomplish everyday tasks."

As calmness is a characteristic of the application that is experienced by the user in a usage situation, it cannot be evaluated in vacuum. It can be evaluated only by the real users who are using the application in a real usage context. What is experienced to be calm by one user, may be experienced to be distracting by another user in another usage situation. Therefore, it is essential to understand the usage context and possible reasons for distraction when high calmness is aspired.

To define the characteristics of calmness, we need to consider what kind of ubiquitous application does not disturb its user. We classify the characteristics of calmness as characters producing *calm timing* and characters producing *calm interaction.* Calm timing means that the application interacts with the user in the right situation. Calm interaction is defined as interaction fitting into the situation. A main point in this

classification is that the characteristics are defined from the user's perspective, and they are related to interaction between the user and the technology. The characteristics are discussed below.

2.1 Calm Timing

Calm timing requires from the application *availability* and *context-sensitive timing*. The term 'ubiquitous' itself implies that the application has to be available everywhere and at anytime the user wants to use it. It does not comfort the user to know that the application would have served her well if it had been available. Availability requires sufficient coverage from the utilized technology. In addition, the application has to be capable of adapting to a varying number of resources in the dynamic local environment. In Satyanarayanan's [4] terms, the application has to be capable of "masking uneven conditioning." For example, if the local environment does not provide wireless connection, the application might hide this by using the values stored in its cache. We include fault tolerance in masking uneven conditioning, since the number of resources can vary due to system failures of various degrees.

Due to availability, interaction is possible everywhere and at anytime when it might be needed. Context-sensitive timing in turn constrains interaction to the situations in which it is needed. Weiser [3] mentions location-awareness as an important feature of a ubiquitous system, but a more thorough understanding of the context is required to adapt the application to the situation at hand, e.g., the collection of nearby people and devices [5]. User's goals, desires, etc. are also included in the context, although in practice it can be quite challenging to discover them reliably, and hence to utilize them in the applications.

In the framework, the focus is on the user, so 'context' and 'situation' are defined from the user's perspective. The scope and the fidelity of the context are based on user's observation capabilities and knowledge. For example, if an application is not able to provide context-sensitive timing and it is judged that "the situation has no effect on an application's behavior," it means that the user cannot recognize such an effect. That statement is true, for example, when the user is always notified about incoming calls in the same manner. From the system's perspective, the situation has an effect on its behavior. As the system knows about the incoming call before the notification, it can be said that it performs a context-triggered notification, triggered by the piece of context "incoming call pending."

2.2 Calm Interaction

Calm timing enables interaction but keeps the application out of user's attention when she does not need any interaction with it. When interaction is needed, it should aim for minimal user distraction [4]. This can be achieved by calm interaction, which we specify by two features, *relevancy of interaction* and *courtesy of interaction.* When interaction is calm, only relevant information is presented and relevant input is asked. Furthermore, consideration is shown for the user; information and input queries are

presented in a manner that minimizes the distraction. An essential property is that the application does not force itself too much to the center of attention. Calm interaction requires context-awareness but also the capability to utilize suitable interface technologies and modalities.

High relevancy of interaction can be accomplished by reasoning the information and queries to be presented on the basis of user's context. Here, the context information needs to be used not only for timing, but also for deciding what to present to the user and how to do that. Information that is not necessary is not presented. Further, the user is not queried for information that is irrelevant or can be discovered by other means.

High courtesy of interaction is achieved when the interaction fits well in the situation; the user does not have to unnecessarily switch her focus from her ongoing activities. When the application is able to serve the user well, it – in Weiser and Brown's [2] words – moves back and forth between the center and periphery of the user's attention, as required by the situation. Weiser and Brown use the term 'periphery' to name "what we are attuned to without attending to explicitly."

For example, an email application can move to the center of attention by automatically opening a user interface on a digi-TV screen and simultaneously giving an audio signal. This would interrupt the user while she would be concentrated in, e.g., watching TV. The same application can use the periphery more efficiently by notifying the arrival of new mail by sophisticatedly changing the color of a detail in a piece of art that presents various state information. The user can then move the email application to the center by opening its window on the digi-TV screen.

In addition to utilizing both the center and periphery of attention, a second requirement of courtesy is, that the interaction is implemented in a natural manner; the user is comfortable with the interaction and does not experience it as distracting. The tangible bits approach suggested by Ishii and Ullmer [6] might produce high courtesy of interaction. In their approach, interaction is coupled with everyday physical objects and architectural surfaces. For example, their ambientROOM system illustrates how information can be communicated at the periphery of attention. The systems utilizes ambient media – such as ambient light, shadows, sound, airflow, and water flow – to display and communicate information.

However, the implementation producing the highest courtesy value depends on the user and the usage context. For example, a young user accustomed to modern technology probably experiences different things to be distracting than an elderly user who never has mastered that technology. Also, interaction methods that are not distracting in a home environment might be distracting in a public space such as a library or a concert hall.

Although interaction parameters of our definition for calmness do have common components with the traditional definition of usability, they have a different kind of emphasis. Usability is traditionally defined as the efficiency of how the user can reach her goals using the application and associated technology, whereas the goal of the calmness is to support the user in performing her normal activities without requiring her to concentrate on the application or technology. Also Mankoff et al. [7] point out

the difference between evaluating systems with clearly-defined tasks and systems that support the user but do not necessarily require much attention.

To conclude this section, a calm application is available everywhere, interacts with the user at the right moment, presents relevant information and queries, utilizes both periphery and center of attention, is easy and natural to use, and fits well into the situation. The calmness of an application can be evaluated by estimating to which extent each of the features specified in this section are present in the application. The framework for evaluating the calmness is the topic of the next section.

3 Evaluation Framework

Based on the characteristics of calmness that we identified in the previous chapter, we can now identify the dimensions and scales for our evaluation framework. We do not present exact quantitative equations for calculating the value of calmness, but qualitative values for each dimension and a graphical presentation.

As calmness is a user-centric measure, the framework is user-centric as well: the characteristics specifying calmness are defined from the user's perspective. The evaluation should be done by the real users in a real usage situation; they should test the application and judge how calm the application is on the basis of their experience. In practice, compromises are required because innovative and complex systems developed in research projects tend to be less reliable than required for such testing.

A second important note to remember is how ubiquitous applications differ from the current desktop applications. Desktop applications are in most cases chosen by the user to act as tools for achieving a usage goal; the user decides to interact with the application. User decides which application to use and then aims for executing an activity efficiently through the usage of an application. On the other hand, ubiquitous applications support the activity the user is engaged in. A key difference is that the interaction is often initiated by the application, not by the user, and hence the application should be able to judge when and how to interact. Furthermore, desktop applications are used mainly at the office environment, whereas ubiquitous applications are in continuous interaction with the users during their everyday life. Our evaluation framework is targeted for ubiquitous applications.

Our framework has two stages. Firstly, all four characteristics of calmness are given one of the values *very low, low, medium,* and *high.* Secondly, a graphical presentation is drawn. The values of the characteristics are marked on the corresponding axis, and the marks are connected by lines. The shape and size of the drawing illustrates the calmness of the application and gives an overall understanding of its characteristics at a glance. The graphical presentation is illustrated in Figure 1, in which each measure has the value *high.* Guidelines for giving values for the characteristics are given below.

Availability. The user should feel that the service or application is always available when she needs it. Interpretation for each value of availability:

- *High :* Practically always available for the user when needed. Even when a fault or uneven conditions occur, the user can perform the desired function normally, without much difficulty or disturbance.
- *Medium :* Usually available when needed. When a fault or uneven conditions occur, the user can perform the desired function, but it is rather difficult or disturbing.
- *Low :* Available only at some desired locations. Severe location, temporal, or other restrictions apply to the availability. When a fault or uneven conditions occur, the user can perform the desired function, but with severe difficulty or disturbance.
- *Very low :* The availability of the application does not correlate with the user's desire of where and when to use it. When a fault or uneven conditions occur, the user is not able to perform the desired function at all.

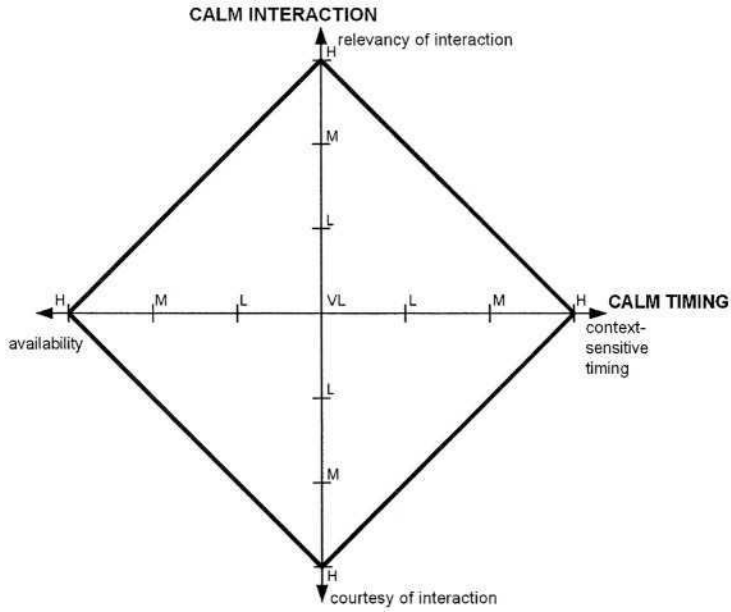


Fig. 1. Graphical presentation for the degree of calmness. 'VL' means very low, 'L' low, 'M' medium, and 'H' high. This shape is obtained when each characteristic has a high value.

For example, the availability of an email service is high, when it is used through a mobile terminal, the utilized network covers all places the user visits, and caching hides temporarily the network's problems. Failures in the mobile terminal and server's software and hardware in this example are supposed to be rare. Medium availability could mean, for example, manual reconnections in some situations or some blind area in the network's coverage. Availability could be judged to be low when emails can be read only when WLAN connection can be established. When emails can be read only at fixed workstations, availability is very low.

Context-Sensitive Timing. User experiences that the application is able to recognize reliably when it should interact with the user. Furthermore, interaction is initiated automatically at the right situations without the user having to remember how or when to initiate it. As applications cannot ever have complete knowledge of the context, perfect timing is not required for high values. Interpretation for each value is as follows:

- *High :* From the user's perspective, interaction occurs nearly always at the right moment. When interaction occurs, the user agrees that it was the right moment, and user does not want interaction when it is not offered. The interaction either supports the ongoing activity, or, when it interrupts the activity, the user agrees that the interruption was justified.
- *Medium :* Interaction occurs mostly at the right moments. In some situations, user disagrees about the need of interaction when it is offered or wants interaction when it was not offered. The interaction either offers unnecessary support or interrupts unnecessarily the ongoing activity.
- *Low :* Only some situations have effect on when interaction is offered. In most cases, user disagrees about the need of interaction.
- *Very low :* The situation has no effect on when interaction is offered.

For example, offering private and non-urgent emails to the user automatically when commuting and during lunch and coffee breaks might deserve a high value for correct timing. If the urgency of private emails would not be recognized by the application, and their delivery would hence be delayed, the value could drop to low. Finally, if the private emails would always be shown when they arrive, value could be very low. This example shows how strongly calmness is related to the user preferences. If the user would like to read emails with different topics at different contexts, then criteria correctness of timing would change accordingly.

Relevancy of Interaction. Only the information and queries relevant to the interaction are presented to the user. The user should not be distracted by unnecessary information and not suffer from lack of information either. Interpretation for each value is as follows:

- *High :* No irrelevant information, queries, or choices are presented. User gets just the needed information.
- *Medium :* Most of the information and given choices are relevant.
- *Low :* Some modest adaptivity to the situation can be observed, but irrelevant information and queries are frequently presented to the user; some information might be missing as well.
- *Very low :* Too much information is presented. Lots of irrelevant command choices are presented in addition to the relevant ones. User needs to search commands or right options from menu hierarchies or from long lists. She experiences that the situation has no effect on presented information or choices. Also failed adaptation gives this value.

High value could be given to the relevancy, for example, when the arrival of a new message is notified by changing the color of a detail in a piece of art and the user associates this change with a new mail. Medium value could be given when an email application shows only the relevant emails but irrelevant commands are presented by the user interface (and the user is used to seeing those commands). Low value might result when user expects to see only personal emails after working hours but only part of the work-related emails are filtered out. Very low value would result when all emails and all command choices are shown always. An example of failed adaptation is adapting menus in a way that confuses the user.

Courtesy of Interaction. The application effectively uses the periphery and center of attention, supporting the user in her actions whenever needed and as much as needed but not consuming her attention more than necessary. Interaction fits well into the situation. Interpretations for each value are the following:

- *High :* The application fluently supports or substitutes the user's actions. Depending on the need in each situation, the application moves from total invisibility to the periphery or to the center of attention. It is in the center of attention only when absolutely necessary. The interaction is implemented in a fashion that the user judges as natural.
- *Medium :* The application unnecessarily demands attention from the user but the disturbance is not severe. Periphery is not utilized when it would be sufficient to present information at the periphery. The user is able to continue her normal actions.
- *Low :* Major disturbance, the user needs to switch attention focus from her normal actions to the service or application.
- *Very low :* The user needs to interrupt her ongoing activity and fully concentrate on the interaction with the system.

An example was presented in the previous section. Notifying the arrival of new mail by changing the color of a detail in a piece of art could give a high value for courtesy of interaction. If the user is watching TV when an email application's user interface is opened on a digi-TV screen, then courtesy is very low.

4 Examples

Next, we present an application scenario from the CAPNET research program and our results of evaluating its calmness. This illustrates how the design choices to be made when designing and implementing an application can be evaluated from the calmness point of view during the design process. We are currently implementing a prototype system that offers the functionality utilized in the scenario. The scenario is presented in the first subsection. In the second subsection, we evaluate the prototype's calmness. As the prototype will be complete only at the beginning of year 2004, we cannot yet present final results that would be based on real usage situations. In the third subsection, we illustrate the framework with some ubiquitous applications presented by others.

4.1 Business Meeting Scenario

Peter works in the CAPNET project. Today he has traveled to Helsinki. As the first thing tomorrow morning, he is having a CAPNET Board meeting in Oulu. At the airport, Peter decides to prepare a short presentation for the meeting while waiting for his flight back to Oulu.

At the airport, Peter checks what equipment will be available in the meeting room. The personal assistant in his PDA presents the list of services that are available in the meeting room the next day. Peter writes the presentation and attaches the completed presentation document into his calendar for the meeting.

The following morning when Peter is at home, his personal assistant gives a notification concerning the meeting. The notification informs that Peter will have the meeting in about 30 minutes and he has enough time to go there.

When Peter arrives into the meeting room, the profile changes automatically from 'general' to 'meeting' and the personal assistant presents the list of services that are available in the room. Peter selects the data projector from the list and then he selects the attached document. Peter's presentation appears on the wall and Peter keeps his presentation.

While Peter is in the meeting, Sarah is trying to call him. Sarah selects Peter from her phonebook and calls him. Actually, the call is not made, but Peter's status is checked. Sarah gets notification that Peter is in a meeting. Sarah's personal assistant inquires: "Would you like to leave a video message or get notification when Peter's meeting is over?" Sarah decides to record a video message to Peter. In the message, she asks Peter to call her. Sarah sends the video message to Peter.

After the meeting, the personal assistant informs Peter: "You have a video message from Sarah. Would you like to see it?" Peter watches the video message and decides to call Sarah later.

4.2 Evaluating the Calmness of the Business Meeting Scenario

A PDA that has the application is connected to the network through GPRS and WLAN. WLAN is used where available; otherwise GPRS. Furthermore, the application is built from components that can be run either on the terminal or in the network. Hence, components can be run in the terminal when necessary, so uneven conditions can be masked (supposing the system is prepared for such conditions). We therefore presume the availability to be high. However, not all users want to carry a PDA with them at all times. These users may experience availability to be lower, if the application is available only in a PDA. Furthermore, the actual user experience is strongly influenced by the reliability of the system to work as planned.

We would expect a high value for context-sensitive timing as well, if the application reliably works as described. At the beginning, the user starts the interaction, so the timing cannot be evaluated. Alternatively, good timing would result if the system would remind of the presentation at the airport. The application reminds in the morning that it is time to go to the meeting. In the meeting, calls are blocked automatically

by setting the profile to 'meeting.' Required services are suggested at the right moments by listing them on the PDA's display. The message is shown after the meeting, which we assume to be the correct timing for our user as the message was not very urgent.

The interaction at the beginning is highly relevant, if Peter can easily find and use the needed applications and information. When Peter enters the meeting room, the available services are listed automatically on the PDA's display. The data projector is listed as well, and Peter can start the projector service by clicking the item in the list. Furthermore, a list of documents related to the meeting is presented automatically and starting the presentation requires only clicking the right item in the list. The interaction in the morning is highly relevant, if only the important details about the meeting are presented.

For the courtesy of interaction, medium value can be reached as the PDA will be used for all interaction, and hence periphery could be utilized more. For example, if the PDA beeps to remind Peter, he has to grab the PDA and read the reminder. Medium value is justified, if Peter is used to this interaction and he can, after reading the reminder, continue his normal actions without major disturbance. Higher value could be achieved, for example, by utilizing the sound system of Peter's home, or virtual display on his eyeglasses.

A graphical presentation of the measures for calmness of this application is shown in Figure 2. It shows that, to further improve calmness of the application, the main target for improvement would be the courtesy of interaction.

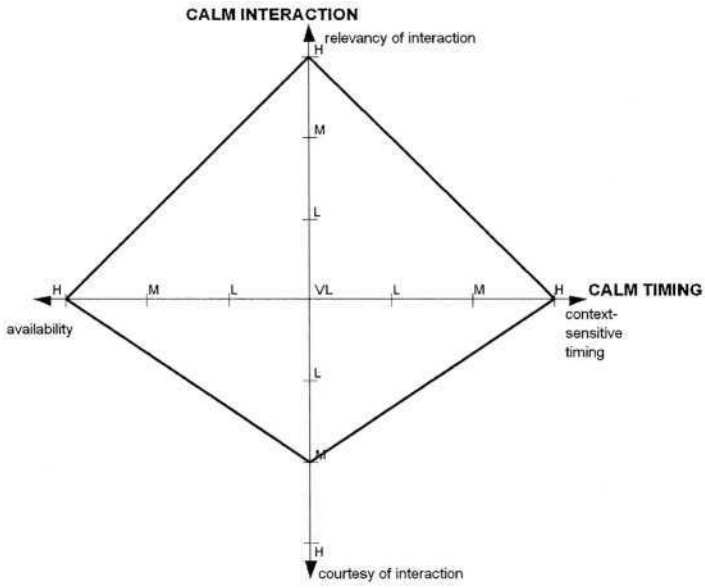


Fig. 2. Calmness of the Business Meeting Scenario.

As a conclusion, we can expect that the design decisions discussed above would most likely result in a quite calm application implementation. As discussed elsewhere in this paper, the actual evaluation should be made by real users in real usage situations but this we can carry out only after the prototype has been completed. A second note is that, although the implementation is quite challenging, a high value for calmness is relatively easy to achieve, as the offered functionality is limited. Bigger challenges arise when ubiquitous applications offer more versatile functionality for different users. Then, achieving high values becomes much more challenging as there are much more options for when and how to interact.

4.3 Evaluating Some Related Work

To illustrate the usage of our framework in different application contexts, we discuss two more applications in this section. Again, the given evaluation should be seen as trendsetting only, as it is based on written documentation of the applications, and not on an evaluation by real users in real usage situations.

Ambient displays [6,7] are promising candidates for high values of both relevancy of interaction (when context is recognized at sufficient detail) and courtesy of interaction. Mankoff et al. [7] point out that the primary goals of ambient displays are relevancy of information, aesthetics, and ease of monitoring. These goals match our definitions of high relevancy and courtesy of interaction. On the other hand, ambient displays might get a low value on availability, if they are not ubiquitously available wherever the user would need the information (unless the information is related to the user by some other means). Furthermore, if the display is not adapted to the user's situation, the value of context-sensitive timing might be low as well.

The prototype presented by Ishii and Ulmer [6] is a good example of an ambient display. It presents each web-hit to a web page as the sound of raindrops. The sound of heavy rain indicates many visits to the web page. This sound would probably otherwise remain at the periphery of user's attention but draw attention when the sound changes considerably. If the user is interested to know whether the web page attracts visitors and wants to be interrupted only when the amount of visitors changes considerably, the application might provide just the right information without unnecessarily disturbing the user. However, even in this case, careful experimenting is needed to achieve high courtesy values. Ishii and Ulmer noted that at times the sounds of rain could be distracting. They implemented also a less distracting prototype in which web-hits cause ripples on the illuminated water surface.

Proactive Instructions for Furniture Assembly is a system for assembling furniture that the user buys from a shop and then assembles by herself at home [8]. By attaching computing devices and sensors onto different parts of the assembly, the system can recognize the user's actions and determine the current state of the assembly. The service supports the user's normal actions and only corrects the user when she goes wrong in the assembly process. However, in this prototype implementation, the user interface for interruptions has not been fully solved, although good ideas are given. Availability is high if the system is included in all the self-assembled furniture.

Timing is high if the user is interrupted only when the assembly would otherwise go wrong. In the reported work, though, the system does not always recognize the context accurately enough, e.g., how many nails the user has hammered in. Relevancy is high when the user is informed only about how she is making progress: if everything goes smoothly she can go on without disruptions, if she's made a mistake the system warns her exactly as it should. Courtesy of interaction is not high if the user needs to stop the assembly in order to look at a computer display to see how she is doing. If the system had a voice interface where the user would be told whether what she had just done was correct and what she should do next, then courtesy of interaction would also measure high.

Graphical presentations for calmness of these two applications are given in Figures 3 a and b, respectively. The ambient display is assumed to be available only at a restricted area which however is included in the set of locations in which the users need the information. Furthermore, elementary context recognition is assumed; the display starts the interaction when a user approaches it. This results in low timing. If just the needed information is presented, high value can be given for relevancy. In the case of the furniture assembly system, the lower courtesy value is achieved when the user needs to switch her attention from the assembly to a display to check how she is doing, but she does not have to interrupt assembling totally. The high courtesy value might be obtained for voice interface, for example.

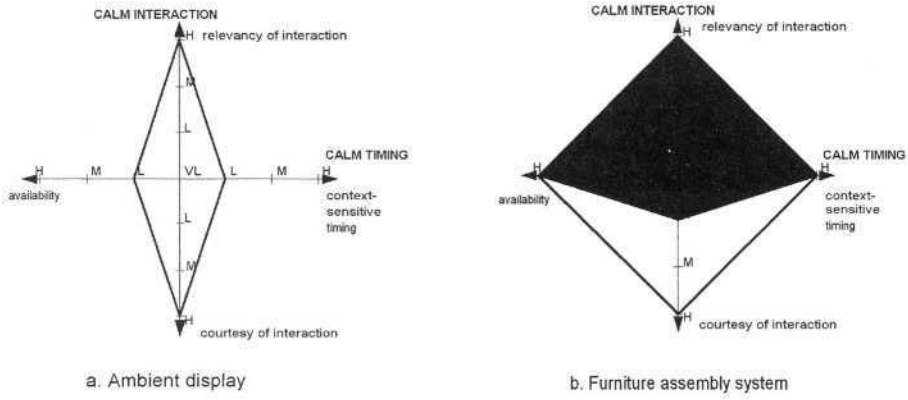


Fig. 3. Calmness of Ambient Displays and Proactive Instructions for Furniture Assembly. The black shape shows the calmness resulting from selecting a display for interaction and the white shape (identical with Figure 1) the result when a voice interface is used.

5 Discussion

The framework presented in this paper allows one to evaluate the application's calmness and to analyze the effects that different design decisions have on the calmness. The evaluation can either focus to some predefined set of situations, or the situations

in which the application is calm can be searched. Of course, all possible situations can be evaluated as well, but that is quite challenging.

The situations in which a certain application is calm can be found by evaluating the four characteristics in the order they are presented here. Before evaluating, minimum values are defined for each characteristic. The first stage results in a set of situations in which the availability is high enough. At the second stage, the context-sensitive timing is evaluated and a smaller set of situations is obtained in which both availability and context-sensitive timing exceed the set minimum values. After evaluating all characteristics, a set of situations has been found in which the application is calm (if such situations exist).

When the application is evaluated in a predefined set of situations, special attention has to be paid to guarantee that a low value of one characteristic does not have an effect on the evaluation of another characteristic. For example, if the context-sensitive timing has a low value when the user is observing relevancy of interaction, the relevancy might be judged to be too low if the poor timing is not taken into account. A low value of one characteristic can even prevent evaluating another characteristic.

Based on the presented analysis and examples, it can be said that calmness is a challenging feature. In addition to being able to offer services everywhere the user needs them, the situation of the user has to be recognized. Even if the state of the environment could be recognized at a sufficient detail, the goals and desires of the user often remain largely unknown.

The value of the availability can be high even when some software or hardware needs to be installed before the actual service can be used – if the user experiences the installation as a part of using the service. This is one example where personal preferences and background affect the experienced calmness. For example, some users might feel that installing a plug-in into the Internet browser before watching a video is quite a natural part of the task, while others might experience it distracting and might even cancel the download.

The research on ambient displays illustrates how high courtesy of interaction can be achieved – and high relevancy as well when the context can be recognized. But the availability and courtesy of interaction can be difficult to achieve at the same time. Ambient displays can produce high courtesy of interaction only at a certain limited region. Ubiquitous courtesy requires such interaction everywhere the user needs it; either fixed ambient displays at each location or, for example, augmented reality displays that the user carries with her.

The difference between availability and context-sensitive timing is that when availability is low, interaction is not possible at all at certain situations but when context-sensitive timing has a low value, the application is available but the interaction is not started automatically in right situations. By observing only the interaction offered by the application, it cannot be judged which one is the reason for distraction; low availability or poor timing.

We have not found any evaluation framework that would as fully cover the area of calmness as our framework does. Indeed, the topic is not much discussed in the literature and with this paper, we hope to open a dialogue about evaluating calmness. Abowd and Mynatt [9] discuss the challenges of evaluating ubiquitous computing

research and state that there are no simple guidelines for steering research efforts. The framework presented here could provide such guidelines for improving the calmness of ubiquitous applications. Furthermore, they present that a spectrum of evaluation strategies is needed from technology feasibility efforts to long-term use studies. At this spectrum, our framework can be positioned near the long-term use studies.

Mankoff et al. [7] present a technique for evaluating the usability and effectiveness of ambient displays. Their heuristics resembles closely our framework's relevancy and courtesy of interaction. However, they do not consider availability or context-sensitive timing, except that their heuristics "Error prevention" corresponds with availability; with its requirement of masking uneven conditioning. This approach of heuristic evaluation might work well in our framework as well.

In their paper on PlantCare, LaMarca et al. [10] discuss the problem of providing users with pleasant and meaningful interactions with practical ubiquitous systems. They believe that the key solution is to emphasize autonomy and thereby minimize demands on users. We agree, as more autonomy means less interaction that needs to be developed to be calm. They state that user-oriented performance metrics are more meaningful for ubiquitous systems than traditional measurements. They propose metrics such as the amount of time people need to spend installing the system, or the amount of time the system takes to restart after a loss of power. We consider the latter to be a subset of availability, part of our "masking faults and uneven conditioning" measure. Installing a system can be considered to be separate from the system's actual function, and our whole framework can be applied to evaluating it separately.

At the Ubicomp 2001 conference, a workshop was organized on the Evaluation Methodologies for Ubiquitous Computing [11]. Their report states that "We need to agree upon the dimensions for evaluation, metrics that are useful for researchers and meaningful from the end-user perspective." Workshop participants formulated "the beginning of a framework for evaluation." The framework includes four axes: Universality, Utility, Usability, and Ubiquity. It is designed for measuring ubiquitous systems in general, not calmness alone, so it is natural that some measures are included that are rather irrelevant for us. Their framework can be used to evaluate systems from a more holistic viewpoint.

We can map into our framework other characterizations of ubiquitous applications. Weiser defined that a ubiquitous system should be effortless, unobtrusive, and must fulfill a real or perceived human need. Different aspects of effortless and unobtrusive systems are covered by all four of our basic characteristics. The last feature we consider irrelevant from the calmness point of view: an application can be calm even if it doesn't serve any human need, the interaction just must not distract the user. Abowd and Mynatt's [9] ubiquitous application themes include natural interfaces and context awareness, which can be mapped into context-sensitive timing and relevancy and courtesy of interaction. For technical requirements, they identify scaling and 24/7 availability. These are handled by our availability measure. Satyanarayanan's [4] characterization of pervasive computing includes minimal user distraction and masking uneven conditioning or managing variations of "smartness." Our whole framework is about minimal distraction, and masking uneven conditioning is part of availability.

The next step in developing the framework is to use it in evaluating our prototypes in real usage situations. In this work, creating heuristics like in Mankoff et al's [7] work offers interesting possibilities. We expect our framework to evolve and grow in detail as we obtain more experience in evaluating calmness.

References

1. Merriam-Webster's Collegiate Dictionary. WWW: http://www.webster.com/cgi-bin/dictionary. 2002.
2. Weiser M & Brown JS: The Coming Age of Calm Technology. In: Beyond Calculation: The Next Fifty Years of Computing. Denning PJ & Metcalfe RM (eds.) Springer-Verlag, New York. (1997) 313 p.
3. Weiser, M.: The Computer for the 21st Century. Scientific American (Sept 1991) pp. 66-75
4. Satyanarayanan, M.: Pervasive Computing: Visions and Challenges. IEEE Personal Communications Journal, Vol. 8. IEEE (2001), pp. 10-17.
5. Schilit, B. Adams, N. and Want R.: Context-Aware Computing Applications. In: Proc. of IEEE workshop on mobile computing systems and applications, 1994.
6. Ishii, H & Ullmer, B.: Tangible bits: towards seamless interfaces between people, bits and atoms. In: Proc. of the SIGCHI conference on human factors in computing systems, March 1997. pp 234-241.
7. Mankoff, J., Dey, A.K., Hsieh, G., Kientz, J., Ames, M., Lederer, S.: Heuristic evaluation of ambient displays. CHI 2003, ACM Conference on Human Factors in Computing Systems, CHI Letters 5(1): 169-176. 2003.
8. Antifakos, S., Michahelles, F., and Schiele, B.: Proactive Instructions for Furniture Assembly. Proc. of UbiComp 2002, p. 351.
9. Abowd, G. and Mynatt, E.: Charting Past, Present and Future Research in Ubiquitous Computing. ACM Transactions on Human-Computer Interaction. Vol. 7. ACM (2000) pp. 29-58.
10. LaMarca, A., Brunette, W., Koizumi, D., Lease, M., Sigurdsson S., Sikorski, K., Fox, D., and Borriello, G.: PlantCare: An Investigation in Practical Ubiquitous Systems. Proc. of UbiComp 2002, p.316.
11. Scholtz J and Richter H.: Report from Ubicomp 2001 Workshop: Evaluation Methodologies for Ubiquitous Computing. SIGCHI Bulletin (Abowd, G., ed.). WWW: sigchi.org/bulletin/2002.1/janfeb02.pdf.

Quality Attributes in Mobile Web Application Development

Axel Spriestersbach[1] and Thomas Springer[2]

[1] SAP-AG, Corporate Research, Germany
axel.spriestersbach@sap.com
[2] Dresden University of Technology, Germany
springet@rn.inf.tu-dresden.de

Abstract. The paper deals with quality attributes for mobile web applications. It describes typical challenges when developing mobile web applications and relates the challenges to the ISO 9126 quality attributes. The quality attributes are summarized to an ISO model that is focusing on the most important quality attributes for the quality assurance of mobile web applications. Finally, the paper proposes that applying formal quality assurance methods during the development of mobile web applications may solve some of the challenges in mobile web application development.

1 Introduction

The mobile Internet promised flexibility and cost efficiency comparable to the normal web. However, experiences indicate that the development of mobile web applications needs to consider special challenges in the areas of usability, development efficiency and runtime consideration.

The major challenge of mobile web application development is the heterogeneity of mobile devices and web browsers installed on the devices. The device specific differences, such as the different form factors and input capabilities, strongly influence the usability of a mobile web application. On the software side the device differ in their pre-installed browsers. Currently most mobile devices, either support WML or subset of (X)HTML. The different markup languages pose threats to mobile web application development. Applications are either automatically or manually adapted to the different languages. Finally the wireless network connection causes additional threats, such as higher network delay, limited bandwidth or dropped connections. These challenges will be investigated in depth in the "3 Challenges for Mobile Application Development" section.

These challenges may be considered as quality problems. However, while several studies deal with quality for e-commerce systems [9] there are no such studies for mobile applications. An adapted quality model that investigates special issues in quality assurance is helpful in two ways. First considering special quality attributes will help developers in increasing the quality of their applications and leverage the acceptance of mobile web applications. Therefore quality assurance methods may contribute in overcoming some of today's challenges in mobile web application

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 120–130, 2004.

development. Second a quality model will support researchers in comparing their approaches in mobile web applications development and device independence.

The paper is organized as follows: the first chapter introduces the ISO 9126 standard. The second chapter describes the major challenges of mobile web application development and relates the challenges to the affected quality attributes. Chapter three summarises the findings of chapter two and indicates the parts of the ISO 9126 standard that are especially affected. The paper concludes with an outlook on future work that is required in order to archive the described goals.

2 Quality Attributes According to ISO 9126

The ISO 9126 is part of the ISO 9000 standard, which is the most important standard for quality assurance. The ISO 9126 defines a set of the quality attributes to be used in a quality assurance process. An overview of the defined attributes is shown in figure 1 below.

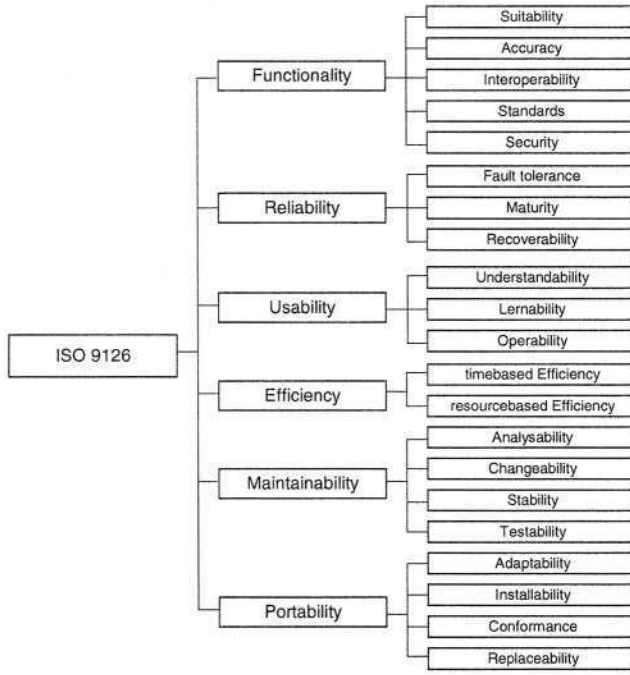


Fig. 1. Overview ISO 9126 standard

The main quality attributes of the ISO 9126 standard are *Functionality, Reliability, Usability, Efficiency, Maintainability* and *Portability* that are further divided into a set of sub-attributes.

Functionality

Functionality is "a set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs." *(ISO 9126: 1991, 4.1).* This quality attribute consists of five sub-attributes: *Suitability* means that the functionality of the application fits the needs of a user to fulfil a certain task without overwhelming the user. *Accuracy* means that the results or application's behaviour is correct. *Interoperability* means that the application is able to interact with specified systems. *Compliance* means that the application is compliant with standards, conventions or regulations in laws and similar prescriptions. *Security* is the ability to prevent unauthorized access, whether accidental or deliberate, to programs and data.

Reliability

Reliability is "a set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time" (ISO 9126: 1991, 4.2). Reliability consists of three sub-attributes: *Maturity* is the frequency of software faults. *Fault tolerance* is the ability of a software to deal with software faults or infringements of its specified interface. *Recoverability* is the capability to recover data affected in case of a failure and measured by the time and effort needed for it.

Usability

Usability is „a set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users" (ISO 9126: 1991, 4.3). The quality attribute consists of three sub-attributes: *Understandability* describes the users' effort for recognizing the logical concept of an application and the applicability of that logical concept. *Learnability* is the users' effort for learning the application as for example, operation control, input and output. *Operability* is the users' effort for operation and operation control.

Efficiency

Efficiency is "a set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions" (ISO 9126: 1991, 4.4). The efficiency of the *time* and *resource* behaviour is distinguished. The time behaviour describes for instance processing times and throughput rates while resource behaviour means the amount of resources used and the duration of use.

Maintainability

Maintainability is "a set of attributes that bear on the effort needed to make specified modifications" (ISO 9126: 1991, 4.5). The quality attribute consists of four sub-

attributes: *Analyzability* is the effort needed for the diagnosis of deficiencies or failures and for the identification of parts to be modified. *Changeability* is effort needed for modification, fault removal or for environmental change. *Stability* is the tolerance of the application towards unexpected effects of modifications. *Testability* is the effort for validating the modification.

Portability

Portabiltiy is "a set of attributes that bear on the ability of software to be transferred from one environment to another" (ISO 9126: 1991, 4.6). The quality attribute consists of four sub-attributes: *Adaptability* is the opportunity for adapting the application to different environments without additional effort. *Installability* is the effort for installing the software. *Conformance* means the conformance of an application to standards or conventions relating to portability. *Replaceability* is the opportunity and effort to use an application as a replacement for another application.

3 Challenges for Mobile Application Development

This chapter describes the typical challenges for the development of mobile web applications and relates them to the ISO 9126 quality attributes described in the last chapter.

Usability

There are implied restrictions on interfaces for user interaction on mobile devices compared to a normal desktop browser. Porting an application from a desktop to a mobile version therefore requires adaptations with respect to output, input and navigation.

The output capabilities of mobile device are determined by their screen, which can range from small monochrome to VGA/SVGA size displays. The screen limits the amount of information for simultaneous display and therefore the bandwidth for user interaction. Applications need to consider this limitation, for example by distributing information across multiple pages or adapting the content of the application [8] [7].

The limited input capabilities are due to size constraints of mobile devices that are usually not equipped with a full size keyboard. The performance for data input is therefore compromised. This must be taken into consideration by applications that should limit the amount data input to the minimal required data. In addition, an application developer should select those input widgets that are most appropriate for the input hardware. For example, when designing a date entry field for a mobile phone, an application should use numeric input widgets (i.e. an input field) instead of selection widgets (i.e. a DropDownBox) since mobile phones are usually equipped with a numeric keyboard that supports numeric input best.

An example shows the effects of different user interface widgets when using a device with limited input capabilities. We evaluated multiple versions of a widget for inserting a date on a Motorola Timeport mobile phone. Fig. 2 and Fig. 3 show two

alternative versions: one using input fields (Fig. 2), the other one using selection boxes (Fig. 3). For the first version the user uses the numeric keypad to insert the date, for the latter version the restricted navigation facilities of such keyboards come into play – the version is far less usable on the mobile phone.

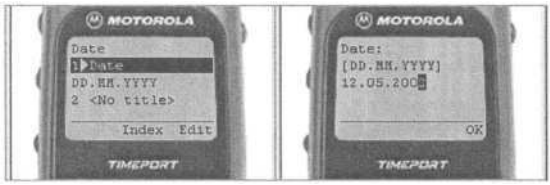


Fig. 2. Date Input - Keypad version

Fig. 3. Date Input - Select version

During the user trial we found that on the average the users needed 18.3 seconds to complete version one while they needed 23.7 seconds for version two. This indicates that for the mobile phone numeric input methods should be used in favour to selection based methods. Similar results for the Nokia 7110 indicate that this is the general trend for devices with numeric keypads.

Another important observation is that the navigation concept needs to be adapted to the device. For instance, applications that are running on devices equipped with a touch screen should support a hypertext based navigation, while for the same application a menu based navigation may be more appropriate on a devices equipped with selection buttons only.

Generally, the functionality of web applications targeted for the use on a desktop are too complex for a direct transformation to mobile devices. Mobile web applications should be focused and simplistic in order to minimize the required interaction. Therefore, not only the individual user interface widget needs to be adapted, the overall complexity of the web application should not overwhelm the user. Functionality that is not required to complete a task should be left out for those devices with restricted user interfaces.

In addition mobility influences the quality attribute operability for interactive tasks - data input and navigation. The hardware limitations require additional user attention to operate the web application when inserting data or navigating. Additionally, the surrounding environment has an impact on the interaction. A user may be distracted and/or in motion when using his mobile device, therefore the accuracy, speed and correctness of the input may be lower.

In conclusion, the quality attributes usability and functionality are especially important and currently a major threat for the success of mobile web applications. Therefore the quality attributes usability, operability and especially suitability need special consideration when designing and implementing mobile applications. Quality

assurance methods should be applied during the development process so an application meets the required quality standards.

Technology Impact

Technology is no longer a guarantee for commercial success. Economics is an important factor for the success of mobile web applications. The costs – network and application - for using an application needs to be considered and a service has to provide an added value to the user that is willing to pay for the service. Currently, the costs for using a service consist of a service fee and network costs. We focus on the network costs, which currently have a greater impact compared to non wireless networks. In GSM networks the network costs are either time or volume based. Mobile applications should therefore try to minimize the time a user needs to complete a task and the amount for data send to the client. This contributes to an application's suitability as well as a time and resource based efficiency.

Suitability and costs have another effect on mobile web applications. The applications are more focused and the usage times are reduced. Therefore the times invested for understanding and learning about the application should be reduced as well. Otherwise, the users could reject the application. This goal may be reached by a common applications design or by using use patterns. However the quality attributes learnability and understandability need special consideration.

Mobility and Context Changes

As mobile web applications are used in different environments, a mobile web application is subject to continuous context changes. The context influences the application, which needs to adapt to those context changes. For example, the location is irrelevant to an office situation where location changes are infrequent. For a mobile web application this situation is different. When used on the road the location is very important, information that may be used to increase the suitability of an application. For example the approach described in [10] uses a user's location to replace certain location dependent user inputs in order to decrease interactivity of the mobile web application. The decreased interactivity decreases the usage times that contributes to the suitability of the application.

A changing context is the nature of mobile web applications that must adapt to those context changes. Context changes have a strong influence on the software engineering process [5] of an application as well as quality engineering. The quality attribute suitability is especially affected.

Wireless Network

In the early days of mobile internet access, technology problems, such as network stability and networks delays, were identified as major issues. Nowadays communication technologies (e.g. devices, networks) have reached a level of maturity that allows abstracting from those issues. However, the differences in the quality of service of mobile networks still differs from the quality of service of fixed networks

and therefore need consideration in the development process of mobile web applications. Network latency and the limited bandwidth also influence the application development. For example, the delay in loading pictures on a mobile device requires text oriented mobile web applications in order to increase the suitable of the application.

Another challenge for wireless network access is security. Tampering a wireless connection is much easier compared to fixed lines. In addition the limited processing power limits the level of the supported security standards. For example WTLS encryption that is used for WAP devices is less robust compared to SSL that is used in the normal web. The differences require special consideration in requirements engineering and quality assurance - especially the quality attribute security.

Device Heterogeneity

The heterogeneity of devices is causing an increased development effort for mobile web applications. Currently there is a lack of accepted application level models for user interaction and content presentation (UICP) which satisfies the needs of efficient application development and device specific limitation. In addition, maintaining multiple device dependent versions is labor intensive, compared to the a few common alternatives, which have to be maintained for normal web applications. Various approaches are being considered stressing different constraints and expectations.

The mechanisms of the World Wide Web have set the standard for UICP based on PC browsers. A large number of existing HTML pages induced efforts to achieve automatic transformation into representations which can be rendered on handheld devices with considerably smaller screen sizes. Such approaches [1] target page transformations based on syntactical analysis of a given HTML page, element type specific reductions (e.g. image rescaling) or exclusions of content parts based on assumptions about the semantics of HTML content parts. Experiments with such systems have shown that, while generated presentations are legible, the aesthetical quality of achieved presentations is rather low [1].

A second school of thought advocates the development of distinct UICP models to be used in the context of distinct devices. Providing a dedicated UICP model for every possible device ensures maximal UICP adaptation, at the expense of development cost [12]. To alleviate this situation, approaches such as [12][2] consider the provision of UICP models for a number different device classes. Still, UICP definition needs to be done several times.

Regarding implementation efficiency, defining and implementing a UICP model which can be reused for every possible end user device is an ideal approach.

To support such device independent authoring various techniques have been considered including:

- abstract user interfaces mapped onto a concrete UI representation of an employed device [11][13][3])
- content elision and transcoding techniques [1][8]
- presentation structure adaptations such as for dynamic layouting and pagination [8][7].

The challenge common to all approaches is finding an optimal tradeoff between the quality and the development effort. The need for automation requires a high level of adaptability of applications as well as the supporting environment. In an ideal case an automated transformation system requires no additional modifications of the application are necessary. However, such a maturity level is difficult to reach –an evolutionary development approach [4] that starts with an application for a smaller set of devices which is extended to new devices is much more likely. The author of device independent applications is required to partially port or at least add additional meta-information for an adaptation engine for new devices. The frequent changes require a maintainable application. Changeability and stability are especially important for mobile web applications. In an ideal case an author is able to invest the minimal effort required to archive the predefined quality level.

Relation of Development Challenges and the ISO 9126 Standard

The challenges for the development of mobile web applications are partially quality challenges. The fact that a software project is developing for a mobile client needs to be considered in requirement analysis and quality assurance. Table 1 summarizes the relationships between mobile challenges and quality factors according to the ISO 9126 standard.

Table 1. Impact of mobility on ISO 9126

Characteristics of mobile web application	Related Quality Factor
Usability – Output / Form factor consideration	Usability Suitability
Usability – Input / Form design	Usability Suitability Operability
Usability – Navigation	Usability Suitability Operability
Network limitations	Resource based Efficiency Time based Efficiency
Wireless connection	Security Suitability
Context changes	Suitability
Short usage times	Understandability Learnability
Application development for heterogeneous devices	Portability Adaptability Changeability Stability

The table lists the challenges in the development of mobile web applications that have been identified in the course of the chapter on the left hand side and the affected quality attributes of the ISO 9126 standards on the right hand side.

4 Challenges Mobile Application Development – A Quality Problem?

As described in the last section a couple of challenges in mobile web applications and the development of such are related to quality issues. The most urgent challenges for mobile web applications concern increasing the usability and the suitability of mobile applications. Both are important quality factors listed in the ISO 9126 standard. A second challenge is on increasing the efficiency of mobile web application development. The heterogeneity of devices requires porting and adapting the application to new devices with new requirements towards the applications. Increasing the quality of the development processes and products/program code in the areas of portability and maintainability will help to lower the cost when adding a new target platform. Finally, the limited processing and network resources require efficient use of the available resources.

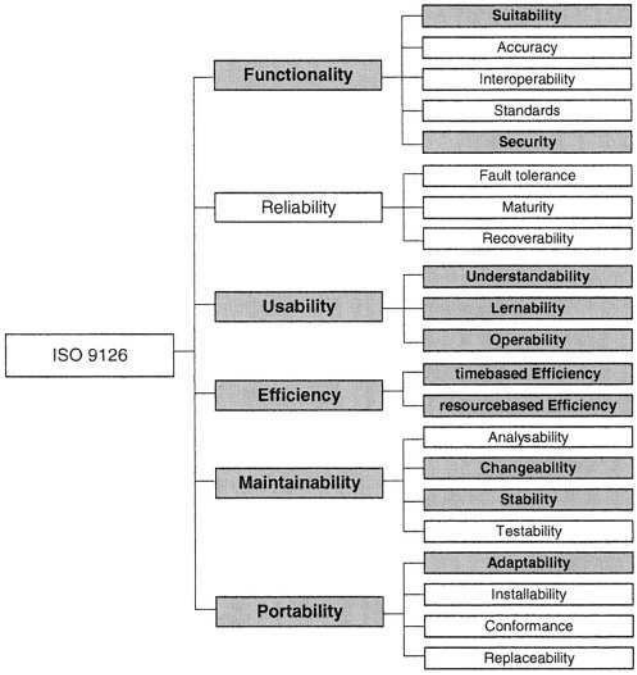


Fig. 4. ISO 9126 - Effects of mobile web applications

An overview of the impacts of mobile web applications on the ISO 9126 standard is shown in Fig. 4. The quality attributes that have been identified of special interest for the development of mobile web applications during the course of the paper are marked bold.

The adapted ISO standard may be used to support the development in tracking the most important quality attributes for mobile web applications. The attributes should be especially considered in the requirement phase. Applying software and quality assurance methods to the development should then lead to increased quality of mobile web applications and therefore better acceptance of future generation of mobile web applications.

5 Conclusion and Future Work

The paper discussed the ISO 9126 standard with respect to the development of mobile web applications. We introduced the standard, explained typical challenges in mobile web application development and linked those challenges to the quality attributes of the standard. The result is a list of quality attributes that require special attention for mobile web application development projects. The final goal of the research is applying quality assurance to help improving mobility challenges.

The next steps for our research is a deeper investigation on the core quality attributes and on methods for measuring those under the requirements imposed by mobile web development, namely the heterogeneity of device that may be too labor intensive, for quality assurance comparable to effort for the development of applications. This may also imply changes to the general software development approaches such as HORIZON [6] process that is used within SAP.

References

1. T. W. Bickmore, B. N. Schilit, Digestor: Device-independent Access to the World Wide Web, Proceedings of the 6th WWW Conference, Santa Clara, CA, USA (1997)
2. I. Cooper, R. Shufflebotham: PDA Web Browsers: Implementation Issues, Technical Report, Canterbury Computing Laboratory, University of Kent, UK (1995)
3. J. Eisenstein, J. Vanderdonckt, A. Puerta: Applying Model-Based Techniques to the Development of UIs for Mobile Computers, Proceedings on Intelligent User Interfaces, Santa Fe (2001)
4. G. Graef, M. Gaedke: An Evolution-oriented Architecture for Web Applications, Proceedings of NOSA '99 (1999)
5. A. Finkelstein, A. Savigni: A Framework for Requirements Engineering for Context-Aware Services, Proceedings 1st International Workshop From Software Requirements to Architectures (STRAW 01) (2001)
6. SAP, Horizon , http://www.sap.com/print/solutions/quality/globaldevqm.asp
7. H. Keränen, J. Plomp: Adaptive Runtime Layout of Hierarchical UI Components, Proceedings of the NordCHI (2002)
8. S. Mandyam, K. Vedati, C. Kuo, W. Wang: User Interface Adaptations, W3C Workshop on Device Independent Authoring Techniques, http://www.w3.org/2002/07/DIAT (2002)
9. A. Stefan, M. Xenos: A model for assessing the quality of e-commerce systems, Proceedings of the PC-HCI 2001 Conference on Human Computer Interaction (2001)

10. A. Spriestersbach, H. Vogler, P. Ebert: Improving the Usability of Mobile Enterprise Applications by Applying Location and Situation Information, Proceedings of the third workshop on Applications and Services in Wireless Networks (ASWN) (2003)
11. User Interface Markup Language, www.uiml.org (2003)
12. Unwired Planet Inc., Developer's Guide Version 1.0, Redwood Shores, CA, USA (1997)
13. Extensible Interface Markup Language, www.ximl.org (2003)

Introducing Quality System in Small and Medium Enterprises: An Experience Report

Lerina Aversano[1], Gerardo Canfora[1], Giovanni Capasso[2], Giuseppe A. Di Lucca[1], and Corrado A.Visaggio[1]

[1] Università degli Studi del Sannio
RCOST – Research Centre On Software Technology
Via Traiano 1 - 82100 Benevento
{aversano, canfora, dilucca, visaggio}@unisannio.it
[2] Elisys S.r.l.,
via Nazionale Appia, 259 -81022 Casagiove (Caserta)
giovanni.capasso@elisys.it

Abstract. The institutionalization of a Quality System improves the levels of technical and managerial efficiency of Enterprises. Moreover, the market itself solicits the acquisition of a Quality Certification for getting a steady confirmation of Enterprise's capabilities. The introduction of a Quality System in Small and Medium Enterprises can entail prohibitive costs for them and affect their agility and flexibility. The paper proposes a lightweight approach as a solution to either avoid or reduce such drawbacks; it consists of a method for redesigning processes and a software system to control and monitoring processes' execution. Consequently, a research question arises: is the approach suitable for establishing a Quality System effectively in a Small Medium Enterprise? In order to have a preliminary evaluation of the proposed approach, a case study has been carried out in an Italian Enterprise, aiming at owning VISION 2000 Certification.

1 Introduction

Establishing a Quality System is widely recognized as a way to improve the productive and economical assets of an enterprise through a best technical-managerial efficiency. The Quality Certification, as for Vision 2000 [9] or CMM [13], also improves the perception of the enterprise on the market. Indeed, it confirms the capabilities of the enterprise to produce and deliver products with constant and defined characteristics.

Certification is also essential to participate in most public tender and private partnership. In summary, the introduction of a quality system is mainly motivated by the market rather than be an autonomous choice of the enterprise.

This has a relevant impact on Small and Medium Enterprises (SME). As a matter of fact, the introduction of a quality system in a Large Enterprise presents several

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 131–145, 2004.

difficulties [8, 12, 10]; it is even more complex for SMEs, due to the following major concerns:

- the cost to be faced for defining and enacting the Quality System;
- the need for not compounding the enterprise organization with formal procedures.

Indeed, one of the main competitive advantages of SMEs is their agility and flexibility, and these could be seriously obstructed by the use of heavy procedures. SMEs cannot easily bear the costs for the introduction and maintenance of a Quality System. Moreover, also the cost of the overall production process may be increased by performing the Quality Assurance activities. A reasonable solution to avoid this drawback could be the use of lightweight [14] approaches in terms of methods and software tools.

This paper proposes a lightweight approach for establishing a Quality System in the context of SMEs and, as research question, discusses the effectiveness of using it.

In particular, the three following research questions have been investigated:

Q_1. The actual processes are compliant with the procedures defined in the Quality System.
Q_2. The cost for defining the Quality System fits well with the budget needs and constraints.
Q_3. The used methods and tools ease the institutionalization of Quality System: they allow the diffusion of the knowledge about the process within the overall enterprise, and the reduction of the costs for executing the processes.

More specifically, the proposed lightweight approach has two perspectives:

- From the methodological point of view a method has been defined in order to design the process model;
- From the technological point of view the use of a software system has been adopted for facilitating the control and monitoring of the process execution.

The chosen open source software system to support the process enactment was GENESIS (Generalized ENvironmEnt for procesS management in cooperative Software) [1, 17]. Thanks to its characteristics of flexibility and adaptability it represents an optimal trade-off between benefits and costs.

In order to give preliminary answers the research question, the paper discusses a case study aimed at the introduction of a quality system in Elisys S.r.l, a SME working in the production of electronic systems, including the control and application software. The Vision 2000 standard for the definition of the Quality System has been used, and a first validation of the approach has been achieved by the application of the Goal/Question/Metric paradigm [21, 22]. Questionnaires have been submitted to the process actors and interviews have been conducted with the enterprise managers

in order to carry out a qualitative analysis on a pilot process. The results encouraged the enterprise to extend the approach to the other processes.

The paper is structured as follows. Section 2 discusses the method used for defining and modeling the production process. Section 3 briefly presents the open source software for process management support. Section 4 gives a description of the case study. Section 5 summarizes the outcomes of the case study. Finally, concluding remarks and lessons learned are given in section 6.

2 The Method Used for Defining and Modeling the Production Process

In the context of SMEs the production process is frequently designed ad hoc during the production activities and it is not formalized a priori. More often, the process is informally defined and partially documented. Then, most of the knowledge about the process is owned by the people involved in the process, under the form of tacit knowledge. Finally, even if the process is defined and documented, when executed actual instances present significant differences with the formalized procedures.

The use of well-defined and documented processes could significantly improve the competitiveness of an enterprise. In this way it is possible to maintain a performance level and facilitate its improvement through the identification of bottleneck or further problems. Moreover, a well defined (and graphically modeled) process can facilitate its understanding and enactment by the actors involved.

Precisely, claimed advantages of modeling processes include [16, 18, 5]:

1. Enable effective communicating the description of a process to others, such as managers, process users, process developers, researchers and users.
2. Reusing the process itself.
3. Support the evolution of the process. Process elements can be saved in a database for modification, learning, and tailoring.
4. Facilitate effective planning, control and management of process.
5. Automation of the process in process sensitive environments.
6. Analysis of static and dynamic inconsistencies in the process.

However, the definition and modeling of a process is difficult and resource consuming. The proposed method to define and model production processes presents two peculiarities, both related to its agility. The first peculiarity is its light formalism, in terms of documentation to be produced and of the procedure's detail level. The second peculiarity is the applicability to different business contexts.

The main expected effects are: the simplification of procedures; the reducing of the effort for documenting the processes; and finally the impact on costs, that are directly connected with the costs of no quality.

Other practices have been used in the recent and far past, characterized by a strong awareness at well detailed procedures and documentation production. The business process reengineering experiences basically point out such characteristics [7, 11].

The proposed method for process definition and modeling, in the form of a Data Flow Diagram (DFD), is shown in Figure 1. It consists of four main steps described in the following.

- **Analysis of the current process.** The first step to be performed is the collection of all the information regarding the organization and its processes. This is addressed conducting interviews with the personnel of the enterprise in order to understand their objective and work modalities. During these interviews it is important to consider what is already formalized in the enterprise documentation. Then all the Production Procedures formalized are analyzed. The aim is to verify which are the processes (or part of processes) already formalized, understand their structure and their deliverables. The result of this step is the description of the Actual Production Process performed in the enterprise. This description can be both informal (i.e. in natural language) or formal (i.e. with a formal specification of the workflow).

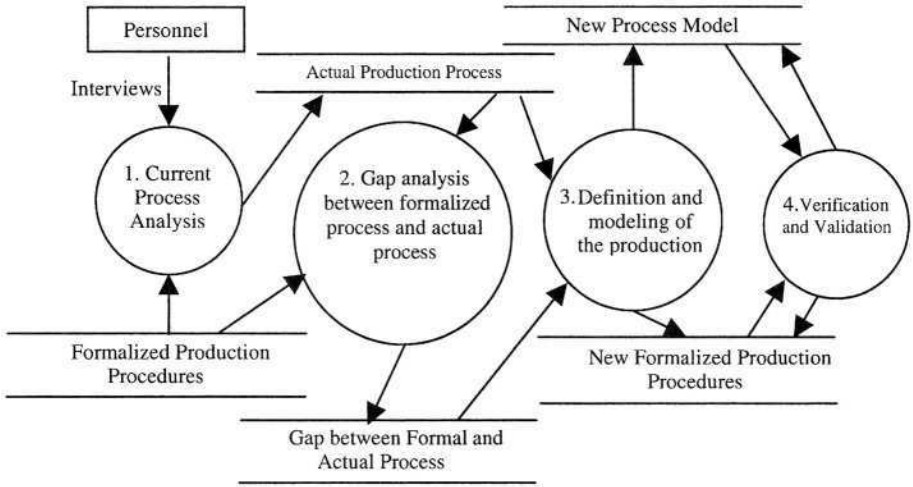


Fig. 1. Reference method for process definition and modeling

- **Gap analysis between formalized procedures and the effective processes.** Where a formal description of the production process exists, it is important to conduct a gap analysis between the formalized process and the process actually performed in order to identify differences, where existing. Such differences can be due to the personnel of the enterprise who does not respect the formalized procedures and tend to modify them without formalizing the changes as this could be too complicate.
- **Definition and modeling of the process.** The description of the performed production process and the analysis of the differences identified are used to define the new process. Then the process is modeled using a graphical representation that facilitates the understanding and enactment by the process ac-

tors. Graphical representation can be achieved by different modeling paradigm, such as UML activity diagrams. This can reduce the typical hostility to the innovations of the personnel and encourage the application of the process itself.

- **Verification and Validation.** The process and the model defined have to be verified and validated through monitored execution of the re-designed process in order to highlight new differences between the formalized process and the performed one. This allows to promptly updating the model and related documentation. At the end of this step the final version of the process model and its documentation are delivered.

3 Genesis: An Open Source Tool for Process Management

It is well known that large enterprises increase their processes quality by using software technologies for process management and control [20, 19, 15].

A representative set is briefly analyzed, including EPOS; SOCCA, SPADE, Spearmint, and then the GENESIS environment is presented.

3.1 Process Modeling Technologies

EPOS [4] (Object-Oriented Cooperative Process Modeling) considers a process as a chain of tasks and products; tools and roles can be assigned to each activity. EPOS lets the process engineer define different kinds of rules for the tasks: dynamic rules for temporal and logic conditions; static rules for consistency check; rules expressing conditions concerning the specific application domain.

SOCCA (Specifications of Coordinated and Cooperative Activities) [3] integrates three different perspectives from which the process can be described. The first perspective concerns the static aspects of the process: what the process produces, what the process requires to be activated, which part of the external universe is affected by the process. In this phase the Entity Relationship Diagram is used. The second perspective regards the transformation of the data set: how data computations are performed through the activities. This perspective is formalized through Data Flow Diagrams. The third perspective deals with the behavioral aspect: which is the specific algorithm that characterizes each procedure. This information is captured through the State Transition Diagram.

SPADE [2] was a joint project of Politecnico Of Milan and CEFRIEL. It is based on the SLANG process model language. The peculiarity of this technology consists of basically two features. Firstly, it relies on formalism of Petri Net in order to express the dynamic aspects of the process. Secondly, it presents different levels of abstraction for the process model, managed with a hierarchical structure. When a more detailed view is required, an activity can be exploded in the next level sub-process. Moreover the same sub-process can be reused in different process models.

Spearmint [23] is an environment for modeling, analyzing and maintaining complex process models, ideated at the Fraunhofer IESE and applied to both internal and external industrial software processes. The main features of Spearmint are summarized in the following. The process is considered as a set of activities that produce, consume or modify artifacts; roles and tools can be assigned to each activity. In order to augment flexibility in providing information about the process, different levels of abstraction, detail and structures are represented. The process can be displayed with partial views in order to avoid the overhead of information to the process user.

The costs of such applications could not fit with the exigencies of small enterprises. The nature of these costs is two-fold: commercial one, that is the marketplace prize, and organizational one, that is the impact of the adoption on organizational processes and competencies. The use of GENESIS represents an opportunity because of two reasons: it is open source and the subjects involved in the experience were well familiar with it.

3.2 GENESIS Environment

GENESIS is an open source platform designed for supporting the cooperative management of software processes within a European Project. It appeared to be suitable for the application in the context of small organization, even different by the area for which it has been defined.

Genesis specifically support the management of software project. Moreover it provides interesting features for cooperation and communication among the people involved in the process. The system is based on a rational combination (goal driven) of product-centric and process-centric views of the design, implementation, testing and maintenance phases of software projects.

The GENESIS platform consists of different subsystems integrated through a common web client and coordination application logic layer:

- Resource Management System: it manages the user data, project data, allocation data of the users in the projects and the access control to the GENESIS platform. Its services can be accessed by the users through the GENESIS manager client module.
- Artifact Management System: it is responsible for creation, modification, deletion, and storage of artifacts. It provides facilities for indexing and searching the artifacts and collecting appropriate metrics.
- Workflow Management System (WfMS): it is in charge of the definition, instantiation, and execution of software processes. It provides functionalities to produce an abstract model of the process, by using the Process Definition Tool, that can be completed later using the Project Management Tool, that allows the creation of project according to process models defined. The Workflow Enactment services allow the management of the to do list and the routing among the process activities depending on events raised within an activity.

- Event Engine: it is in charge of collecting events that happen during the process, such as the production of an artifact or the termination of an activity, and notifying them to the different software components and users of the GENESIS site and other cooperating sites.
- Communication System: it provides both synchronous communications, managed by a forum system, and asynchronous communications, managed by a mail system.

4 Case Study

The approach for process definition and modeling and the GENESIS environment, have been adopted, in Elisys S.r.l., with the aim to verify the real benefits that it can introduce for supporting Quality System.

The Elisys S.r.l. is located in the district of Caserta, Italy. It was found in 1987 by a group of software engineers coming from multinational organization operating in the development of TLC devices. It works in the context of the development of electronics products with high technological content. It owns the production cycle from the ideation of the product to its implementation.

The organic of the enterprise consists mainly of working partners. In the 2001 the personnel was of 16 annual working units, external consultant are involved when needed.

The company already had a Quality System certified by the ISO 9001:1994 normative, but the strategic plan of the company foresees the improvement of the quality system with the adjustment to the certification ISO 9001:2000. Finally, the company had already developed an ad hoc software system to partially support some of the activities of the production processes related to the Human Resource Management and the Supply Chain.

In this organizational context the approach for process definition and modeling and the GENESIS environment has been adopted. The following sub-sections discuss the application of the approach to the *'Design and Engineering Process'*.

4.1 Analysis of the 'Design and Engineering Process'

The first step of the method required an active collaboration of the personnel and two weeks of work.

During this phase, information concerning the organization and organizational processes has been acquired, both by referring to the existing prescriptions of Quality System's documentation, and throughout interviewing personnel.

The interviews aimed at understanding the procedures and the operational behavior used by the personnel for executing all the different tasks, neglecting all that was formalized within Quality's documentation; as a result, the process actually implemented was obtained.

4.2 Gap Analysis between Production's Formalized Procedures and Real Process

In this phase, the process actually implemented was compared with what reported in the Organizational Procedures and relative Operative Instructions; those are part of the certified Quality System's documentation.

In order to achieve this goal, monitoring of Departments was accomplished, with the support of cards summarizing all the exiting documentation that rules the activities of the monitored area; furthermore, ad-hoc produced checklists were used.

As a result of this monitoring, a relevant gap was discovered between the formalized reality, described within the Quality System, and the processes actually performed. This situation highlighted the existence of difficulties in applying correctly the Quality System procedures.

The Quality System was not perceived as 'usable', due to a certain 'bureaucratic heaviness' it introduced and some redundancies it presented. A further cause of the gap was due to the organizational restructuring. In that period the Company was turning to a divisional structure from a functional one.

On the basis of such results, the Organization chose an overall re-organizing approach, in order to face all the concerns arisen from the monitoring, rather than forcing the existing process to be comliant with the one described in the documentation. The motivations stood basically in the higher costs required by this kind of intervention. Furthermore documentation needed to be upgraded anyway, in order to adequate to ISO 9001:2000's body of rules.

Moreover, the need for an Information System (IS) supporting both the Quality System and the flow of the Organizational Processes arose. The IS would have to generate information related to process indicators, for supporting business decisions making of the CEO. Such a lack caused a strong additional discomfort in the correct managing and verifying of Organizational activities.

4.3 Definition and Modeling of the 'Design and Engineering Process'

In this step new processes' models with the documents regarding the Organizational Procedures (OP) and Operative Instructions (OI) were defined and realized. In the new versions of OPs and OIs redundancies, overlapping and inconsistencies, present in the following versions, have been removed.

This made documentation slender, with a reduction of about 50% for the OPs and of about the 12% for the OIs. The processes were graphically modeled by the Activity-Object Diagrams (derived form the UML Activity Diagram): each diagram describes a specific production process; each diagram carries activities characterizing the process, describes the execution's sequence of the activities (parallelism of execution between activities may be expressed also), indicates the input and output for each activity.

Different symbols and notations are used within the diagrams in order to: refer to different kinds of activity; indicate connection points with other processes; identify

the input and output of each activity, the decision points, the process' start and end points; entry notes; define the responsibilities for each activity.

A hypertext web-based system, accessible from the Company Intranet, was realized in order to improve the usability and the accessibility to this documentation. The system let each employee retrieve documents needed for knowing what has to be produced and how to operate, according with the defined Quality System, directly from his/her own workstation.

The system lets everyone know and accede to the modifications done on either the Process model or the OPs and OIs suddenly: on the contrary, the use of the traditional paper based diffusion not ever let that. The Figure 2 shows a typical screenshot of the system realized: in the right, the process' diagram is displayed whereas, in the left, information concerning the selected diagram's element (document or activity) is displayed.

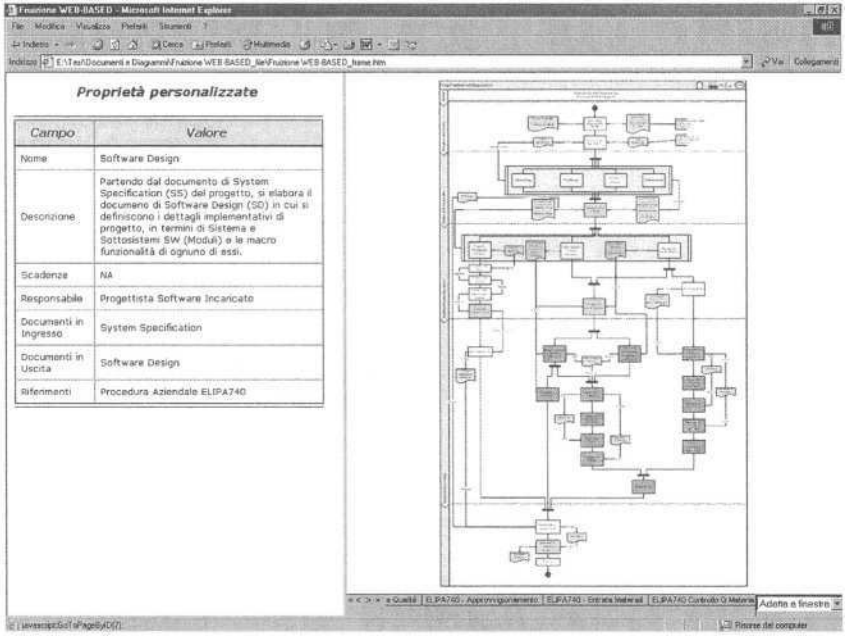


Fig. 2. A screenshot of the web based application realized for supporting usage and access to Quality System documentation

After having defined, modeled and provided with the related documentation all the processes, a support for the automation of the process "Design and engineering" was provided, by the tool GENESIS. This process was used to verify and validate the approach and to evaluate its quality against the customers' expectations, within a real project. The project dealt with the realization of a GSM system based on "satellite technologies", and included the realization of software, hardware and mechanical components.

All the activities and the transitions in the model, together with all the input and output documents' flows of the activities were defined, by using the features of the GENESIS' WfMS component. Then, resources' allocation to the project was defined with the GENESIS' Resource Manager System component.

4.4 Verification and Validation of the 'Design and Engineering Process'

In the last step of Verification and Validation, the "Design and engineering" process was executed, aiming at verifying and validating the proposed approach. The tools utilized for this purpose were those of Training and Monitoring. By ' Training' it is meant a persistent support for the personnel both in terms of comprehension and interpretation of the model and of accomplishment of it, throughout the function "Analysis, Measures and Controls".

The purpose consisted of validating the processes' models and assuring that their modification and evolution was enough easy to quickly tailor them to modification and evolution of the productive processes. This kind of approach was chosen mainly for this reason: the personnel operating the function "Analysis, Measures and Controls", responsible for controlling the process during the execution in term of both produced documentation and quality registrations, was just the same personnel analyzed and modeled the processes; consequently the most suitable for providing such a support.

With the Monitoring the correct execution of the processes in conformity with the models was verified. GENESIS resulted very suitable for the purpose, because well accepted by the personnel, especially thanks to its operational flexibility; moreover, it was considered not much invasive, notwithstanding the evidences of what verified must be communicated officially to the correspondent person in charge.

The execution of this step pointed out that the adoption and usage of hypertext system within the organizational intranet brought a relevant improvement of the actual and correct application of the formalized processes; employees' comprehension of the process to be executed is increased.

The employment of GENESIS let the employees involved in the project and assigned to some activities accede to "ToDo List", see Figure 3, in order to get acquainted with the activities to be performed and their current state (ACTIVATED or SCHEDULED).

5 Analysis of the Case Study

The case study described in the previous section aims at providing a preliminary evaluation of the proposed lightweight approach's effectiveness. The current section discusses the analysis of the experience.

Concerning the research question Q_1 *'The actual processes are compliant with the procedures defined in the Quality System',* useful indices are offered by the *'Gap Analysis'* step. The lightness of the approach helped the identification of the correct

process actually performed by the enterprise. Moreover, the management of both the activities' and documents' flow has been automated in the GENESIS environment. In this way the effort needed for carrying out activities was minimized and ambiguities due to the circulation of not definitive documents' versions were avoided; as a matter of fact, GENESIS let automatically the distribution of stable documentation only, reducing errors.

Finally, the employees performed the process as it was modeled, completely fulfilling the Q_1 research question. The only exceptions were little dissimilarities due to a few residual ambiguities / inconsistencies within the documentation. GENESIS provided the employees with leading and controlling during the activities the process consisted of. This was a further guarantee of process compliance with the Quality System.

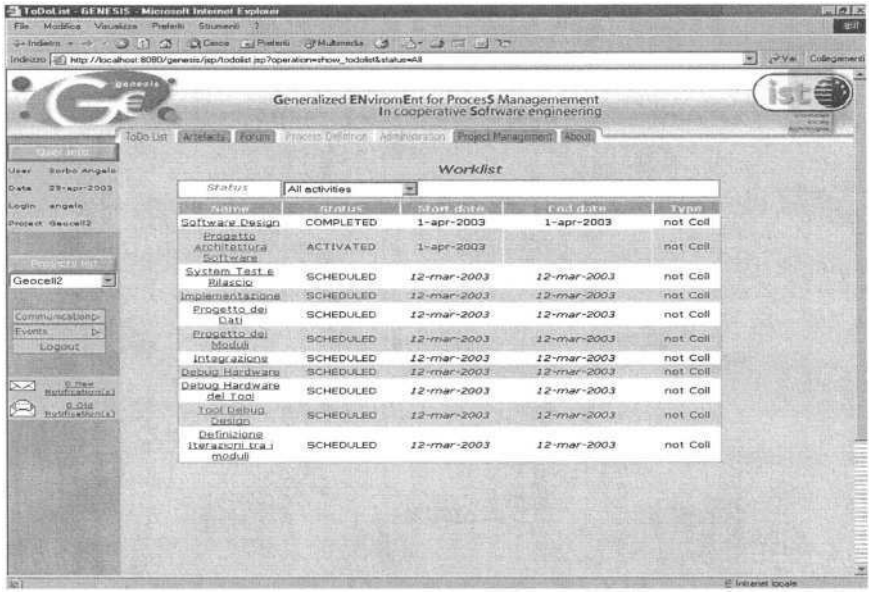


Fig. 3. A screenshot of the GENESIS 'ToDo List'

Concerning the research question Q_2 *'The cost for defining the Quality System fits well with the budget needs and constraints',* it is necessary to introduce what costs mean. On the one hand, it must be considered the cost due to the application of a methodological approach; the proposed approach imposes a few constraints because it is defined at a high level of abstraction. In this way it leaves organization free to specialize its application as well as allowed by the planned expense. On the other hand, the costs due to the introduction of the software should be considered. The use of an Open Source environment could assure cheapness in this direction. In particular, the cost for acquisition of GENESIS was null while the costs of training was low thanks to the background knowledge of people involved in the experience. From the case

study it resulted that the actual costs was close to the expected ones and the confirmation for this has been achieved by an evaluation of the approach's Quality.

The evaluation aimed at comparing the Expected Quality with the Actual Quality. The Expected Quality was defined by the authors with respect to project's and user's documentation of the tool GENESIS; concerning the organizational processes, the authors referred to body of rules ISO 9001:2000. The evaluation of the actual quality was realized by applying the methodology of Goal/Question/Metric (GQM); relying on GQM, ad-hoc questionnaires were written and then submitted to the employees involved in the monitored project, previously described. The quality model selected as reference in order to accomplish the evaluation was the ISO 9126; more precisely, the features considered and thus evaluated were those reported in [6]: functionality, usability, efficiency, reliability, and portability.

One Goal was associated to each model's characteristic directly connected with the costs. A set of Questions was defined for each Goal and reported within a questionnaire. An overall amount of 30 questions was defined for the evaluation of GENESIS' features and 13 questions for the evaluation of the method.

For each question one answer among four predefined answers could be chosen; an integer value corresponded to each answer, varying in the discrete interval (0,1,2,3), together with a weight associated to each question; in this way a metric was defined, aiming at evaluate the quality characteristics. A mapping function was established, reported in table 1; it linked the metrics' values computed on the basis of the given answers in the questionnaires and an ordinal scale, used for evaluating the characteristics.

Table 1. Mapping between metric values and characteristic values

Measured Value	Characteristic Value
$0.000 \leq X < 0.800$	Low
$0.800 \leq X < 1.550$	Medium
$1.550 \leq X < 2.300$	High
$2.300 \leq X \leq 3.000$	Very high

The actual values obtained for each evaluated characteristic both for GENESIS and for the proposed method are reported in the Table 2. These values are compared with the expected ones; the gap between the expected values and the actual ones is indicated. The gap null supports the conjecture of costs reduction. As it can be noticed from the Table 2, only in two cases the actual values differ, in pejorative manner, from the expected one and in both cases for the tool GENESIS. This is due mainly to two reasons. The tool was in a prototypal version; this affected the usability. The people held greater expectations with regard to the support the tool was supposed to provide; this affected the feature of functionality.

Table 2. Comparison between the expected values and the actual ones.

Element	Characteristic	Actual Value	Mapping	Expected value	Gap
Genesis	Functionality	1,277	Medium	High	-1
Genesis	Usability	1,442	Medium	High	-1
Genesis	Efficiency	2,035	High	High	0
Genesis	Reliability	2,059	High	High	0
Genesis	Portability	1,904	High	High	0
Method	Functionality	1,811	High	High	0
Method	Usability	1,958	High	High	0
Method	Efficiency	1,972	High	High	0
Method	Reliability	2,166	High	High	0
Method	Portability	2,055	High	High	0

The last the research question Q_3 is *'The used methods and tools ease the institutionalization of Quality System: they allow the diffusion of the knowledge about the process within the overall enterprise, and the reduction of the costs for executing the processes'*. In order to verify the fulfillment of this goal some post-experience interviews have been conducted with the top management. They pointed out a strong commitment of the people involved in the process, due to a great awareness and consciousness of tasks' execution procedures. The approach fosters the optimization of procedures and organizational resources in order to meet process' business goals. This leaded to a major reduction of effort dispersion and consequently to a reduction of costs.

6 Conclusions

SMEs encounter specific problems when dealing with both management and definition of productive processes quality, modeling and implementation; the main hurdles regard identification and representation of the elements processes consist of and the relationships between them in order to monitor, control and act properly, accordingly with the rising needs. This paper presented a lightweight approach consisting of different phases, indicating the interventions to be accomplished within an organization in order to identify the processes; it was described also an open source software supporting the workflow management.

The proposed method was applied to an enterprise context, the Elisys S.r.l. , an Italian Small Industrial Enterprise located in the district of Caserta and working in the field of electronic systems' and associated software design and production. The accomplished experience carried out very encouraging outcomes; they showed that the agility of the method presented allowed to effectively establishing the SME's Quality System.

In particular, the following preliminary conclusions about the effectiveness of the approach can be drawn:

1. The processes actually performed are strictly compliant with those formalized. As matter of fact, the employees involved in the process were facilitated to follow rigorously the activity flow implemented in the GENESIS environment, due to the process control of the WfMS.
2. Thanks to the agility of the method, the cost for defining the Quality System was acceptable with regards to the expectations of the Enterprise's top management. More precisely, the expended effort was 3 months/man; 2 months/man were necessary for the implementation of the method and the development of the prototype in GENESIS. The rest of the time was spent in conducting interviews and making observations about the process model.
3. According to top management's considerations, the institutionalization of the process was successfully achieved, mainly due to the agility of the method. Firstly, the overall lifecycle time of the process resulted reduced and consequently the costs were decreased. Secondly, the knowledge of the process was widely spread among the people in charge in the process: indeed they were correctly acquainted with all their responsibilities.

Further observations, not directly connected with the focus of the study, emerged from the case study. The first one regards the use of Vision 2000; compared to the past editions of standard for Quality it seems to be more appealing for the small and medium enterprises. Due to its lower complexity, it results less expensive to be implemented. The second one concerns the GENESIS environment, that resulted easy to be adapted to different contexts without impacting its effectiveness.

As future work we aim at investigating further applications of the method, possibly in different contexts, headed to enforce the evidence of our conjecture: the lightweight approach's effectiveness in defining SME'S Quality System.

Acknowledgments. We would like to thank Emiliano De Gregorio, Master of Science in Computer Engineering, for his valuable contribution in the definition of the proposed approach and its application in Elisys S.r.l.

References

1. Aversano L., CimitileA, De Lucia A., Stefanucci S., Villani M. L., (2003). Workflow Management in the GENESIS Environment., *2nd Workshop on Cooperative Supportfor Distributed Software Engineering Processes,* Benevento, Italy, 2003.
2. Bandinelli S., Fuggetta A., and Ghezzi C. (1993). Software Process Evolution in the SPADE environment, *IEEE Transactions Software Engineering,* vol. 19, n0.12, pag 1-144.
3. Blaha R., Premerlani E., and Lorensen (1991). *Object-Oriented model and design.* Englewoof Cliffs, Prentice-Hall.
4. Conradi R., Orjord E., Westby P. H., and Liu C. (1991). Initial Software Process Management in EPOS. *Software Engineering Journal,* 6(5):275-284.

5. Curtis B., Kellner M.I., and Over J. (1992). Process Modeling. *Communications of the ACM,* 35(9), 75-90.
6. De Gregorio E. (2003). Modellazione di Business Process e Valutazione della Qualità di uno strumento Sofwtare a Supporto. Laurea Thesis, University of Naples Federico II, Dept. Of Computer Science and Systems.
7. Hammer M., & Champy J. (1993). Reengineering the Corporation: a Manifesto for Business Revolution, NY: HarperCollins.
8. Hartmann P., Studt R., and Wewers T. (2001). A Framework for Classifying Interorganizational Workflow-Controlled Business Processes Focusing on Quality Management. *Porc. Of the 34th Hawaii International Conference on System Sciences.*
9. International Standard: **ISO 9000:2000** "Quality Management Systems - Fundamentals and vocabulary", *28 novembre 2001.*
10. Jarvinen J., Hamann D., and van Solingen R. (1999). On Integrating Assessment and Measurement: Towards Continuous Assessment of Software Engineering Processes. *Proc. of the 6th IEEE International Symposium on Software Metrics.*
11. Jacobson I., Ericsson M., & Jacobson A. (1995). The Object Advantage: Business Process Reengineering with Object Technology. ACM Press, Addison-Wesley, 1995.
12. Machado C.F., de Oliveira L.C., and Fernandes R.A. (1999). Experience Report - Restructure of Processes based on ISO/IEC 12207 and SW-CMM in CELEPAR. *Proc. of the 4th IEEE International Symposium and Forum on Software Engineering Standards.*
13. Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber (1993). "Capability Maturity Model for Software, Version 1.1", Software Engineering Institute, CMU/SEI-93-TR-24, DTIC Number ADA263403, February 1993.
14. Miller G. (2001). Sizing up Today's Lightweight Software Processes. *IT Porfessional,* May 2001, IEEE Computer Society, Los Alamitos, CA, 46-49.
15. Oracle Workflow, web site http://www.oracle.com/appsnet/products/procurement/collateral/ds_workflow.html
16. Pan D., (1997). Modeling Software Process, web site http://sern.ucalgary.ca/courses/seng/621/W97/pand/essay.html;
17. Ritrovato P. and Gaeta M., (2002). Generalised Environment for Process Management in Co-operative Software. *Proceedings of the 1st Workshop on Cooperative Supports for Distributed Software Engineering Processes,* Oxford, UK, IEEE Computer Society Press, 2002.
18. Rombach H.D. and Verlage M., (1995). Directions in Software Process reasearch. Advances in Computer, Academic Press, (41): 1-63.
19. Staffware plc, web site http://www.staffware.com/
20. Ultimus, web site http://www.ultimus.com/
21. Victor R. Basili, (1992). Software modeling and measurement: The Goal/Question/Metric paradigm. *Technical Report CS-TR-2956,* Department of Computer Science, University of Maryland, College Park, MD 20742, September 1992.
22. Van Solingen R., Berbhout E., (1999). *The Goal/Question/Metric Method.* McGraw-Hill.
23. Zettel J. (2000), *Spearmint Multi-View Architecture,*Technical Report Fraunhofer Institute of Software Engineering, Kaiserslautern, Germany, IESE-Report No. 013.99/E, Version 1.0.

Definition and Empirical Validation of Metrics for Software Process Models

Félix García, Francisco Ruiz, and Mario Piattini

Alarcos Research Group, University of Castilla-La Mancha, Paseo de la Universidad, 4
13071 Ciudad Real, Spain
{Felix.Garcia, Francisco.RuizG, Mario.Piattini}@uclm.es,
alarcos.inf-cr.uclm.es/english/

Abstract. Software companies are becoming more and more concerned about software process improvement, when they are promoting the improvement of the final products. One of the main reason of the growing interest in software metrics has been the perception that software metrics are necessary for software process improvement. Measurement is essential for understanding, defining, managing and controlling the software development and maintenance processes and it is not possible to characterize the various aspects of development in a quantitative way without having a deep understanding of software development activities and their relationships. In this paper a representative set of metrics for software process models is presented in order to evaluate the influence of the software process models complexity in their quality. These metrics are focused on the main elements included in a model of software processes, and may provide the quantitative base necessary to evaluate the changes in the software processes in companies with high maturity levels. To demonstrate the practical utility of the metrics proposed at model level, an experiment has been achieved which has allowed us to obtain some conclusions about the influence of the metrics proposed on two sub-characteristics of the maintainability: understandability and modifiability, which besides confirm the results of a subjective experiment previously performed.

1 Introduction

There is a direct correlation between the quality of the process and the quality of the resulting software products and for this reason companies are becoming more and more concerned about software process improvement, when they are promoting the improvement of the final products. To support the software process evaluation and improvement, a great variety of initiatives have arisen establishing reference frameworks. Among these initiatives, of special note are CMM [17], CMMI [18] and the ISO 15504 [11]. Among the above-mentioned improvement initiatives, CMMI (Capability Maturity Model Integration) stands out as being especially important. Within the context of CMMI, the company should continuously understand, control and improve its processes, in order to reach the aims of each level of maturity. As a result

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 146–158, 2004.

of effective and efficient processes, a company will, in return, receive high quality products that satisfy both the needs of the client and of the company itself.

One of the main reason of the growing interest in software metrics has been the perception that software metrics are necessary for software process improvement [7]. Measurement is essential for understanding, defining, managing and controlling the software development and maintenance processes and it is not possible to characterize the various aspects of development in a quantitative way without having a deep understanding of software development activities and their relationships [15].

Therefore, in order to provide the a quantitative basis for the improvement of software process it is necessary the measurement of the process, but before it is necessary to know the elements involved. For this reason the issue of process modelling has received growing attention in the software community during last years. The process model is seen as the starting point to analyse, improve and enact the process, but the need of strict coupling between process modelling and process measurement has not yet clearly emerge [13]. We have treated this issue in previous works [8;9].

In this paper a representative set of metrics for software process models is presented in order to evaluate the influence of the complexity in the software process models in their maintainability. These metrics are focused on the main elements included in a model of software processes, and may provide the quantitative base necessary to evaluate the changes in the software processes in companies with high maturity levels.

To demonstrate the practical utility of the metrics proposed at model level an experiment has been carried out which has allowed us to obtain some conclusions about the influence of the metrics proposed on two sub-characteristics of the maintainability: understandability and modifiability.

Firstly, we present a set of representative metrics for the evaluation of software process models and an example of calculation. In Section 3 the empirical validation of the metrics proposed at model level is presented. Finally, some conclusions and further works are outlined.

2 Proposal of Metrics for Software Process Models

Research on the software process evaluation has been focused in the collection of project data to obtain throughput, efficiency and productivity metrics and for this reason explicit metrics on software process models have not been defined.

The study of the possible influence of the complexity of software process models in their execution (enactment) could be very useful. For this reason the first step is the definition of a collection of metrics in order to characterise the software process model. The main objective to be achieved is the development of a empirical study to demonstrate the influence of the metrics proposed (which are applied on the attributes of software process models) on the maintainability of software process models. These metrics are indicators of a software process model structural complexity, and they could be every useful, taking into account that a software process model with high degree of complexity will be much more difficult to change, and this affects to their

maintainability. This affirmation is based on the theoretical basis for developing quantitative models relating to structural properties and external quality attributes provided by [4] which is illustrated in Figure 1. This basis could be applied to the software process in the same way they are applied to software artifacts. Software process models hardly maintainable affect to the execution of projects (more expensive in resources and schedule) and to the final quality of the software products obtained.

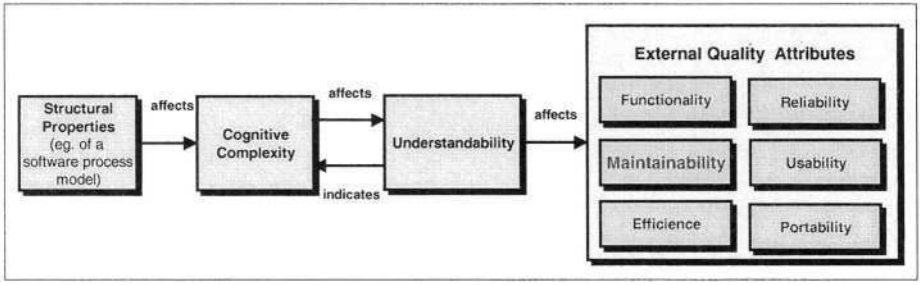


Fig. 1. Relationship between structural properties and external quality attributes

The metrics have been defined following the SPEM terminology [19], but they can be directly applied to other process modeling languages. The conceptual model of SPEM is based on the idea that a software development process consists of the collaboration between abstract and active entities, referred to as process roles, that carry out operations, called activities, on tangible entities, called work products. The basis of software processes consists of the interaction or collaboration of multiple roles through the exchange of work products and triggering the execution , or enactment of certain activities. The aim of a process is to bring a set of work products to a well-defined state.

SPEM has not a graphical notation by itself, but basic UML diagrams can be used to present different perspectives of a software process model. In particular, the following UML notations are useful: Class diagram, Package diagram, Activity diagram, Use case diagram and Sequence diagram. Figure 2 shows an example of a simplified software process model which belongs to the Rational Unified Process [12]. For the graphical representation of the model the Activity Diagram notation and the stereotypes which represent the SPEM constructors has been used.

As we can observe in Figure2, using UML Activity diagrams it is possible to represent a view of the software process in which the different activities, their precedence relationships, the work products consumed or produced and the responsible roles are included.

The metrics have been defined examining the key software process constructors of the SPEM metamodel and they could be classified as:

- **Model Level Metrics.** They are applied in order to measure the structural complexity of the overall software process model. These metrics are repre sented in Table 1 and an example of calculation of the metrics for the software process model represented in the Figure 2 is shown in Table 2.

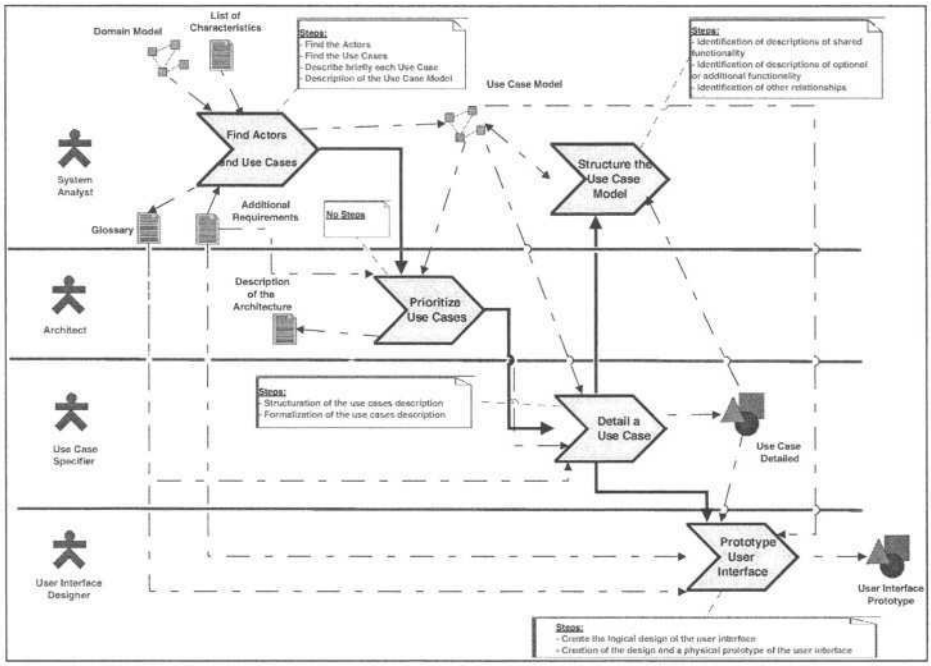


Fig. 2. Example of a Software Process Model represented with SPEM

- **Fundamental Element Metrics**. They evaluate the structural complexity of the main elements of the software process model: Activity, Work Product and Process Role. The definition of these metrics is presented in [9].

3 Empirical Validation of the Model Level Metrics

In order to prove the practical utility of the metrics it is necessary to run out empirical studies. In this section we describe an experiment we have carried out to empirically validate the proposed measures as early maintainability indicators. We have followed some suggestions provided in [20][14] [5] and [6] on how to perform controlled experiments and have used (with only minor changes) the format proposed in [20] to describe it.

We performed a previous controlled experiment [10], pursuing a similar objective. In it, as in this one, the independent variable is the software process model structural complexity. In the previous experiment the dependent variables were three maintainability sub-characteristics (understandability, analysability and modifiability) measured by means of user ratings on a scale composed of seven linguistic labels (from extremely easy to extremely difficult for each sub-characteristic). Even though the results obtained in the previous experiment reflect that several of the model level

Table 1. Model Level Metrics

Metric	Definition
NA(PM)	Number of **Activities** of the software process model
NWP(PM)	Number of **Work Products** of the software process model
NPR(PM)	Number of **Roles** which participate in the process
NDWPIn(PM)	Number of input dependences of the **Work Products** with the **Activities** in the process
NDWPOut(PM)	Number of output dependences of the **Work Products** with the **Activities** in the process
NDWP(PM)	Number of dependences between **Work Products** and **Activities** $NDWP(PM) = NDWPIn(MP) + NDWPOut(MP)$
NDA(PM)	Number of precedence dependences between **Activities**
NCA(PM)	Activity Coupling in the process model. $NCA(PM) = \frac{NA(PM)}{NDA(PM)}$
RDWPIn(PM)	Ratio between **input dependences** of Work Products with Activities and **total number of dependences** of Work Products with Activities $RDWPIn(PM) = \frac{NDWPIn(PM)}{NDWP(PM)}$
RDWPOut(PM)	Ratio between **output dependences** of Work Products with Activities and **total number of dependences** of Work Products with Activities $RDWPOut(PM) = \frac{NDWPOut(PM)}{NDWP(PM)}$
RWPA(PM)	Ratio of **Work Products** and **Activities**. Average of the work products and the activities of the process model. $RWPA(PM) = \frac{NWP(PM)}{NA(PM)}$
RRPA(PM)	Ratio of **Process Roles** and **Activities** $RRPA(MP) = \frac{NRP(MP)}{NA(MP)}$

Table 2. Values of Model Level Metrics

Metric	Value	Metric	Value
NA(PM)	5	NDA(PM)	4
NWP(PM)	8	NCA(PM)	5/4= 1,25
NPR(PM)	4	RDWPIn(PM)	13/18=0,722
NDWPIn(PM)	13	RDWPOut(PM)	5/18=0,278
NDWPOut(PM)	5	RWPA(PM)	8/5=1,6
NDWP(PM)	18	RRPA(PM)	4/5= 0,8

metrics (NA, NWP, NDWPIn, NDWPOut, NDWP y NDA) were highly related to-software process models maintainability, we are aware that the way we choose to measure the dependent variable was subjective and relies solely on judgment of the users, which may have biased the results. Therefore, we decided to carry out another experiment measuring the dependent variable in a more objective way. This experiment is presented in the following subsections.

3.1 Definition

Using the GQM template [1], for goal definition, the experiment goal is defined as follows:

Analyse	*Software process models (SPM) structural complexity metrics*
For the purpose of	*Evaluating*
With respect to	*their capability of being used as software process model maintainability indicators*
From the point of view of	*Software Process Analysts*
In the context of	*Software engineers of a company for the development and maintenance of information systems.*

3.2 Planning

After the definition of the experiments -*why* the experiment is conducted-, the planning took place. The planning prepares for *how* the experiment is conducted, including the following activities:

- **Context selection.** The context of the experiment is a group of professionals of a software company, and hence the experiment is run on-line (in an industrial software development environment). The subjects have been thirty-one software engineers of the Cronos Iberica Consulting, company dedicated to the development and maintenance of software for information systems. The experiment is specific since it focuses on SPM structural complexity metrics. The ability to generalise from this specific context is further elaborated below when we discuss threats to the external validity of the experiment. The experiment addresses a real problem, i.e., which indicators can be used to assess the maintainability of SPM? To this end it investigates the correlation between metrics and maintainability.
- **Selection of subjects.** The subjects have been chosen for convenience, i.e., the subjects are professionals of a software company who have wide experience and knowledge in software product modelling (UML, databases, etc.), but they have not experience or knowledge in the conceptual modelling of SPM.
- **Variables selection.** The independent variable is the SPM structural complexity. The dependent variable is SPM maintainability.
- **Instrumentation.** The objects have been 18 SPM belonging to different standards and methodologies. The independent variable has been measured through the metrics proposed at process model level (see section 2). The dependent vari-

ables have been measured by the time the subjects spent answering the questions of the first section related with the understandability of each model (understandability time) and by the time subjects spent carrying out the tasks required in the second section of the experiment (modifiability time). Our assumption here is that, the faster a class diagram can be understood and modified, the easier it is to maintain.

- **Hypothesis formulation.** We wish to test the following two set of hypotheses:
  1) Null hypothesis, H_{0e}: There is no significant correlation between structural complexity metrics (NA, NWP, NPR, NDA, NDWP, NDWPIn, NDWPOut, NCA, RDWPIn, RDWPOut, RWPA, RRPA) and the understandability time.
  2) Alternative hypothesis, H_{1e}: There is significant correlation between structural complexity metrics and the understandability time.
  3) Null hypothesis, H_{0m}: There is no significant correlation between structural complexity metrics and the modifiability time.
  4) Alternative hypothesis, H_{1m}: There is significant correlation between structural complexity metrics and the modifiability time.
- **Experiment design.** We selected a within-subject design experiment, i.e., all the tests (experimental tasks) have had to be solved by each of the subjects. The subjects were given the tests in different order.

Table 3. Metric values for each class diagram.

Model	NA	NWP	NPR	NDWPIn	NDWPOut	NDWP	NDA	NCA	RDWPIn	RDWPOut	RWPA	RRPA
1	6	6	3	5	6	11	6	1,000	0,455	0,545	1,000	0,500
2	5	6	4	5	5	10	4	1,250	0,500	0,500	1,200	0,800
3	2	13	2	12	3	15	1	2,000	0,800	0,200	6,500	1,000
4	9	25	9	25	21	46	11	0,818	0,543	0,457	2,778	1,000
5	5	6	4	5	5	10	8	0,625	0,500	0,500	1,200	0,800
6	4	11	4	14	9	23	3	1,333	0,609	0,391	2,750	1,000
7	8	17	1	15	11	26	9	0,889	0,577	0,423	2,125	0,125
8	5	8	4	13	5	18	4	1,250	0,722	0,278	1,600	0,800
9	7	12	1	12	11	23	6	1,167	0,522	0,478	1,714	0,143
10	24	37	10	72	40	112	24	1,000	0,643	0,357	1,542	0,417
11	7	12	5	12	11	23	6	1,167	0,522	0,478	1,714	0,714
12	2	8	3	6	4	10	1	2,000	0,600	0,400	4,000	1,500
13	3	6	1	8	3	11	4	0,750	0,727	0,273	2,000	0,333
14	3	5	7	5	3	8	2	1,500	0,625	0,375	1,667	2,333
15	4	9	1	9	7	16	6	0,667	0,563	0,438	2,250	0,250
16	8	6	4	9	9	18	7	1,143	0,500	0,500	0,750	0,500
17	4	24	1	20	11	31	3	1,333	0,645	0,355	6,000	0,250
18	5	21	3	21	11	32	4	1,250	0,656	0,344	4,200	0,600

3.3 Operation

It is in this phase where measurements are collected, including the following activities:

– **Preparation.** Subjects were given an intensive training session before the experiment took place. However, the subjects were not aware of what aspects we intended to study. Neither were they aware of the actual hypothesis stated. We prepared the material we handed to the subjects, consisting of eighteen SPM and

one example solved. These models were related with different universes of discourse but they were general enough to be understood by the subjects. The structural complexity of each diagram is different, because as table 3 shows, the values of the metrics are different for each model.

Each diagram had an enclosed test (see appendix A) that included two sections: the first composed by five questions related with the model and the second composed by different steps to perform for the modification of the model. Each subject had to answer the questions of the section 1 and perform the modifications specified in the section 2. For each section the subject had to specify the starting and ending times. The difference between the two times in the first section is that we call understandability time (expressed in minutes and seconds) and the time difference in the second section is called the modifiability time. The modifications to each SPM were similar, including adding and deleting of activities, work products, roles and their dependences.

- **Execution.** The subjects were given all the materials described in the previous paragraph. We explained how to do the tests. We allowed one week to carry out the experiment, i.e., each subject had to do the test alone, and could use unlimited time to solve it. We collected all the data including the times of understanding and modification, the answers of the first section and the original models modified as a result of the second section.
- **Data Validation.** Once the data was collected, we have controlled if the tests were complete and if the modifications had been done correctly. We have discarded the tests of two subjects because they were incomplete. Therefore, we have taken into account the responses of 29 subjects.

Table 4. Spearman´s correlation coefficients between metrics and understandability and modifiability time

Metric	Spearman´correlation coefficients understandability time	Spearman´correlation coefficients modifiability time
NA(PM)	**0,604** p=0,008	0,171 p=0,496
NWP(PM)	**0,694** P=0,001	0,364 p=0,138
NPR(PM)	0,211 p= 0,402	0,348 p=0,157
NDWPIn(PM)	**0,740** p=0,000	0,383 p=0,117
NDWPOut(PM)	**0,747** p=0,000	0,212 p=0,398
NDWP(PM)	**0,772** p=0,000	0,338 p=0,170
NDA(PM)	**0,529** p=0,024	0,060 p=0,814
NCA(PM)	-0,275 p=0,269	0,151 p=0,549
RDWPIn(PM)	0,142 p=0,573	0,324 p=0,190
RDWPOut(PM)	-0,142 p=0,573	-0,324 p=0,190
RWPA(PM)	0,150 p=0,554	0,117 p=0,644
RRPA(PM)	-0,304 p=0,220	0,101 p=0,691

3.4 Analysis and Interpretation

We had the metric values calculated for each SPM (see table 3), and we have calculated the mean of the understandability and modifiability time. So this is the data we want to analyse to test the hypotheses stated above. We have applied the Kolmogorov-Smirnov test to ascertain if the distribution of the data collected was normal. As the data have been non-normal we have decided to use a non-parametric test like Spearman's correlation coefficient, with a level of significance $\alpha = 0.05$, correlating each of the metrics separately with understandability time and with modifiability time (see table 4).

For a sample size of 18 (mean values for each diagram) and $\alpha = 0.05$, the Spearman cutoff for accepting H_{0e} and H_{0m} is 0,4684 [21]. Because the computed Spearman's correlation coefficients for the understanding time (see table 4) for the metrics NA, NWP, NDWPIn, NDWPOut, NDWP and NDA are above the cutoff, and the p-value < 0,05, the null hypothesis H_{0e}, is rejected. Hence, we can conclude that there is a significant correlation between these metrics and the understandability time. Respect to modifiability time all the correlation values are below the cutoff and for this reason there is not correlation with the metrics defined. We think these results are produced due to that the requested modifications in the different diagrams were similar and the subjects had previously answered the questions related with the understandability. So in future experiments we have to focus specially on the influence of the metrics on the modifiability time.

3.5 Validity Evaluation

We will discuss the various issues that threaten the validity of the empirical study and how we attempted to alleviate them:

- **Threats to conclusion validity.** The only issue that could affect the statistical validity of this study is the size of the sample data (522 values, 18 models and 29 subjects), that perhaps are not enough for both parametric and non-parametric statistic test [3]. We are aware of this, so we will consider the results of the experiment as preliminary findings.
- **Threats to Construct Validity.** The dependent variables we used are understandability and modifiability time, so we consider these variables constructively valid.
- **Threats to Internal Validity.** Seeing the results of the experiment we can conclude that empirical evidence of the existing relationship between the independent and the dependent variables exists. We have tackled different aspects that could threaten the internal validity of the study, such as: differences among subjects, knowledge of the universe of discourse among SPM, precision in the time values, learning effects, fatigue effects, persistence effects and subject motivation. The only issue which could affect internal validity was the fatigue effects because the average duration of the experiment was two hours and twenty-four

minutes. But in this experiment we think this does not affect because subjects are professionals. In future empirical studies we have to consider to plan experiments with more reduced duration if the subjects are students.

- **Threats to External Validity.** Three threats to external validity have been identified which could limit the realism of the experiment [16] and the ability to generalize the research results to the population under study:
  - Materials and tasks used. In the experiment, we have used software process models based on standards and methodologies found in the bibliography and tasks representative of real cases, but more empirical studies, using real software process models from software companies, must be carried out.
  - Subjects. The experiment has been performed by professional subjects which eases the generalization of the results.
  - Environment. The experiment was performed in the company but the tasks had to be done by using pen and paper. In future experiments we could consider the use of software tools to perform the activities required in order to provide a more realistic environment.

4 Conclusions and Future Work

In this work a set of representative metrics have been proposed in order to evaluate the influence of the complexity in the software process models in their quality. These metrics are focused on the main elements included in a model of software processes, and may provide the quantitative base necessary to evaluate the changes in the software processes in companies with high maturity levels.

In order to evaluate the relationship between the structural complexity of the software process models, measured through the metrics proposed, and their maintainability we have carried out an experiment to select the metrics related with the time necessary to understand and modify a software process model. This experiment has allowed us to obtain some conclusions about the influence of the metrics proposed at model level in the maintainability of the software process models through two of its sub-characteristics (understandability and modifiability). As a result of these experiments performed we could conclude that the metrics NA, NWP, NDWPIn, NDWPOut, NDWP and NDA are good understandability indicators, however we cannot say the same about the metrics NPR, NCA, RDWPIn, RDWPOut y RWPA. These results confirm part of the results of one previous subjective experiment [10]. However with this experiment we have not demonstrated the relationship between the metrics and the modifiability. We have to conduct new experiments related with the modification of the models to confirm or discard these results.

Although the results obtained in these experiments are good respect to the understanding, we can not consider them like definitive results. It is necessary elaborate new experiments centered in the evaluation of concrete metrics we consider relevant and that in this experiment seem not to be correlated like the metric NCA and NPR. According to the issues previously identified, the lines for improvement for future studies, we can point out the following:

- Development of replicas [2] to confirm the results obtained. In this replicas we could also consider the perception of the participants about the understandability and modifiability of the process models in order to detect possible relationships among this perception (measured in a subjective way), the metrics, and the times employed to understand and modify he models.
- Carrying out a new experiment centered in the evaluation of concrete metrics we consider relevant (NRP, NCA) and that according the previous experiments performed not to seem be correlated with the maintainability of software process models.
- Carrying out case studies using real software process models of companies.
- Consideration of other views related with the modelling of software processes, like for example roles and their responsibilities on work products, in order to define and validate new possible metrics.

Acknowledgements. This work has been partially funded by the TAMANSI project financed by "Consejería de Ciencia y Tecnología, Junta de Comunidades de Castilla-La Mancha" of Spain (project reference PBC-02-001) and by the MAS project partially supported by "Dirección General de Investigación of the Ministerio de Ciencia y Tecnología" (TIC 2003-02737-C02-02).

References

1. Basili, V. and Rombach H. The TAME project: towards improvement-oriented software environments. IEEE Transactions on Software Engineering, 14(6), (1988), 728-738.
2. Basili, V., Shull, F. and Lanubile, F. Building Knowledge through Families of Experiments. IEEE Transactions on Software Engineering, 25(4), (1999), 435-437.
3. Briand, L., El Emam, K. and Morasca, S. Theoretical and empirical validation of software product measures. Technical Report ISERN-95-03, International Software Engineering Research Network (1995).
4. Briand., L., Wüst, J. and Lounis, H. A. Comprehensive Investigation of Quality Factors in Object-Oriented Designs: an Industrial Case Study. In Technical Report ISERN-98-29, International Software Engineering Research Network (1998).
5. Briand L., Arisholm S., Counsell F., Houdek F. and Thévenod-Fosse P. Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of the Art and Future Directions. Empirical Software Engineering, 4(4), (1999), 387-404.
6. Calero, C., Piattini, M. and Genero, M.: Empirical Validation of referential metrics. Information Software and Technology". Special Issue on Controlled Experiments in Software Technology. Vol.43, N° 15 (2001).
7. Fenton, N.: Metrics for Software Process Improvement. Software Process Improvement: Metrics, Measurement and Process Modelling. Haug, M., Olsen, E. W. and Bergman, L (eds). Springer (2001), 34-55.
8. García, F., Ruiz, F. and Piattini, M. Metamodeling and Measurement for the Software Process Improvement. Proceedings of ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'03). Tunis (Tunisia). 14-18 July (2003).

9. García, F., Ruiz, F., Cruz, J.A., Piattini, M. Integrated Measurement for the Evaluation and Improvement of Software Processes. 9th European Workshop on Software Process Technology (EWSPT'9). Lecture Notes in Computer Science. Helsinki (Finland), 1-2 September (2003).
10. García, F., Ruiz, F. and Piattini, M. Proposal of Metrics for Software Process Models. Accepted for publication in Software Measurement European Forum 2004, Rome, 28-30 January, 2004.
11. ISO/IEC: ISO IEC 15504 TR2:1998, part 2: A reference model for processes and process capability, (1998).
12. Jacobson, I, Booch, G. and Rumbaugh, J. The Unified Software Development Process. Addison Wesley (1999).
13. Morisio, M.: Measurement Processes are Software Too. Journal of Systems and Software, vol 49(1), December (1999).
14. Perry, D., Porte, A. and Votta, L. Empirical Studies os Software En-gineering: A Roadmap. Future of Software Engineering. Ed:Anthony Finkelstein, ACM, (2000), 345-355.
15. Pfleeger, S.L.: Integrating Process and Measurement. In Software Measurement. A. Melton (ed). London. International Thomson Computer Press (1996) 53-74
16. Sjoberg, D., Anda, B., Arisholm, E., Dyba, T., Jorgensen, M., Karahasanovic, A., Koren, E. and Vokác, M. Conducting Realistic Experiments in Software Engineering. Proceedings of the 2002 International Symposium on Empirical Software Engineering (ISESE'02).
17. Software Engineering Institute (SEI). The Capability Maturity Model: Guidelines for Improving the Software Process, (1995). In http://www.sei.cmu.edu/cmm/cmm.html
18. Software Engineering Institute (SEI). Capability Maturity Model Integration (CMMI^{SM}), version 1.1. March (2002). In http://www.sei.cmu/cmmi/cmmi.html
19. Software Process Engineering Metamodel Specification; adopted specification, version 1.0. Object Management Group. November (2002). Available in http://cgi.omg.org/cgi-bin/doc?ptc/02-05-03.
20. Wohlin C., Runeson P., Höst M., Ohlson M., Regnell B. and Wesslén A. Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers. (2000).
21. http://department.obg.cuhk.edu.hk/ResearchSupport/Minimum_correlation.asp

Appendix A

SPM 7. With the SPM shown (Figure 2), you have to perform the following tasks:

Tasks: Part I. Answer the following questions:
Write down the starting hour (indicating hh:mm:ss): _____
1.- Can the Use Case Specifier participate in **Structure the Use Case Model?** __
2.- Does **Structure the Use Case Model** precede to **Prototype User Interface?**__
3.- Is it necessary to use the product *Use Case Detailed* like input of the activity **Structure the Use Case Model?** __
4.- Is the work product *Use Case Model* output of **Prioritize Use Cases?**
5.- When **Prototype the User Interface** is executed, has the *Use Case Detailed* product already produced? __

Write down the ending hour (indicating hh:mm:ss): _____

Tasks: Part II. Carry out the necessary modifications to satisfy the following requirements:

Write down the starting hour (indicating hh:mm:ss): _____

1.- The activity **Detail a Use Case** uses the *Description of the Architecture* like input.

2.- The activity **Detail a Use Case** is considered not to precede to **Structure the Use Case Model** and this last one will be precede by **Prototype the User Interface.**

3.- After the activity **Find Actors and Use Cases** is desired to perform a **Checking of the consistence Requisites-Domain,** which receives like inputs the *Domain Model,* the *Use Case Model* and an *Historical of the Domain.* This new activity precedes to **Prioritize Use Cases.**

4.- The Verification Group is the responsible for **Checking of the consistence Requisites-Domain.**

Write down the ending hour (indicating hh:mm:ss): _____

Multiview Framework for Goal Oriented Measurement Plan Design

Pasquale Ardimento, Maria Teresa Baldassarre, Danilo Caivano, and Giuseppe Visaggio

Dipartimento di Informatica – Research Center of Software Technology (RCOST) – Università di Bari - Via Orabona, 4, 70126 Bari – Italy
{ardimento, baldassarre, caivano, visaggio}@di.uniba.it

Abstract. The need for systematic evaluation of process quality and of the resulting products has always been an issue of primary importance within the software engineering community. In the past few years many factors have determined changes in the software process scenario that inevitably impact on software quality. To this end, goal oriented measurement models, such as Goal Question Metrics (GQM), have become essential for assessing desired quality and for enacting software process improvement (SPI). Although the importance of measurement is a shared issue, many software organizations encounter difficulties and strive to define and adopt measurement plans successfully. Causes are most likely attributable to critical issues such as measurement plan dimensions, complexity, dependencies among goals. Often the industrial projects are characterized by GQM plans with numerous quality factors and, consequently, numerous goals. This makes both measurement and interpretation procedures quite onerous. Furthermore, managing a measurement plan turns out to be quite costly and requires numerous investments. To this end, this work proposes a GQM-based approach (Multiview Framework) that provides support in designing a structured measurement plan in order to overcome the common problems mentioned previously, and manage large industrial measurement plans. The proposed approach has been validated through a post mortem analysis, consisting in a legacy data study, carried out on industrial project data.

1 Introduction

In the past few years many factors have determined changes in the software process scenario. Among these: the need for developing software accordingly to cost and time constraints keeping quality at the same level; the criticality of software systems which requires constantly increasing reliability; technological innovation which addresses maintainable and scalable software. Such aspects inevitably impact on the quality of software, on its costs and on the ability of managing software projects. Furthermore, they point out the need for continuously monitoring and improving software production processes in order to fulfill the fixed quality goals.

Quality is intended as the set of desired characteristics that a software process or system should have in order to satisfy the requirements. It is well known that software quality is highly correlated with the quality of development and maintenance processes. In particular, effective process control and improvement implies effective process measurement. According to [9] “you cannot control what you cannot

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 159–173,2004.

measure". So, measurement becomes an essential aspect for software process improvement (SPI). Although the importance of measuring is a shared issue, many software organizations encounter difficulties and strive to define and adopt measurement plans successfully. The reasons are often related to the fact that measurement plans in an industrial setting tend to be very large and include numerous goals and metrics; with a high complexity of interpretation, because many metric values must be controlled and analyzed for each goal; and high dependencies between goals. This increases interpretation complexity and makes it more difficult to identify aimed improvement actions. So, although literature suggests to reduce number of goals and gradually introduce the measurement plan [3] [28], in practice, this seldom occurs. For example, when an enterprise intends obtaining an ISO or CMM certification, often a GQM plan is specified to define the goals in accordance to the quality standards to be measured [10, 14]. In such cases it is unlikely to remain within a 4-5 goal limit.

Given these important issues, this work proposes a GQM-based approach (Multiview Framework) that provides guidelines for designing a measurement plan in order to overcome well known common problems and manage large industrial measurement plans. The proposed approach has been validated through a post mortem analysis carried out on industry wide project data.

The remaining part of the paper is organized in the following way: Open Issues, discusses some important issues of goal oriented measurement. Proposed Approach describes the guidelines for applying the Multiview Framework. In Validation, the approach is applied to a measurement plan, and results are discussed. Finally, conclusions are drawn.

2 Open Issues

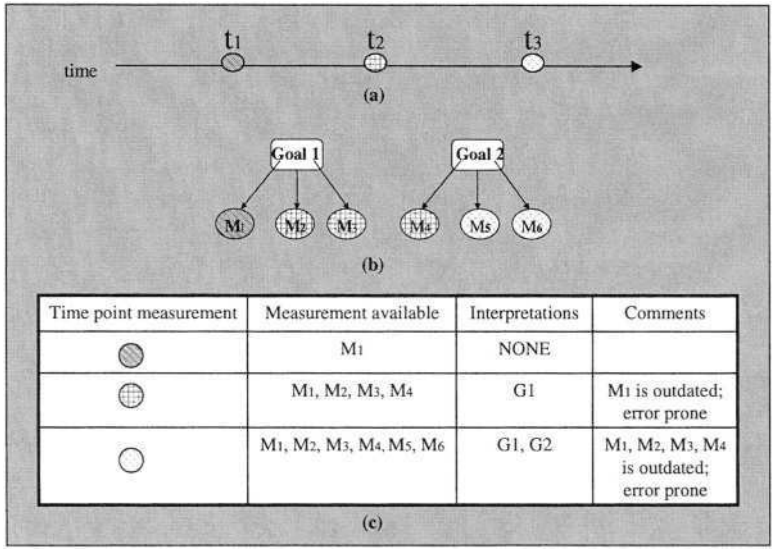
Software product and process quality assessment are issues of primary importance for any software enterprise. In general, software engineers are of the opinion that software measurement should be goal oriented because it adapts to business and project needs. One well known approach to goal oriented measurement plan definition is the Goal Question Metrics (GQM) [1].

The main idea behind GQM is that measurement should be goal-oriented and based on context characterization. It uses a top-down approach to define metrics and a bottom-up approach for analysis and interpretation of measurement data. Quality goals reflect the business strategy and GQM is used to identify and refine goals based on the characteristics of software processes, products and quality perspectives of interest. Furthermore, it provides a general paradigm for defining a measurement plan. A proposed threshold for the number of goals to include in a measurement plan is 4-5 goals [3]. In that paper, the authors suggest that "goals should not cluster more than one purpose, quality focus, or viewpoint" in order to avoid confusion. In other words, a large measurement plan is easier to manage if the measurement goals are specific, clearly defined and structured. Also, when applying a GQM paradigm to define measurement goals within an organization, it is recommended to start with a small number of goals, gain experience and confidence with measurement strategies and then extend the measurement plan.

Unfortunately, in real projects the number of goals aren't manageable nor determined by the measurer. Rather, they depend from the project characteristics and from the business goals that must be achieved. So, it is often necessary to have models with many goals. Also, the evaluation of the measures must be done to guarantee traceability between metrics and business goals. Therefore, in many cases it is not possible to gradually include the goals in the measurement plan. In spite of evidence of successful application of goal oriented measurement programs in industrial contexts such as NASA [5], Motorola [8], HP[12], AT&T [7], Schlumberger RPS [25] there are still many open issues: dimensions of a measurement plan; complexity of interpretations; dependencies between goals; time points for measurement activities.

- Dimensions of a measurement plan depend from the project being executed in an organization. Moreover, in order to correctly evaluate aspects such as its processes, products, resources, economical issues and so on, many goals and therefore metrics may be necessary. A representative example is given by integrating large-grain quantifications such as SW-CMM [19], with fine-grain evaluations provided by GQM. This inevitably leads to a measurement plan with a high number of goals and metrics [14]. On the contrary, a measurement plan with a limited number of goals in industrial field risks to be based on few generic goals, each including too many quality factors [10];
- Complexity and dependency issues are tightly related. Complexity of a measurement plan refers to the number of decisions that must be made to interpret the goal. Decisions depend from the metrics involved in the interpretation model of each goal. Therefore the more the metrics, the more the decisions and the higher the complexity. If the goals of a measurement plan are not appropriately structured, they risk to be dependent from one another. Two goals are dependent if one goal must rely on the interpretation results of the other goal in order to be interpreted. Consequently, complexity of interpretation increases in that a higher number of decisions must be made.
- Another important issue are the time points in which measurement activities occur. More precisely, any measurement plan will most likely include goals having metrics that are collected at different time points (daily, monthly, every trimester etc). So at each time point, only a specific number of metrics can be collected and only part of the goals of the entire measurement plan can be interpreted. Furthermore, if the goals aren't appropriately defined and structured, they may include metrics that are collected in different moments. This means that interpretation is more error prone because part of the measures involved for interpretation have become obsolete by the time all the other metrics needed are also collected. An example is shown in Figure1. In figure1.a., a time line is defined with three time points representing the moments in which measurement occurs; figure 1.b. shows two goals defined without considering when measurement occurs. For this reason they include metrics related to different moments. This means, as represented in figure1.c., that after time t1 no interpretation can be made. Only after t2, G1 can be interpreted but some of the measurements (M1) are outdated. This inevitably impacts the reliability of interpretation, because in that time interval process performances or product characteristics may have changed. The ideal situation is to have goals that include concurrent measures, that are carried out at the same time point.

So, although GQM "represents a systematic approach for tailoring and integrating goals to models of the software processes, products and quality perspectives of interest, based upon the specific needs of the project and the organization" [1], it doesn't provide guidelines for structuring the goals and organizing metrics in order to manage large, complex measurement plans with many goals and measurements that are not necessarily concurrent.



Time point measurement	Measurement available	Interpretations	Comments
	M1	NONE	
	M1, M2, M3, M4	G1	M1 is outdated; error prone
	M1, M2, M3, M4, M5, M6	G1, G2	M1, M2, M3, M4 is outdated; error prone

(c)

Fig. 1. Measurement time points

Literature provides many examples of improvements, extensions and integrations made to the original definition of the GQM approach [2,4,6,13,15,16,17,18,23,24] as a consequence to gained experience in industrial settings. Nevertheless, they do not investigate the previously mentioned issues. In the next sections the authors present the approach (Multiview Framwork) and point out how it faces and overcomes such problems. In this sense, the main contribution is to provide an approach for designing a measurement plan so that each goal includes metrics that are measurable in a specific time point; each goal refers to a specific aspect that must be measured; goals are independent. Furthermore, it focuses on models whose number of goals depends from industrial projects, without being bonded either to the heuristic 4-5 goals rule nor to the need to reduce the number of metrics for each goal.

3 Proposed Approach

"A prerequisite for any measurement plan is a deep understanding of processes and products to be measured" [4]. In this sense the Multiview Framework (MF) characterizes the software project being evaluated and provides a model for defining a GQM according to the project definition. The model designs the goals by defining them according to what is being measured. In this way it is possible to select and

carry out measures on a subset of goals that make up the entire measurement plan without having to execute it entirely. In the following the 4-step approach is described.

Step 1: PROJECT DEFINITION

A project is the execution of one or more activities that produce intermediate artifacts and deliverables and lead to a final product. Any project involves one or more processes.

In this first step the project, must be defined.

A project can be seen as a tuple PRJ=(P,D,PM,FI) where:

- P=(p1,p2,...pn) is the set of processes, phases or activities executed to obtain the final products requested, starting from the raw products in input, to reach the fixed targets;
- D=(d1,d2,...dm) the set of deliverables, i.e. input and output products and the artifacts produced during the execution of the processes in use;
- PM: the set of project management activities and related artifacts needed to plan and control the project;
- FI: the set of activities and artifacts needed to evaluate the fitness of investment.

All the processes (P) involved and deliverables (D) produced must be identified. Furthermore, it is necessary to identify the activities involved in planning and controlling the project execution (Project Management-PM) and to evaluate the Fitness of Investment (FI). This is due to the fact that each project has a budget to respect and as a consequence it is important to adopt metrics for measuring and controlling this aspect.

Given the project definition, we define a view as a set of goals related to each of the elements of the project definition included in Process, Product, Project Management or Fitness of Investment. This concept will be detailed in the next step.

Step 2: GOAL SETTING

In this step, each measurement goal is defined using the GQM goal template which consists of five dimensions [1]:

1. *Analyze* the object under measurement (product, process...)
2. *for the purpose of* understanding, controlling or improving the object
3. *with respect to* the quality focus of the object that the measurement focuses on
4. *from the viewpoint of* the people that measure the object
5. *in the context of* the environment in which measurement takes place

Goal setting is done, according to the project PRJ=(P,D,PM,FI) characteristics, and the quality factors to be measured:

a. *Process goals:* aimed at evaluating the quality factor of interest with respect to a process or activity $p_i \in P$;
b. *Product goals:* aimed at evaluating the quality of interest with respect to a deliverable or artifact $d_j \in D$;

c. *Project Management goals:* aimed at evaluating those activities (PM) needed to plan and control the project execution;
d. *Fitness of Investment goals:* evaluate aspects such as cost-benefit ratio and possible risks (FI).

Steps 2a and 2b, are strongly influenced by the process granularity. For example, lets suppose the process granularity is as described in figure2.a. In this case it is necessary to define one or more goals that refer to the entire process P=(p) and one or more goals for each deliverable $d_j \in D=(d_0,d_3)$. If, on the other hand, the process is defined at a lower granularity level as in figure2.b, then it's necessary to define one or more goals for each process $p_i \in P=(p_1,p_2,p_3)$ and one or more goals for each deliverable $d_j \in D=(d_0,d_1,d_2,d_3)$ exchanged between processes. Therefore, the more detailed the processes within the organization are, the more detailed and large the measurement plan will be. If we go into further detail, as in figure2.c., the set of processes include $P=(p_{11}, p_{12}, p_{21}, p_{22}, p_{23}, p_3)$, and the deliverables are $D=(d_{01}, d_{02}, d_{11}, d_{12}, d_2, d_3)$ The number of views increases proportionally to the granularity with which the processes are defined.

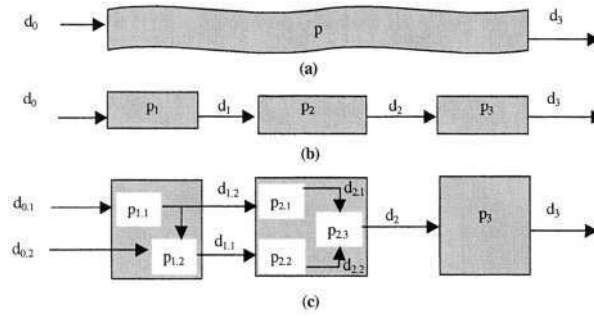


Fig. 2. Different granularity of views

Also, the goals have to be defined keeping in mind the time points when the metrics are to be collected. Therefore goals should include concurrent metrics, collected in the same time point, that relate to the view and quality factor being measured.

Based on what has been mentioned in the proposed approach, it is possible to have the following views within the measurement plan:

- a view for every $p_i \in P$, including all the goals that refer to a specific process p_i. With refer to the template for goal definition, each view includes all the goals in which the *Analyze* section contains a specific $p_i \in P$.
- a view for every $d_j \in D$, including all the goals that refer to a specific process dj. With refer to the template for goal definition, each view includes all the goals in which the *Analyze* section contains a specific $d_j \in D$.
- a view for all the goals related to Project management. With refer to the template for goal definition, the *Analyze* section can contain any process $p_i \in P$ or product $d_j \in D$. What allows to distinguish these goals from the others is the *With respect*

to section in which the quality factors evaluated relate to aspects such as: work progress, control of process execution, management of human or instrumental resources.

- a view for all the goals related to Fitness of Investment. With refer to the template for goal definition, the *Analyze* section can contain any process $p_i \in P$ or product $d_j \in D$. The *With respect to* section evaluates quality factors related to aspects such as: cost-benefit ratio, risks, coherence of execution times, costs of personnel and computer resources, project budget etc.

Note that applying the MF approach doesn't reduce the number of goals that make up the measurement plan. At a first glance size may seem to represent a problem for managing the measurement plan itself, but this is not the case. In fact, the "structured" organization of the measurement plan allows to identify which goals and therefore measures refer to each view (processes, products, project management, fitness of investment). So, the quality manager and GQM team may focus on a subset of goals related to the view they are interested in evaluating according to the organization or project needs in a specific time point.

Also, being the goals divided into views, it is easier to understand the relationship between the entities' attributes and the metrics used to measure them. This allows to identify the most appropriate persons for each measurement activity. The measurers clearly see the connection between the measures to collect and the problems they allow to overcome, understand the need for metrics, are motivated in providing accurate data and avoid hiding shortcomings. Such a measurement plan takes into account the set of success factors presented and discussed in [11] and [20] from both an anecdotal and empirical point of view.

Step 3: CROSS-VALIDATION

The measurement plan structure is validated by tracing the goals in a Goal-View cross reference table. It is defined by placing the goals on the rows, and the views on the columns (table1). So, for example, each "X" entry in an (i,j) cell of the table means that the i-th goal has metrics that relate to the j-th view.

Table 1. Goal-view cross reference table

	PROCESS			PRODUCT				PROJ. MANAG	FITNESS OF INV
	p_1	p_2	p_3	d_0	d_1	d_2	d_3		
G_1	X				X				
G_2		X							
G_3					X				
G_4								X	X
G_5			X				X		

The best resulting matrix is a diagonal one. In fact this means that there is one goal for each *object of study* forecasted by the model's granularity.

This means that each of the goals focus on the quality of a single object of study included in the project model, independently from the other objects in the process

itself. The aim of the table is to point out dependencies between goals. An objective evaluation of such measures is given by the following two indicators:

#Dependencies:
Points out the dependencies. Dependencies within the model are present when a goal takes into account more than one view, because it considers multiple processes and/or products, business or management issues measured at different time points. It is calculated with the following formula:

$$\#\,Dependencies = \sum_{i=1}^{k} (Ni - 1)$$

where k=total number of rows of the cross-reference table;
N_i = number of X in the cross-reference table, with refer to the i-th row;

The ideal condition is a dependence of, or very close to, zero. This means that each goal considers only one view and that the resulting table is a diagonal matrix. A high dependence implies a high complexity in that, on one end, the interpretation must consider such dependencies, on the other, a change introduced on a quality factor of a goal impacts on all the other quality factors and views that depend from the goal itself.

Density of Dependencies
Gives and idea of how strong the dependencies are. It is calculated with the following formula:

$$Density\,of\,Dependencies = \frac{\#\,Dependencies}{(\#\,columns * \#\,rows) - \#\,goals}$$

Considering that, in an ideal situation, each goal should consider one view, the resulting value of this indicator depends from the number of extra "X" for each goal in the cross-reference table. The indicator increases when the number of "X" for each goal increases. The fixed threshold for the indicator depends from the organization adopting the metric plan for accessing its processes. In other words, if the organization requires a very accurate quality model, a low threshold such as 0.1 should be assessed. In other cases, where the aims are less restrictive the fixed threshold may be higher, for example, 0.3. To this moment these values have been decided heuristically and have not yet been empirically validated.

According to the values of the two indicators, there are guidelines related to if and how to continue to structure the measurement plan. They consist of a goal analysis first, and then of a view analysis. The analyses involve the rows and columns of the cross reference table respectively. These guidelines are described as follows:

Goal Review
After calculating the two indicators it is necessary to consider if it is the case to redefine some of the goals. In particular, one of the following cases may occur:

1. there are no dependencies, therefore the density of dependencies is zero or not significant. So, measurement plan doesn't need to be improved.

2. there are dependencies and the density of dependencies is higher than the fixed threshold. In this case the structure of those goals that consider more than one view must be restructured. In other words, those goals that have more than one "X" per row in the cross reference table. Such goals must be restructured so that they consider only one viewpoint. This means that the questions and the metrics of the original goal must be adequately distributed among the new goals. The new structure of the metric plan can then be represented in a new cross reference table which will have a greater number of rows.

View Review

The previous step involves a row analysis which results in a set of goals, each considering one view. Nevertheless there may be numerous goals that evaluate the same view. Therefore, it is necessary to analyze the structure of the measurement plan according to the views. One of the following cases may occur for each view:

1. the goals of a view consider the same quality factors. In this case it is advisable to define only one goal for that view;
2. the goal related to a view includes many quality factors. In this case it is advisable to divide the goal in multiple ones, for each quality factor considered.

Once the view analysis has been carried out, the goals that make up the metrics plan can then be represented in the cross reference table.

Step 4: VERIFICATION OF INTERPRETATION COMPLEXITY

The interpretation of the measures is a critical issue. In this approach, decision tables [21, 22] are used to represent the interpretation of a goal.

A decision table is defined for each goal of the measurement plan. It can be seen as a tabular representation of a decision situation, where the state of a number of conditions determines the execution of a set of actions. Not just any representation, however, but one in which all distinct situations are shown as columns in a table, such that every possible case is included in only one column. A description of how decision tables are used for the interpretation is presented in figure2.

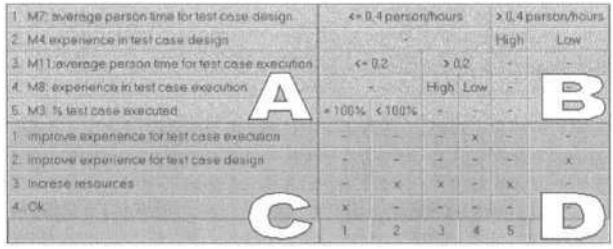


Fig. 3. Decision table

A decision table is composed of four parts: condition stub contains the metrics used for the interpreting the goal (A); the condition entries correspond to the metric baselines (B); the action stub is the set of improvement actions that must be applied if the metrics do not reach the fixed baseline values (C); the action entries are the result

of the combination of condition entries of the metrics involved and the possible actions (D). We convey in defining each action entry as a rule. Moreover, every table column of the decision table, from here on called *rule,* indicates which actions should (or should not) be executed for a specific combination of condition entries [21].

The complexity of the interpretation is equal to the total number of rules, or action entries that make up the decision table. The structure of the measurement plan resulting from the MF approach, consists of many specific goals each related to a single view, having a limited number of metrics, possibly homogeneous, concurrent and scarcely dependent. This reduces the overall complexity of the interpretation. Thanks to the decision tables, the interpretation results more comprehensible in that it clearly points out how metrics, condition entries and improvement actions are linked [27]. In other words, it is easier to understand the relationship between the attributes and the metrics used to measure them. Ideally the approach assures that each decision table interprets one goal, and in this way, avoids dependencies.

More precisely, decision tables point out dependencies among goal interpretations. If the condition stub of a goal, lets say G_i, not only includes metrics but also other goals, such as G_k, necessary for interpretation, then G_i can be interpreted only after G_k. Furthermore, if G_i and G_k have metrics collected at different time points, dependencies may lead to error prone interpretations. For example, suppose that G_i's metrics are collected at time t_2, and it uses G_k for interpretation. If G_k's metrics were collected during a previous time point (t_1), interpretation conclusions would most likely be different than if G_k had been interpreted with metrics collected at time t_2. Consequently, G_i interpretation is inevitably influenced. When decision tables point out dependencies between goals, *Goal Review* must be carried out again to remove or minimize such relations.

4 Validation

In the following, approach validation is presented and results are discussed.

4.1 Legacy Data Method

A first validation of the model has been done through a legacy aged study [26]. It is classified as a historical method because it collects data from projects that have already been completed. For this reason, we considered an existing measurement plan, defined within an executed project, and applied the proposed approach to it. The validation was based on analysis of how the structure of the measurement plan would have been if the MF approach presented in this work had been used to design it. Data was then collected and compared among the two measurement plans.

The industrial project considered for the study consisted in reengineering a legacy system in order to migrate the operative environment from monolithic centralized to client-server architecture, and the language from assembler to Cobol and C++. A measurement plan was defined in order to monitor project execution according to the ISO9001 standard. The goals were defined according to the GQM approach, as known in literature. They measure quality factors, such as reliability, efficiency,

maintainability, portability, process quality, project conformance, risks, usability. Being the project finished, a large amount of data was available for analysis.

In order to validate the proposed model, the four steps of the MF were applied to the previous measurement plan.

The original measurement plan will be identified from here on with "NS-GQM" (non structured). The measurement plan resulting from the application of the multiview framework will be referred to as "S-GQM" (structured).

Step1: Process Definition/Identification

In step1 the project (PRJ) was defined. A graphical representation of the high level process used is given in figure 3 and detailed as follows:

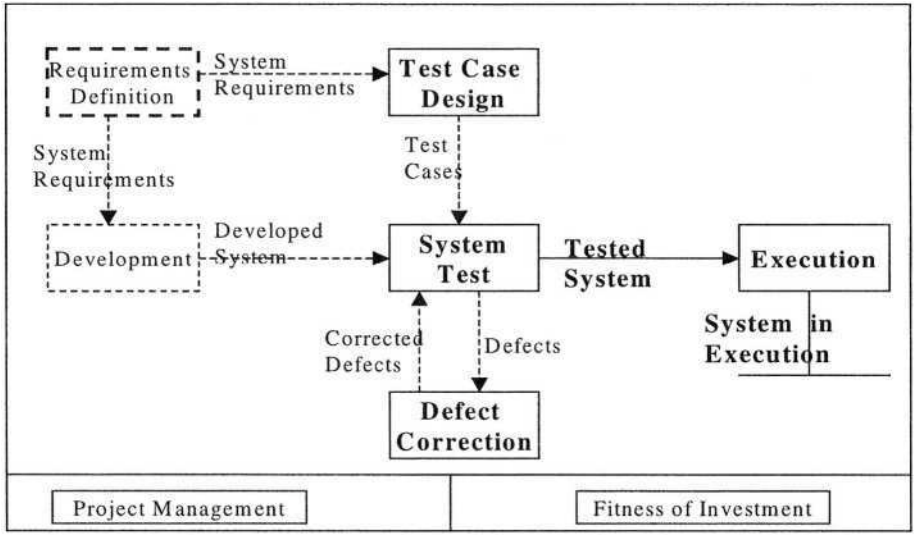


Fig. 4. Graphical representation of the project

PRJ:

- P = {Requirements Definition, Test Case Design, Development, System Test, Execution, Defect Correction};
- D = {System Requirements, Developed System, Test Cases, Corrected System, Defects, Tested System, System in Execution};
- PM = project management;
- FI = fitness of investment;

This step gives an overall view of what is being assessed in the project.

Step2: Goal Setting

Goals were defined for each view by using the GQM template. In this step the goals of the new measurement plan are defined according to the project definition PRJ. Athough figure3 represents the entire project, only part of the processes (Test Case Design, System Test, Defect Correction, Execution) and products (Defects, Tested System, System in Execution) were involved in the monitoring process.

For this reason the processes and products that weren't involved in the monitoring activities are reported in dotted lines. Note that, when we applied the approach and analyzed in detail all of the quality factors and metrics, we assessed that the metrics used to measure the product *Defect* were actually related to *Tested System.* For this reason, in the S-GQM the metrics were included in this view, and the Defect product is no longer present in table3.

Goal definition according to the PRJ characterization assures that goals measure different views and that the metrics chosen are significant for the views being evaluated. Moreover, metrics identified and included also depend from the process granularity and from the time points when measurement activities occur.

Step3: Cross Validation

Cross-validation was carried out. To this end, the comparison between the goal-view cross reference tables of the NS-GQM and the S-GQM before and after applying the approach is interesting. They are presented respectively in table 2 and 3.

Table 2. NS-GQM goal-view cross reference table

	Process				Product			Proj Mng	Fit. Inv
	T.Cse Des	Syst. Test	Def Corr	Exec	Def	Test Syst	Syst Exe.		
G_1		X	X	X	X	X			
G_2								X	
G_3						X	X		
G_4	X	X					X		
G_5						X	X		
G_6		X						X	X
G_7						X		X	X
G_8				X		X			

Table 3. S-GQM goal-view cross reference table

	Process				Product		Proj Mng	Fit Inv
	TCse Des	Syst Test	Def Corr	Exec.	Test Syst	Syst Exe		
$G^{TestCaseDesign}$	X							
$G^{SystemTest}$		X						
$G^{DefectCorrec}$			X					
$G^{Exec,\ 1}$				X				
$G^{Exec,\ 2}$				X				
$G^{TestSyst}$					X			
$G^{SystExec,1}$						X		
$G^{Syst.Exec,\ 2}$						X		
$G^{ProjectManag}$							X	
$G^{FitnessOfInv,\ 1}$								X
$G^{FitnessOfInv,\ 2}$								X

In the first case, since the model had not been applied, the goals considered many views, and therefore inevitably clustered more than one purpose, quality focus, or viewpoint. This effect is visibly identified by the fact that the matrix is sparse, differently from the S-GQM, where the resulting matrix is close to a diagonal one. For example, for measuring the process *System Test* and the product *Tested System,* the NS-GQM required seven goals: three process (G_1, G_4, G_6) and five product (G_1, G_3, G_5, G_7, G_8) goals. In particular G_1 considered both process and product aspects. On the contrary, the S-GQM required only two goals: one process ($G^{SystemTest}$) and one product ($G^{TestSyst}$).

Step4: Verification of Interpretation Complexity

Finally interpretation complexity was verified. In other words, for each goal, decision tables were defined and complexity was calculated as the number of rules. The overall complexity of the S-GQM ranged from 6 to 32 rules. Not only decision tables reduce effort interpreting collected measures, but they allow to identify improvement actions more accurately. This is assured by the structure of the decision table itself, in that it considers all combinations of conditional states, and explicates the relation between attributes and entities' attributes and the measures used to assess them.

After applying the model, we analyzed and compared data related to the NS-GQM and S-GQM. A synthesis of the results obtained is presented in table 4.

Table 4. Comparison of NS-GQM and S-GQM

Data collected	NS-GQM	S-GQM
Nr. Goals	8	11
Nr. Metrics	168	168
Nr. Metrics per goal (min)	21	11
Nr. Metrics per goal (max)	46	34
Nr. Metrics per goal (average)	32,75	20,18
Interpretation Complexity (min)	32	6
Interpretation Complexity (max)	63	32
Interpretation Complexity (average)	44,87	18,45
# Dependencies	24	0
Density of Dependencies	0,27	0

As it can be seen, although the overall number of goals is greater in the S-GQM, the average interpretation complexity is less. This is due to the lower number of metrics for each goal, achieved as a consequence to applying the Multiview Framework to the NS-GQM.

5 Conclusions

This paper proposes a GQM-based approach for managing company wide quality models that require many goals and involve many quality factors: technical, functional, economical, managerial and so on. The approach provides:

a. guidelines for designing a GQM so that each time point involves a limited number of goals to measure and interpret;
b. guidelines for controlling and improving the quality model's complexity;
c. guidelines for controlling and improving interpretation;
d. support for making goal interpretation more comprehensible

The proposed approach has been applied to a goal oriented quality model defined within a real industrial project many years ago. Application of the previous points a), b), and c) through the MF approach have proven the method's efficacy. Point d) has been assessed in a previous study [27] through a controlled experiment. In particular, it has proven that MF requires less effort for interpretation. In other words, less effort for identifying the metrics to measure and interpret, according to the specific process or product being evaluated. At the moment, the empirical data available relates to the usability of the GQM defined with MF in terms of comprehensibility and efficiency [27].

The experimentation and method application carried out to this moment are surely not enough to generalize the proposed approach to wider contexts. For this reason the authors aim in extending and executing further empirical investigation on other industrial projects.

References

[1] Basili V.R., Caldiera G., Rombach H.D.: Goal Question Metric Paradigm. Encyclopedia of Software Engineering, John Wiley & Sons, Vol.1, 1994, pp. 528-532.
[2] Bianchi A, Caivano D., Lanubile F., Rago F., Visaggio G.: Towards Distributed GQM. Proc.7th IEEE Workshop on Empirical Studies of Software Maintenance –WESS01- Florence Italy, 2001
[3] Briand L.C., Differding C.MRombach., H.D.: Practical Guidelines for Measurement-Based Process Improvement. SOFTWARE PROCESS – Improvement and Practice, Vol.2, 1996 pp. 253-280.
[4] Brockers A., Differding C., Threin G.: The role of software process modeling in planning industrial measurement programs. Proc.3rd International Software Metrics Symposium, Berlin, March 1996, pp.31-40
[5] Basili V.R., Green S.: Software Process Evolution at the SEL. IEEE Software, vol.11, no.4, July 1994, pp.58-66.
[6] Briand L.C., Morasca S., Basili V.R.: An Operational Process for Goal-Driven Definition of Measures. IEEE Transactions on Software Engineering, Vol28, No 12, December 2002, pp.1106-1125.
[7] Barnard L., Price A.: Managing Code Inspection Information. IEEE Software, vol.11, no.2, pp.59-69, Mar. 1994.
[8] Daskalantonakis M.K.: A Practical View of Software Measurement and Implementation Experiences within Motorola. IEEE TSE, vol.18, no.11, 1992, pp.998-1010
[9] Demarco T.: Controlling Software Projects. Yourdon Press, New York, 1982
[10] Fuggetta A., Lavazza L., Morasca S., Cinti S., Oldano G., Orazi E.: Applying GQM in an Industrial Software Factory. ACM Transactions on Software Engineering and Methodology, Vol 7, No.4, October 1998, pp.411-488
[11] Gopal A., Krishnan M.S., Mukhopadhyay T., Goldenson D.R.: Measurement Programs in Software Development: Determinants of Success. IEEE TSE, vo.28,no.9,2002, pp.865-875.

[12] Grady R.B.: Practical Software Metrics for Project Management and Process Improvement. Hewlett-Packard Professional Books, 1992
[13] Kilpi T.: Implementing a Software Metrics Program at Nokia. IEEE Software, November-December, 2001, pp.72-77.
[14] Loconsole A.: Measuring the requirements management key process area. Proc. 12th European Software Control and Metrics conference - ESCOM01, London England, April, 2001, pp. 67-76.
[15] Latum F.V.et al.: Adopting GQM-Based Measurement in an Industrial Environment. IEEE Software, January-February 1998, pp. 78-86
[16] Mendonça M.G, Basili V.R.: Validation of an Approach for Improving Existing Measurement Frameworks. IEEE Transactions on Software Engineering, Vol.26, No.6, June 2000, pp. 484-499.
[17] Offen R.J., Jeffrey R.: Establishing Software Measurement Programs. IEEE Software, March-April, 1997, pp.45-53.
[18] Olsson T., Runeson P.: V-GQM: A Feed-Back Approach to Validation of a GQM Study. Proc. of the 7th International Software Metrics Symposium -METRICS 01 – London England, April 4-6, 2001, pp.236 – 245.
[19] Paulk M.C., Curtiss B., Chrissis M.B., Weber C.B.: Capability Maturity Model for Software. Version 1.1., Pittsburg, Software Engineering Institute, 1993.
[20] Pfleeger S.L.: Lessons Learned in Building a Corporate Metrics Program. IEEE Software, May 1993, pp.67-74.
[21] Pooch U.W.,:Translation of Decision Tables. Computing Surveys, vol.6, no.2, June 1974, pp.125-151
[22] available at: http://www.econ.kuleuven.ac.be/tew/academic/infosys/research/prologa/prologa.htm
[23] Solingen R.V., Berghout E.: Improvement by goal-oriented measurement - Bringing the Goal/Question/Metric approach up to Level 5. Proc. European Software Engineering Process Group conference-E-SEPG-, June 16-20, 1997, Amsterdam, The Netherlands.
[24] Solingen R.V., Berghout E.: Integrating Goal-Oriented Measurement in Industrial Software Engineering:Industrial Experiences with and Additions to the Goal/Question/Metric Method. Proc. 7th International Software Metrics Symposium - METRICS01-, pp.246-258
[25] Solingen R.V., Latum F.V., Oivo M., Berghout E.W.: Application of Software Measurement at Schlumberger RPS: towards enhancing GQM. Proc. 6th European Software Control and Metrics Conference – ESCOM95, The Netherlands, May 17-19, 1995.
[26] Zelcowitz M.V., Wallace D.R.: Experimental Models for Validating Technology. IEEE Computer, May 1998, pp.23-31.
[27] Baldassarre M.T., Caivano D., Visaggio G.: Comprehensibility and Efficiency of Multiview Framework for Measurement Plan Design. Proceedings of the International Symposium on Empirical Software Engineering, Rome Italy, October 2003;
[28] Solingen R.V., Berghout E.: The Goal/Question/Metric Method: a practical guide for quality improvement of software development. McGraw Hill International, UK 1999.

Eliminating Over-Confidence in Software Development Effort Estimates

Magne Jørgensen[1] and Kjetil Moløkken[1,2]

[1] Simula Research Laboratory,
P.O.Box 134, 1325 Lysaker, Norway
{magne.jorgensen, kjetilmo}@simula.no
[2] University of Oslo, Norway

Abstract. Previous studies show that software development projects strongly underestimate the uncertainty of their effort estimates. This overconfidence in estimation accuracy may lead to poor project planning and execution. In this paper, we investigate whether the use of estimation error information from previous projects improves the realism of uncertainty assessments. As far as we know, there have been no empirical software studies on this topic before. Nineteen realistically composed estimation teams provided minimum-maximum effort intervals for the same software project. Ten of the teams (Group A) received no instructions about how to complete the uncertainty assessment process. The remaining nine teams (Group B) were instructed to apply a history-based uncertainty assessment process. The main results is that software professionals seem to willing to consider the error of previous effort estimates as relevant information when assessing the minimum effort of a new project, but not so much when assessing the maximum effort!

1 Introduction

Assume that you are a project manager and ask one of the software developers to estimate the development effort for a task. The software developer believes that the task requires about 40 work-hours. You want to plan the project task with a low risk of over-run and therefore ask the developer to assess the minimum and maximum effort for the task. You instruct the developer to be "almost certain" that the actual effort will be included in his/her minimum-maximum intervals. The developer assesses the minimum to be about 35 and the maximum to be about 60 work-hours. How confident should you be in the accuracy of that minimum-maximum effort interval? A good guess, supported by the studies presented in *Section 2,* is that the actual probability of including the actual effort in the minimum-maximum interval is only about 60%, i.e., far from the desired confidence level of "almost certain". If you base your contingency buffer on the provided maximum effort there is, therefore, a high risk that your plan is too optimistic.

What should we do about this tendency to provide over-optimistic minimum-maximum effort intervals? This paper describes an attempt to improve the accuracy of the judgment-based minimum-maximum effort intervals through the use of historical uncertainty information. The background for our attempt is described in *Section 3.* The study itself is described in *Section 4,* discussed in *Section 5,* and, concluded in *Section 6.*

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 174–184, 2004.

2 Studies on Over-Confidence

The accuracy of software development minimum-maximum effort intervals has only been studied by researchers in the last few years (the first study was published, as far as we know, in 1997). Nevertheless, there is strong evidence to suggest that there is a systematic tendency towards too narrow minimum-maximum intervals, e.g.:

- Connolly and Dean [1] report that the actual effort used by student programmers to solve programming tasks fell inside their 98% confidence minimum-maximum effort intervals only in about 60% of the cases, i.e., the intervals were much too narrow to reflect 98% confidence. Explicit attention to, and training in, establishing good minimum and maximum effort values increased the proportion inside the intervals to about 70%, which was still far from the required 98%.
- Jørgensen and Teigen [2] conducted an experiment in which 12 software professional were asked to provide 90% confidence minimum-maximum effort intervals on 30 previously completed maintenance tasks. In total, 360 minimum-maximum effort intervals were provided. The software professionals had access to a small experience database for similar projects and were informed about the actual effort of a task after each minimum-maximum effort interval assessment. Although "90% confident", the professionals included, on average, only 64% of the actual effort values on the first 10 tasks (Task 1-10), 70% on the next 10 task (Task 11-20), and, 81% on the last 10 tasks (Task 21-30). In other words, even after 20 tasks with feedback after each task, there was a systematic bias towards intervals that were too narrow.
- Jørgensen, Teigen and Moløkken [3] studied the software development minimum-maximum effort intervals of 195 student project activities and 49 industry project activities. The minimum-maximum effort intervals of the activities of the student projects were based on a 90% confidence level, and included only 62% of the actual effort values. The effort intervals of the activities of the industrial projects were based on a confidence level of being "almost certain", and included only 35% of the actual effort values, i.e., a strong underestimation of uncertainty.
- Jørgensen [4] studied the effort estimation intervals provided by seven realistically composed estimation teams. The teams assessed the uncertainty of the effort estimate for the same two projects, i.e., in total fourteen project minimum-maximum effort intervals. The actual projects had been completed in the same organization as the software professionals participating in the study. Only 43% of the teams' effort prediction intervals included the actual effort.

In addition to these studies, there are results about uncertainty assessment in other domains. Most studies from other domains seem to report levels of overconfidence similar to that in the software domain; see, for example, the studies described in [5-8]. Lichtenstein and Fischhoff [9] report that the level of over-confidence seems to be unaffected by differences in intelligence and expertise, i.e., we should *not* expect the level of over-confidence to be reduced with more experience. Arkes [10] provides a recent overview of studies on over-confidence that strongly supports the overconfidence claim.

3 Background

The focus of this paper is to investigate whether greater explicit use of the distribution of the estimation error for similar projects improves the minimum-maximum effort intervals. The idea behind of our distributional-based approach can be illustrated by the real-life case described in [11]:

A team that was concerned with the development of a high school curriculum on thinking under uncertainty was conducting a planning session. The question was raised of the time that would be required to complete the first version of a textbook. The participants in the discussion were asked to estimate this value as realistically as possible; the seven estimates rangedfrom 18 months to 3 years. The team leader then turned to one of the participants, an educator with considerable expertise in the problems of curriculum development, with the following question: 'What has been the experience of other teams that have tried to write a textbook and develop a curriculum in a new area where no previous course of study existed? How long did it take them to complete a textbook, from a stage comparable to the present state of our project?' The chilling implications of the answer appeared to surprise the expert who gave it, as much as they surprised the other participants: 'Most teams I could think of failed and never completed a textbook. For those who succeeded, completion times have rangedfrom five to nine years, with a median of seven.'

Kahnemann and Tversky [11] comment on this case: *A notable aspect of this anecdote is that the relevant distributional information was not spontaneously used, although it was available to one expert from personal knowledge and could have been estimated quite accurately by several other participants.* Their comment suggests that if we do not remind the estimators to use the historical performance as an indicator of future performance, it may not be used at all.

When viewed in the context of our software estimation uncertainty assessment, the above case suggests that accuracy may be increased when estimators are forced to focus explicitly on the distribution of estimation error of similar projects. Assume, for example, that an estimator is informed that about 50% of the projects similar to the one to be estimated have been subject to effort estimation overruns of more than 30%. This information should, given rational behavior, reduce the probability that the project leader provides strongly over-optimistic maximum values. In other words, it may be harder to remain over-optimistic about the estimation accuracy when possessed of explicit (pessimistic) historical information about previous estimation inaccuracy.

The outcome of explicit use of historical performance is, however, not obvious. There are examples suggesting that people may remain over-optimistic even in the light of historical performance data. The amusing study described in [12] exemplifies resistance to the application of historical data: *Canadians expecting an income-tax refund were asked to predict when they would complete and mail in their tax forms. These respondents had indicated that they typically completed this chore about 2 weeks before the due rate; however, when asked about the current year, they predicted that they would finish, on average, about 1 month in advance of the due date. In fact, only 30% of the respondents were finished by their predicted date - on average they finished, as usual, about 2 weeks before the deadline.* It is possible that the same tendency to neglect the relevance of historical data is present in the context of software development effort estimation uncertainty assessment.

A potential reason for over-optimism, despite the availability of relevant information about historical estimation error performance, is that people tend to apply an "inside view" when planning and estimating work instead of a history-based "outside view" [13]. People typically divide the work into phases and activities (the "inside" of a project) and estimate each of these phases, instead of comparing the current project as a whole (the "outside" of the project) with other projects. As suggested in the high school curriculum case described earlier this section, thinking based on the "inside view" can easily lead to estimates that are too low and views on the uncertainty that are too optimistic, while an "outside view" may increase realism. It is, consequently, not obvious when awareness of previous estimation error on similar projects does improve the realism. A first attempt to study the effect of information about previous estimation errors is described in Section 4.

4 A Study on Use of Estimation Error Distribution

4.1 The Participants

Nineteen estimation teams, all from a medium-large Norwegian company that develops, among other things, web solutions, participated in the study. The company manager in charge of all projects ensured that each team was realistically composed for the estimation work to be performed, i.e., that each team possessed the necessary skill and experience. The participants knew they were part of an experiment, but were instructed to behave as similarly as possible to the way in which they would have done in real estimation work. An important incentive for serious estimation work was that all the teams should present their estimation work in the presence of the other teams and the company's management.

4.2 The Estimation Task

The estimation teams were instructed to estimate the effort of a real project, which was to implement a web-based system for storing and retrieving research studies at our research laboratory (Simula Research Laboratory). Earlier we had received project bids from other web-development companies based on the same twelve-page requirement specification. The lowest bid reflected a use of effort of about 30 work-hours, the highest about 800 work-hours. The high variation in estimates was partly due to the flexibility in how the requirements could be implemented and partly to differences in development performance. Based on previous project bids we expected a high variation in the most likely estimates among the estimation teams.

4.3 The Study Design

The nineteen estimation teams were separated at random into two groups. The Group A teams received the estimation instructions:

a) Estimate the most likely effort of the project, assuming the normal productivity level of your company.
b) Assess the uncertainty of the estimate as % of most likely effort (most likely effort = 100%).

The format of the uncertainty assessment was as follows:
We are "almost sure" that the actual effort is between:
Minimum: _____ % of most likely effort
Maximum: _____ % of most likely effort

The Group B teams received the instructions:

a) Estimate the most likely effort of the project, assuming the normal productivity level of your company.
b) Recall what you believe is the "historical estimation error distribution" of similar projects in your company. Use the table below. *(The table instructed the estimation team to assess the frequency (in %) of projects with: More then 100% effort overrun, 50-100% effort overrun, 25-49% effort overrun, 10-25% effort overrun, +/- 10% effort estimation error, 10-25% too high effort estimates, 26-50% too high effort estimates, and, more than 50% too high effort estimates.)*
c) Assess the uncertainty of the estimate as % of most likely effort (most likely effort = 100%).

The only difference in the instructions was step b) for the Group B teams.

The teams spent between 1 and 1.5 hours on the estimation work, i.e., less than in a normal estimation situation. From a statistical point of view, the effect of spending less than the usual amount of time on estimation should be a widening of the minimum-maximum effort intervals, i.e., that the minimum-maximum intervals would indicate a higher uncertainty in the effort estimates.

The *hypothesis* that we tested in this study is that information about the distribution of estimation error for similar projects leads to minimum-maximum effort intervals that more accurately reflect the actual uncertainty of the effort estimates.

There is no obvious test of the accuracy of individual minimum-maximum effort intervals. "Almost sure" does not imply that *every* minimum-maximum effort interval should include the actual effort. Assume, for example, that the actual effort turns out to be higher than the maximum effort in a project where the estimator was "almost sure" to include the effort in the minimum-maximum effort interval. The estimator may still have provided a proper uncertainty assessment. It may, for example, be the case that the project experienced an unusual amount of bad luck and that the actual effort in 99 out of 100 similar project executions would have fallen inside the interval. In other words, the optimal evaluation of minimum-maximum intervals would be based on a large set of effort minimum-maximum intervals. This type of evaluation is not possible in the current study. Instead, we had to base our evaluation of the minimum-maximum intervals on the assumption that the actual uncertainty of the team's effort estimates is close to or higher[1] than that reflected in the estimation error

[1] Reflecting the limitations regarding time spent on the estimation task.

for previous, similar projects. For example, if among similar projects there had been a 10% frequency of projects with more than 100% effort over-runs, a 90% confidence maximum value should be 200% of the estimated most likely effort. The study [11] described in Section 3, and the results reported in study [14], suggest that the empirical error distribution is, in many cases, a good indicator of future effort estimation uncertainty. Even if the assumption were not true in our experimental context, the results from the study could be useful for analyzing how the uncertainty assessments are impacted by information about previous estimation error, e.g., to analyze differences in how the error distribution information is applied in the minimum and maximum assessment situations.

4.4 The Results

Table 1 shows the estimates of most likely effort, minimum effort, and maximum effort (minimum and maximum in % of most likely effort). Table 2 shows the estimation error distributions provided by the estimation teams in Group B.

The large difference in estimates indicates that the estimation teams interpreted, and intended to provide solutions for, the requirements quite differently. This high variation in interpretation of the requirement specification also means that we should analyze the estimates carefully. In principle, we should treat the data as the estimation of 19 different projects, i.e., we should only compare the minimum-maximum effort values of one team with the same team's distribution of error for similar projects. However, since the variation of effort estimates is similar in Groups A and B, we assume that the "average interpretation" in Group A and Group B is similar, i.e., that it is meaningful to compare the mean minimum and maximum values of the Group A and Group B estimation teams.

We instructed the estimators to be "almost certain" to include the actual effort in their minimum-maximum intervals. It is not clear how to translate the concept of being "almost certain" into a statement of probability. However, we believe it is safe to state that being "almost certain" should correspond to there being at least an 80% probability that the actual effort is included in a minimum-maximum effort interval. We further assume a symmetric error distribution and base our interpretation of "almost certain" on a 10% probability of actual effort higher than the maximum effort and a 10% probability of actual effort lower than the minimum effort. These assumptions are open to discussion, but it makes no great difference to the results when changing the "almost certain" interpretation to, for example, a 5% probability of actual effort higher than the maximum and lower than minimum effort. Table 3 applies the above interpretation of "almost certain" and shows the minimum and maximum effort values we would expect if the estimation teams had based their uncertainty assessments on the historical distribution of estimation errors for similar projects. An example to illustrate the calculations is this. Estimation team 13 assessed that 10% of similar projects had more than 100% effort overrun, (see Table 2.) We therefore calculated the maximum effort to be 200% of the most likely effort.

The wide uncertainty categories of the estimation error distribution, e.g., +/-10% error, means that it is not always obvious which value to choose as a history-based minimum and maximum. Minor deviations between our interpretation of the history-based and the actual minimum and maximum effort values do not, therefore, necessarily reflect little use of the estimation error distributions.

Table 1. Estimated, Minimum and Maximum Effort

Team	Group	Estimate (work-hours)	Minimum (% of Estimate)	Maximum (% of Estimate)
1	A	800	90	150
2	A	413	75	130
3	A	750	50	200
4	A	5200	50	120
5	A	136	66	150
6	A	1543	65	130
7	A	192	90	115
8	A	149	90	150
9	A	310	60	200
10	A	152	70	150
Mean values – A		**965**	**71**	**150**
11	B	484	70	120
12	B	2280	90	160
13	B	2080	100	150
14	B	752	90	200
15	B	456	75	150
16	B	2640	110[2]	150
17	B	800	90	125
18	B	88	90	140
19	B	820	90	120
Mean values – B		**1160**	**88**	**146**

Table 2. Distribution of Estimation Error of Similar Projects

	Teams (Group B only)									
Estimation Error Category	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**Mean value**
>100% overrun	45	18	10	10	10	5	10	0	18	14
50-100% overrun	20	40	35	20	10	5	20	5	25	20
25-49% overrun	15	22	25	30	30	35	40	20	30	27
10-24% overrun	10	15	25	20	30	45	20	40	15	24
+/- 10% of error	7	4	0	5	10	10	10	20	12	10
10-25% too high estimates	3	1	0	10	5	0	0	10	0	3
24-50% too high estimates	0	0	0	0	5	0	0	5	0	1
>50% too high estimates	0	0	0	0	0	0	0	0	0	0

[2] Statistically, a minimum of 110% of most likely effort is meaningless and should lead to an increase of most likely effort. Estimation team 16 defended the minimum value by pointing to the fact that they never had observed any similar project that used less than 110% of the estimated most likely effort. This viewpoint is, to some extent, valid when not separating estimates of most likely effort and planned effort. See the discussion in Jørgensen, M. and D.I.K. Sjøberg, *Impact of effort estimates on software project work.* Information and Software Technology, 2001, **43**(15), p. 939-948.

Table 3. History-Based and Estimated Minimum and Maximum

Teams (Group B only)									
Teams	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
History-Based Minimum	90%	100%	100%	90%	90%	100%	90%	90%	90 %
Estimated Minimum	70%	90%	100%	90%	75%	110%	90%	90%	90 %
Correspondence minimum?	Close	Close	OK	OK	Close	OK	OK	OK	OK
History-Based Maximum	200%	200%	200%	200%	200%	175%	200%	140 %	200 %
Estimated Maximum	120%	160%	150%	200%	150%	150%	125%	140 %	120 %
Correspondence maximum?	No	No	No	OK	No	Close	No	OK	No

Table 3 suggests that all the Group B teams achieved a correspondence between the history-based and their actually estimated *minimum* effort. Comparing the mean minimum effort values of the two groups of estimation teams, see Table 1, shows that the teams in Group B have a minimum that is higher (mean value 88% of most likely effort) than that of the teams in Group A (mean value 71% of most likely effort). The difference in mean values, combined with the results in Table 3, suggests that the minimum values of the estimation teams in Group B were influenced by the error distribution for similar projects. For example, being aware that no similar project ever had been over-estimated by more than 10% (teams 16 and 17) seems to have made it difficult to argue that the minimum effort should be lower than 90% of most likely effort. We interpret this as meaning that the minimum values of the Group B teams were more accurate than those of the Group A teams.

Surprisingly, the same impact from the historical error distribution was not present when estimating the *maximum* effort. Only three of the Group B teams estimated maximum effort values close to the history-based maximum. Comparing the mean maximum effort the two groups of estimation teams shows that the teams in Group B had a mean maximum value similar (mean value of 146%) to that of the teams in Group A (mean value of 150%). In other words, while the estimation teams seemed to have applied the distribution of estimation of similar projects when estimating the minimum effort, only a few of them applied the same information when estimating the maximum effort. We discuss reasons for, and threats to the validity of, these results in Section 5.

5 Discussion of the Results

There are several potential reasons explaining the problems of applying historical error data when providing realistic minimum-maximum effort intervals:

- *Conflicting goals*: A potentially important reason for resistance towards providing sufficiently wide minimum-maximum effort intervals is reported in [3]. One of the software professionals participating in that study stated: *"I feel that if*

I estimate very wide effort minimum-maximum intervals, this indicates a total lack of competence and has no informative value for the project manager. I'd rather have fewer actual values inside the minimum-maximum interval, than providing meaningless, wide effort intervals". In the same study it was evaluated how project managers actually assessed the skill of software developers and it was found that they did, indeed, evaluate those software developers as more skilled who provided narrower intervals and exhibited higher confidence. Interestingly, the evaluation of skill based on the width of the interval persisted even in situations when the managers received the information that the assessments were strongly over-confident. In other words, there seems to be an immediate and assured reward for over-confidence. It follows, therefore, that over-confidence does not necessarily lead to punishment (one may be lucky or the management may choose not evaluate the minimum-maximum intervals). In addition, potential punishment is in most cased delayed until the project is completed. The use of historical data to increasing the accuracy of uncertainty assessments may therefore be hindered by the goal of appearing skilled.

- *The "better-than-average"-bias*: Assume that the estimation teams accepted that previously completed similar projects had an error distribution that suggested a higher maximum value than the one they provided. The estimation teams may nevertheless believe that the estimation error history is not relevant for the uncertainty assessment of their effort estimate. In some cases this may be a valid belief. The organization may, for example, have learned from previous experience how to improve the estimates or reduce the project uncertainty. Another possibility is, however, that the estimation teams are subject to the well-known "I-am-better-than-average"-bias [16], i.e., that most teams believe that their effort estimates are better than those of other teams.
- *The lack of statistical skill:* The minimum-maximum assessment instructions given to the Group B estimation teams did not explicitly state how to apply the distribution of previous estimation error. It is, therefore, possible that some of the teams had problems understanding how to link the estimation error distribution to minimum and maximum effort values. A lack of statistical skill does, however, not explain the difference in the impact of the historical information about error distribution on the assessments of minimum and maximum effort and can only explain a minor portion of the problem.

In this study we have used software professionals, realistically composed estimation teams, and a real-life requirement specification. However, there are at least two important limitations to the validity of the results, both related to the artificiality of the estimation context:

- Unrealistic *time restrictions* mean that we cannot automatically generalize the findings into contexts in which the teams spend much more effort on the estimation tasks. Based on the results in [4], however, we would expect greater knowledge about how to solve a task to lead to more, not less, over-confidence.
- The experimental context may have led to a *greaterfocus on appearing skilled* than in a realistic minimum-maximum assessment context. The estimation teams knew that they would not have to implement the project and may therefore have focused more on appearing skilled when presenting the results to the other teams

and the management. This limitation does, however, not explain the difference in use of the historical distribution of estimation error when deriving the minimum and the maximum effort.

Taking into consideration the potential explanations and major limitations of our study, we believe that our most robust finding is the difference in the use of historical information when providing minimum and maximum effort, i.e., that it seems to be more difficult to accept the relevance of historical estimation performance when assessing maximum (worst case) compared with minimum (best case) use of effort. A potential consequence of that finding is described in Section 6.

6 Conclusion and Further Work

The experiment described in this paper is a first step towards better processes for effort estimation uncertainty. Our results indicate that in the attempt to ensure history-based maximum effort values, it is not sufficient to be aware of the estimation error distribution of similar projects. Based on the identification of potential reasons for not applying the estimation error distribution we recommend that the uncertainty assessment process be extended with the following two elements:

- More detailed instructions on how to apply the distribution of previous estimation error to determine minimum and maximum effort for a given confidence level.
- The presence of an uncertainty assessment process facilitator who ensures that the distribution of previous estimation error is properly applied. The facilitator should be statistically trained in uncertainty distributions and require that deviations from the history-based minimum and maximum are based on sound argumentation.

We intend to introduce and evaluate the extended history-based uncertainty assessment process in a software organization.

References

1. Connolly, T. and D. Dean, *Decomposed versus holistic estimates of effort required for software writing tasks.* Management Science, 1997. **43**(7): p. 1029-1045.
2. Jørgensen, M. and K.H. Teigen. *Uncertainty Intervals versus Interval Uncertainty: An Alternative Method for Eliciting Effort Prediction Intervals in Software Development Projects.* In *International conference on Project Management (ProMAC).* 2002. Singapore: p. 343-352.
3. Jørgensen, M., K.H. Teigen, and K. Moløkken, *Better Sure than Safe? Overconfidence in Judgment Based Software Development Effort Prediction Intervals.* To appear in: Journal of System and Software, 2004.
4. Jørgensen, M., *Top-Down and Bottom-Up Expert Estimation of Software Development Effort, 2004.* **46**(1): p. 3-16.

5. Alpert, M. and H. Raiffa, *A progress report on the training of probability assessors,* in *Judgment under uncertainty: Heuristics and biases,* A. Tversky, Editor. 1982, Cambridge University Press: Cambridge, p. 294-305.
6. Kahnemann, D., P. Slovic, and A. Tversky, *Judgement under uncertainty: Heuristics and biases.* 1982: Cambridge University Press.
7. Tversky, A. and D. Kahneman, *Judgment under uncertainty: Heuristics and biases.* Science, 1974.**185**: p. 1124-1130.
8. Yaniv, I. and D.P. Foster, *Precision and accuracy of judgmental estimation.* Journal of behavioral decision making, 1997. **10**: p. 21-32.
9. Lichtenstein, S. and B. Fischhoff, *Do those who know more also know more about how much they know?* Organizational Behaviour and Human Decision Processes., 1977. **20**(2): p. 159-183.
10. Arkes, H.R., *Overconfidence in judgmental forecasting,* in *Principles of forecasting: A handbook for researchers and practitioners,* J.S. Armstrong, Editor. 2001, Kluwer Academic Publishers: Boston, p. 495-515.
11. Kahneman, D. and A. Tversky, *Variants of uncertainty,* in *Judgment under uncertainty: Heuristics and biases,* D. Kahneman, P. Slovic, and A. Tversky, Editors. 1982, Cambridge University Press: Cambridge, United Kingdom, p. 509-520.
12. Griffin, D. and R. Buehler, *Frequency, probability, and prediction: Easy solutions to cognitive illusions?* Cognitive Psychology, 1999. **38**(1): p. 48-78.
13. Kahneman, D. and D. Lovallo, *Timid choices and bold forecasts: A cognitive perspective on risk taking.* Management Science, 1993. **39**(1): p. 17-31.
14. Jørgensen, M. and D.I.K. Sjøberg, *An effort prediction interval approach based on the empirical distribution of previous estimation accuracy.* Journal of Information and Software Technology, 2003. **45**(3): p. 123-136.
15. Jørgensen, M. and D.I.K. Sjøberg, *Impact of effort estimates on software project work.* Information and Software Technology, 2001. **43**(15): p. 939-948.
16. Klein, W.M. and Z. Kunda, *Exaggerated self-assessments and the preference for controllable risks.* Organizational behavior and human decision processes, 1994. **59**(3): p. 410-427.

Measuring the Object-Oriented Properties in Small Sized C++ Programs – An Empirical Investigation

S. Kanmani[1], V. Rhymend Uthariaraj[2], V. Sankaranarayanan[3], and P. Thambidurai[1]

[1] Department of Computer Science and Engineering
Pondicherry Engineering College, Pondicherry – 605 014, India
n_kanmnai@hotmail.com
[2] Ramanujam Computing Centre
Anna University, Chennai – 605 025, India
[3] Crescent Engineering College, Chennai, India

Abstract. This paper discusses about the results of using OO (Object-Oriented) measures for the small-sized programs in C++. The metric values for the class level Object-Oriented properties: inheritance, coupling and cohesion are computed and compared with the existing study results for medium sized programs. Among the three properties, inheritance is used widely in the programs. The effective use of the three properties for the proper abstraction of the class design is investigated using six hypotheses proposed. The result of the investigation shows that inheritance and cohesion are used well for the design of attribute level abstraction in the classes.

1 Introduction

The (OO) technologies (programming languages, tools, methods and processes) claim improvement in the quality of software product deliverables. A large amount of work in this domain focuses on how the quality of OO artifacts (eg. design and code) can be assessed. One direction in the empirical research involve with the development and evaluation of quality models. The immediate goal of this research is to relate the structural attribute measures (for encapsulation, polymorphism, inheritance, coupling and cohesion) to the quality indicators (such as fault proneness, development effort, test effort, rework effort, reusability and maintainability).

Many measures have been proposed in the literature for OO code and design [2], [6], [8-9], [10-15] & [18], which provide the way to automate the measurement and assess the quality of software. Only empirical studies can give tangible answers for, which of them are really suitable. The existing studies have investigated the impact of these measures on the quality attributes such as fault-proneness [7], productivity or effort or the amount of maintenance modifications [19].

The application domain is usually seen as a major factor determining the usefulness of measures or the effectiveness of technologies. The existing studies were made on the systems developed in the industrial environment or on heavy / medium sized projects. Shift of research effort also required, from defining new measures, to investigate their properties and applications on different domain. This is to understand what these measures are really capturing, whether they are really different and

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 185–202, 2004.

whether they are useful indicators of quality or productivity properties of interest in that domain. In this paper, we apply the OO measures on the small-sized programs (having 500 LOC-Lines of code at the maximum) and study their implications.

The following section introduces about the empirical study in detail. Section 3 discusses the descriptive statistics of the measures and compares with the results of the previous study. Section 4 gives the correlation results obtained by evaluating the measures against the number of attributes/methods with the six hypotheses proposed. Section 5 discusses about the results obtained and section 6 concludes the paper.

2 Empirical Study

The empirical study follows the framework suggested by Briand et al [19] and includes the phases of definition, planning, operation and interpretation as follows.

2.1 Study Definition

The motivation of this study is formulated with four key questions.

1. What is the range of values for each of the metric in small-size OO programs? Do the values justify the familiarity and use of the concepts by the developers?
2. How far the metric values obtained by this study are comparable with the values obtained in other domains?
3. How far the designed classes use Object-Oriented properties – coupling, cohesion and inheritance? (as C++ does not compel the developer to have it so) – Which of them are exploited in the design of proper class abstraction?
4. Does the Object-Oriented measures suit for measuring these lower level users' programs?

The programs considered in this study were developed by a class of (60) UG students in the Department of Computer Science and Engineering at Pondicherry Engineering College. These programs were developed during their study period of the laboratory course on OO programming with C++ for 7 weeks. Each week they were given 3 hours of time to program the given problem and to test. They were given Turbo C++ environment with all standard libraries to develop the programs. The students did not have prior knowledge in OO but were already trained in procedural paradigm (PASCAL and C). The course had been framed such that experiments were offered by introducing the OO features gradually.

2.2 Study Planning and Operation

Many measures have been proposed in the literature to capture the structural properties of OO code and design. They are applied on attributes, methods, classes, subsystems and system. In this study, only the class level measures for the OO

properties coupling, cohesion and inheritance are considered (refer Tables 7,8 and 9 for the list of measures).

For metric collection, we used the static analysis tool named OOMC [20] – Object Oriented Metric Calculator to automatically collect metrics data for each of the classes in the program. It is an automated tool developed in C++ programming language to collect the OO design metrics. It assumes that the given source program is syntactically correct. It locates the following statements and collects the measures.

The class derivation/definition statement
The class attributes declaration statement
The class method signature
The class method definition statement
Friend class definition statement

This tool cannot collect the measures, which need the statements of the method to be parsed. They are computed manually (eg: metrics computed with method invocations).

The metric values computed are compared with the values obtained in the existing study results. The varying values show the varying phenomena used (high or low) in the two systems. This comparison brings out the reasons for the varying values.

A correlation study is carried out to identify the properties (inheritance, coupling, cohesion), which are exploited properly by these programmers in the class abstraction. Six hypotheses proposed are used for this study.

2.3 Study Limitations

This study only examines the programs developed by the students and tested by the teacher during their study of the laboratory course. No attempt is made to control for other variables that may affect the result such as the skill/experience level of the student or the process by which the individuals developed their modules (classes).

3 Metric Results and Comparison

The results of the metric computation are given in this section. The values obtained in this study (referred as System 1) small sized programs (2 to 85 LOC in a class) are compared with the values obtained by the study made previously [7] (referred as System 2). The study in System 2 was carried out at the University of Maryland with PG (Post Graduate) students grouped in to eight teams of three students each. Each team was asked to develop a medium sized (2 to 554 LOC in a class) management information system to support video rental business in C++ using GNU environment and OSF/MOTIF during the implementation.

3.1 Coupling Measures

A good quality software design should obey the principle of low coupling. Coupling is the measure of the strength of association established by a connection from one module to other. Therefore stronger the couplings between modules more difficult to

understand, change and correct these modules. This results in more complex software system. The various mechanisms that constitute coupling are categorized [4] as follows. (Assuming c and c' are the classes having the implemented methods m and m' respectively)

- m and m' share a global data (public attributes etc)
- m references an attribute in c'
- m invokes m'
- m receives pointer to m'
- c is the type of a class c's attribute
- c is the type of a methods(m') parameter or return type
- c is type of a methods' local variable
- class is type of a parameter of a method invoked from within another method

Other variations are found by introducing inheritance, friendship or no relationship between the classes c and c'. The list of validated coupling measures considered in this study is given in Table 8.

Table 1 presents the descriptive statistics for coupling measures in the two systems. For each measure, the maximum, minimum, median, mean values and the standard deviation are given in the respective columns.

Coupling measures are measured only with non-library classes (classes defined only in the program in both the systems). From Table 1 the following observations are made from system 1 results

- RFC and RFC1 are almost equal because of very few indirect method invocations found in the classes.
- Compared to CBO', CBO is high because it also adds inheritance based coupling.
- Similarly in ICP measures Inheritance based coupling is found to be high (IH-ICP > NIH-ICP)
- DAC shows that maximum of 6 classes used in the (class) attribute declaration and 4 are the maximum distinct classes used.

The Class-Attribute coupling and Class-method coupling between the classes related by inheritance is not present in System 1. This shows that most of the attributes and methods in the ancestors are declared in the public section and can be assessed directly without an explicit instantiation. In addition, the method-method coupling between classes related through friendship not found. This shows that friends are declared only for attribute interactions. The rest of the measures carry non-zero values even though quantitatively they are less compared to system 2. However, in both the systems the attribute based coupling measures have relatively equal values. Only the method level coupling measures have poor values in system 1. It is due to the less number of methods defined in the class definitions. Comparatively system 1 uses more inheritance based coupling (AMMIC, DMMEC and IH-ICP).

In summary, among the independent classes attribute level coupling are more than method level coupling. Also with the classes related with friendship relation.

However, in the classes related with inheritance, method coupling is found more than attribute coupling.

Table 1. Descriptive statistics for coupling measures

Metric	System 1 results			System 2 results		
	Maximum& minimum	Mean& median	Standard Deviation	Maximum& minimum	Mean& median	Standard Deviation
ACAIC	0,0	0,0	0	3,0	0.02,0	0.28
ACMIC	0,0	0,0	0	13,0	0.12,0	1.22
AMMIC	10,0	0.96,0	1.66	7,0	0.2,0	1.15
CBO	4,0	1.3,1	1.03	12,0	2.02,2	2.16
CBO'	4,0	0.29,0	0.73	12,0	1.97,2	2.14
DAC	6,0	0.32,0	1.13	10,0	0.86,0	1.57
DAC'	4,0	0.16,0	0.65	6,0	0.59,0	0.96
DCAEC	0,0	0,0	0	3,0	0.03,0	0.28
DCMEC	0,0	0,0	0	13,0	0.12,0	1.22
DMMEC	6,0	0.96,0	1.56	21,0	0.21,0	1.98
FCAEC	1,0	0,0	0.19	2,0	0.07,0	0.32
FCMEC	4,0	0.1,0	0.53	9,0	0.28,0	1.39
FMMEC	0,0	0,0	0	16,0	0.56,0	2.14
IC	4,0	0.5,0	0.76	Not considered in the study		
ICP	10,0	1.11,0	1.72	97,0	7.4,0	15.13
IFCAIC	1,0	0,0	0.19	2,0	0.07,0	0.32
IFCMIC	4,0	0.11,0	0.55	9,0	0.28,0	1.37
IFMMIC	0,0	0,0	0	13,0	0.56,0	2.07
IHICP	10,0	1.04,0	1.73	12,0	0.37,0	1.96
MPC	10,0	1.05,0	1.68	58,0	4.4,0	9.05
NIHICP	2,0	0,0	0.37	97,0	7.1,0	14.99
OCAEC	1,0	0.14,0	.35	27,0	0.73,0	2.74
OCAIC	4,0	0.11,0	0.62	8,0	0.73,0	1.4
OCMEC	1,0	0,0	0.13	95,0	2.4,0	11.6
OCMIC	1,0	0,0	0.13	38,0	2.43,0	5.54
OMMEC	2,0	0,0	0.34	36,0	3.64,0	7.05
OMMIC	3,0	0,0	0.48	56,0	3.64,0	8.37
RFC	13,0	3.71,3	2.50	138,2	19.5,13	20.53
RFC-1	13,0	2.82,2.5	2.09	86,2	16.94,13	13.72

3.2 Cohesion Measures

A good quality software design should obey the principle of high cohesion. Cohesion is defined as a measure of the degree to which the elements of a module belong together. In highly cohesive module, all elements are related to the performance of a single function. Therefore, higher the cohesion of a module easier the module to develop, maintain, reuse and less fault prone. The different mechanisms those constitute cohesion are categorized as follows [5] (assuming m and m' are the two methods of a same class c and a is an attribute defined in the class c)

- m references a
- m invokes m'
- m and m' directly reference an attribute a of class c in common
- m and m' directly or indirectly reference an attribute a of class c in common

Table 2. Descriptive statistics for cohesion measures

Metric	System 1 results			System 2 results		
	Maximum& minimum	Mean& median	Standard Deviation	Maximum& minimum	Mean& median	Standard Deviation
CO	1.0,0	.800,1	.422	1,0	0.36,0.33	0.28
COH	1.0,0	.870,1	.289	0.82,0	0.34,0.31	0.19
ICH	1,0	.11,0	.32	72,0	6.36,2	11.35
LCC	1,0	.90,1	.30	1,0	0.5,0.52	0.38
LCOM1	3,0	.22,0	.66	819,0	62.1,24.5	115.20
LCOM2	3,0	.10,0	.55	818,0	43.1,11	105.80
LCOM3	3,1	1.13,1	.42	40,1	5.2,4	4.8
LCOM4	2,1	1.09,1	.30	40,1	5.1,4	4.9
LCOM5	1.5,0	.135,0	.360	2,0	0.78,0.77	0.28
TCC	1,0	.90,1	.30	1,0	0.4,0.36	0.35

Table 7 shows the list of cohesion measures considered in this study. Table 2 presents the descriptive statistics obtained for cohesion measures in both the systems. The following observations are made from the results of the system 1.

- LCC and TCC have exactly the same value because the indirect connection between the methods is not found at all (i.e. method invocation in more than one level). The median value of LCC and TCC shows that most of the classes are connected.
- The value of ICH is found to be very less because in some cases number of parameters used in the method definition is 1(at the maximum) and in others, method call inside another method of the same class is not present.

- LCOM1, LCOM2 have very poor value because the number of method pairs assessing no common attributes is less.
- Relatively high values (minimum = 1) obtained for LCOM4, LCOM5 because of good number of method pairs assessing common attribute.
- COH and CO have values between 0 and 1. However, CO becomes invalid in some of the classes because the number of methods is not greater than 2. Hence, the denominator of CO computation becomes 0.

Comparing the results of the two systems, system 2 has more number of methods in each class resulting in very high numeric values for each of the metric. But system 1 even though it has very poor number of methods (refer appendix Table 11), they are much cohesive (from the low values of the negative cohesion measures – LCOMs and high median values for the positive cohesion measures CO, LCC and TCC whose minimum and maximum range is 0 to 1).

In summary, less number of methods in the classes and very few numbers of parameter in the method definition resulted in low range of non-zero values. This result in confusion to decide, whether low value (eg. zero) refers good (negative) correlation or poor use of concepts.

3.3 Inheritance Measures

Inheritance is the practice of defining object types that are specializations or extensions of existing object types. By inheritance, generic behaviors are inherited to the objects from the parent type and specific behaviors are added additionally. This encourages code reuse, minimizes duplication of effort and promotes consistency of behavior between similar modules. The following are the possible mechanisms used for the inheritance measures

- Measuring position of the class with respect to the hierarchy
- Measuring the inherited components (methods and attributes)

Other variations are found by applying different types of polymorphism [1] (static, dynamic). Table 9 gives the list of inheritance measures considered in the study.

Table 3 presents the descriptive statistics of each inheritance measure. The following observations are made from system 1 results.

- The median value for the measures NOA, DIT, AID, NOD and NOP (=1) shows that inheritance is used commonly in most of the classes.
- Static polymorphism is not present in the system. In addition, the students do not use the overloading feature.

The inheritance measures show a good range of values in each metric in general, except at method level measures. In comparison with system 2, system 1 uses a good amount of inheritance features in all respect.

Table 3. Descriptive statistics for inheritance related measures

Metric	System 1 results			System 2 results		
	Maximum& minimum	Mean& median	Standard Deviation	Maximum& minimum	Mean& median	Standard Deviation
AID	3,0	0.91,1	0.75	4,0	0.85,0	1.19
CLD	3,0	0.61,0	0.82	2,0	0.08,0	0.33
DIT	3,0	0.93,1	0.78	4,0	0.85,0	1.19
DPA	3,0	0.36,0	0.67	Not considered in the study		
DPD	2,0	.20,0	0.55	Not considered in the study		
NMA	5,0	1.61,2	1.3	41,1	11.38,9	7.89
NMI	10,0	1.73,1	2.22	104,0	6.97,0	16.13
NMO	3,0	.27,0	0.65	10,0	0.61,0	1.566
NOA	4,0	0.89,1	0.95	4,0	0.87,0	1.23
NOC	6,0	0.77,0	1.14	5,0	0.18,0	0.77
NOD	6,0	1,1	1.29	9,0	0.23,0	1.10
NOP	2,0	.77,1	0.57	1,0	0.42,0	0.5
OVO	0,0	0,0	0	Not considered in the study		
SIX	3,0	0.9,0	0.56	0.52,0	0.04,0	0.1
SPA	0,0	0,0	0	Not considered in the study		
SPD	0,0	0,0	0	Not considered in the study		

4 Empirical Evaluation

In order to study the effective use of the OO features in the class abstraction, an investigation into a correlation with the metric values and the number of attributes/methods defined in the class has been carried out with the following six hypotheses (since most of the measures use the attributes / methods as basic elements in their computation).

- H1: The more inheritance in the class, less attributes defined in that class. This hypothesis is made on the assumption that inheritance derives additional attributes required for the class (attributes added through inheritance).
- H2: The more coupling in the class, less attributes defined in that class. This hypothesis is made on the assumption that coupling gets required attributes from outside the class (attributes imported by coupling).
- H3: The more cohesion in the class, more attributes defined in that class. This hypothesis is made on the assumption that defining cohesive classes

(self-dependent) require all attributes defined in the class (cohesion by additional attributes).

- H4: The more inheritance in the class, less methods defined in that class. This hypothesis is made on the assumption that inheritance derives additional methods to the class (method added through inheritance).
- H5: The more coupling in the class, fewer methods defined in that class. This hypothesis is made on the assumption that coupling gets required methods used from outside the class (methods imported by coupling).
- H6: The more cohesion in the class, more methods defined in that class. This hypothesis is made on the assumption that defining cohesive classes (self-dependent) require all methods defined in the class (cohesion by additional attributes).

The metric values serve as the independent variable. The dependent variable used is the number of implemented attributes (NA) and number of implemented methods (NM) in the class for the first and last three hypotheses respectively. The Pearson correlation coefficients for each of the metrics against the attributes and methods are computed. This coefficient is a measure of the degree of linear relationship between the two variables.

The results of the correlation are given in the Tables 4, 5 and 6 with the coefficient value and the significance level. The correlation coefficients, which are in the expected trend, are given in *Italics.* Table 4 shows the correlation results with the cohesion measures. The negative cohesion measures (Lack of cohesion among methods – LCOM) have negative values with attributes and the positive cohesion measures Co, Coh, ICH, LCC and TCC have positive values, which show that cohesion increases in the program with the increase of the attribute declaration. However, the trend changes with the correlation of methods. Only the positive cohesion measures Co, Coh & ICH have expected correlation with high significance.

Table 5 shows the correlation results with the inheritance measures and attributes. Low correlation results found with attributes (in high significance) for all of the measures (except the unrelated method level counts – NMI, NMA, NMO, DPA, DPD & SIX). The hierarchy level measures with negative values (AID, DIT, NOA & NOP) show that increase in inheritance level reduces the number of attribute definition. CLD has a negative value since it is inversely proportional to DIT-hierarchy depth. Same trend follows for the measures CLD & NOP with method correlation, but changes for others. However, the significance level is poor for all the measures with methods.

Table 6 presents the results arrived at the correlation with coupling measures and attributes/methods. Among the measures the attribute level interaction are collected by DAC, DAC', **CAIC (all Class-Attribute Import Coupling), **CMIC (all Class-Method Import Coupling) & IC. In these measures IFCAIC, IC & OCMIC show the negative trend, which mean that coupling reduces the attributes to be defined. The paired measures of IFCAIC and OCMIC – FCAEC & OCMEC have the positive correlation (the classes, which export to other classes, have increase in attribute with increase in number of export relationship to satisfy more number of classes' requirement). Among the measures, which capture method level interaction, AMMIC & DMMEC pair shows a good trend. Others are not showing any influence.

Table 4. Correlation results of cohesion measures

Metric	With attributes		With methods	
	Co-eff	p-value	Co-eff	p-value
CO	***0.241***	0.001	***0.310***	0.000
COH	***0.200***	0.000	***0.159***	0.000
ICH	***0.110***	0.006	***0.258***	0.000
LCC	***0.114***	0.006	0.029	0.488
LCOM1	***-0.095***	0.022	0.100	0.016
LCOM2	***-0.167***	0.000	0.110	0.011
LCOM3	***-0.162***	0.000	0.011	0.785
LCOM4	***-0.135***	0.001	***-0.053***	0.208
LCOM5	***-0.027***	0.512	0.102	0.014
TCC	***0.114***	0.006	0.029	0.488

Table 5. Correlation results of inheritance related measures

Metric	With attributes		With methods	
	Co-eff	p-value	Co-eff	p-value
AID	***-0.199***	0.000	0.075	0.585
CLD	***0.266***	0.000	0.137	0.319
DIT	***-0.193***	0.000	0.096	0.488
DPA	-0.174	0.000	-0.140	0.307
DPD	-0.139	0.000	-0.101	0.461
NMI	-0.194	0.000	0.160	0.243
NMO	-0.035	0.263	-0.029	0.834
NOA	***-0.052***	0.100	0.179	0.191
NOC	0.100	0.000	0.033	0.812
NOD	0.200	0.000	0.087	0.529
NOP	***-0.337***	0.000	-0.033	0.812
SIX	-0.209	0.000	0.169	0.217

Table 6. Correlation results of coupling measures

Metric	With attributes		With methods	
	Co-eff	p-value	Co-eff	P-value
AMMIC	-0.123	0.000	***-0.203***	0.000
CBO	0.461	0.000	0.296	0.000
CBO'	0.341	0.000	***-0.179***	0.000
DAC	0.245	0.000	0.265	0.000
DAC'	0.485	0.000	-0.29	0.369
DMMEC	0.366	0.000	***0.457***	0.000
FCAEC	***0.318***	0.000	-0.400	0.000
FCMEC	***0.223***	0.000	-0.280	0.000
IC	***-0.117***	0.000	0.148	0.000
ICP	***-0.075***	0.018	0.240	0.000
IFCAIC	***-0.028***	0.379	0.499	0.000
IFCMIC	0.035	0.263	0.341	0.000
IHICP	-0.143	0.000	0.217	0.000
MPC	***-0.014***	0.000	0.213	0.000
NIHICP	0.318	0.000	0.100	0.002
OCAEC	-0.059	0.062	***0.318***	0.000
OCAIC	0.491	0.000	***-0.091***	0.004
OCMEC	-0.019	0.537	0.000	1.000
OCMIC	***-0.019***	0.537	0.280	0.000
OMMEC	-0.178	0.000	-0.163	0.000
OMMIC	0.379	0.000	0.039	0.218
RFC	-0.022	0.482	0.526	0.000
RFC-1	0.144	0.000	0.595	0.000

5 Discussion

The data collected by this study shows that except few of the coupling measures all others have non-zero values, which proves that all the mechanisms used in the metric

definition exist in the small-scale programs also. Among the three OO features inheritance, coupling and cohesion – inheritance is more commonly used equivalent to system 2. This shows that the students feel comfortable to design classes with decomposition (inheritance relationship). However, the reasons for the low values in the coupling and cohesion measures are identified as follows.

- Less number of method definitions (improper decomposition of methods or improper class behavior implementation)
  - due to the definition of all functionalities in few methods
  - due to the implementation of the part of the operations of the methods in the main program
- Most of the attribute/method definitions of the classes are in public visibility. Hence, interaction is possible without explicit coupling. (eg. no ancestor class instantiation in descendant, resulting zero values in ACAIC, DCAEC, ACMIC & DCMEC)
- Instantiation of classes in global visibility (some of the objects are available in open assess)
- Cohesion measures focus on method pairs or group of methods assessing common attributes – the applicability of these measures for all of the following types of classes is a question mark
  - classes with no attributes but with multiple methods (utilizes inherited attributes)
  - classes with attributes but no methods (class defined for attribute abstraction–used in the top of the hierarchy)
  - classes with multiple attributes and multiple methods
  - classes with multiple attributes but one method (with all functionalities)
  - classes with no attribute but with one method (utilizes inherited attributes)

Only for the category 3, all of the measures can be employed, but for the rest one or two measures are found. This may also be the reason for the poor values

- As the programs are for 7 different problems, enforcement of a solution method may also be the reason for the poor values.

In the correlation study only few of the measures show a medium correlation with the attributes and methods, which may be due to the above-discussed reasons. However, however, the inheritance and cohesion measures show the expected trend with the number of attributes defined. It reveals that the inheritance and cohesion are properly exploited among the programs in the design of the class level attributes.

6 Conclusion

In this paper, an investigation into the use of Object-Oriented features in the class design of small-sized programs has been carried out. The results arrived at this study

shows that low range of metric values for the OO features result in low correlation. The main reason for the values is found to be the poor design of methods in the class. As the classes are designed by sharing the OO features and as the OO features are non-mutually exclusive, results are not predominant in satisfying all of the six hypotheses. However, inheritance and cohesion are utilized among the programs to properly design the class level attributes. Future work can be carried out with replicating the study with students of varying knowledge levels and varying the programming language. Studies can also be carried out to identify the small set of measures, which could evaluate the program design during the training course of varying levels.

References

1. Benlarbi, S., Melo, W.L. :Polymorphism Measures for early risk prediction, Proceedings of the 21st International Conference on Software Engineering, Los Angeles, USA, (1999) 335-344
2. Bieman, J., Kang, B.: Cohesion and Reuse in an Object-Oriented System. Proc. ACM Symposium Software Reusability (SSR'94), (1995) 259-262
3. Briand, L., Arisholm.E., Counsell, S., Houdek, F., Thevenod-Fosse,P.: Empirical Studies of Object-Oriented Artifacts, Methods and Processes: State of the Art and Future Directions. Empirical Software Engineering, 4(4), (1999) 387-404.
4. Briand, L., Daly, J., Wust, J.: A unified framework for coupling measurement in Object-Oriented systems, .IEEE Transactions on Software Engineering, 25(1), (1999) 91-121
5. Briand, L., Daly, J., Wust, J.: A unified framework for cohesion measurement in Object-Oriented systems, Empirical Software Engineering Journal, 3(1), (1998) 65-117
6. Briand, L., Devanbu, P., Melo, W.: An investigation into coupling measures for C++. Proceedings of ICSE 1997, Boston, USA (1997)
7. Briand, L., Wust, J., Daly, J.W., Porter, D.V.: Exploring the relationships between design measures and software quality. Journal of Systems and Software 51 (2000) 245-273
8. Chidamber.S,, Kemerer.C.: Towards a metric suite for object-oriented design. Proceedings of Conference on Object-Oriented Programming: Systems, Languages and Applications (OOPSLA'91) published in SIGPLAN Notices 26(11) (1991) 197-211
9. Chidamber,S. R., Kemerer, C. F.: A metric suite for Object-Oriented design. IEEE Transactions on Software Engineering 20(6) (1994) 476-493
10. Henderson-Sellers B.: Object-oriented metrics – Measures of complexity. Prentice Hall, New Jersey (1996)
11. Hitz, M., Montezeri, B.: Measuring coupling and cohesion in Object-Oriented systems. Proc. International Symposium on Applied Corporate Computing, Monterrey, Mexico (1995)
12. Lake, A., Cook, C.: Use of factor analysis to develop OOP software complexity metrics. Proceedings of 6th Annual Oregon Workshop on Software Metrics, Silver Falls, Oregon (1994)
13. Lee, Y., Liang, B., Wu, S., Wang, F.: Measuring the coupling and cohesion of an Object-Oriented program based on information flow, Proceedings of International Conference on Software Quality, Maribor, Slovenia (1995)
14. Lorenz, M., Kidd, J.: Object-Oriented software metrics: A practical guide: Prentice Hall Inc.: New Jersey (1994)
15. Li, W., Henry, S.: Object-Oriented metrics that predict maintainability. Journal of Systems and Software, 23(2), (1993), 111-222

16. Ohlsson,M.C., Runeson, P.: Experience from Replicating Empirical Studies on Prediction Models , Eighth IEEE Symposium on Software Metrics, Ottawa, Canada (2002)
17. Tang, Kao, Chen.: An empirical study on Object-Oriented Metrics. Proceedings of Metrics (1999) 242-249
18. Tegarden.D, Sheetz.S, Monarchi.D.: A software complexity model of Object-Oriented systems, Decision Support systems, 13(3-4), (1995) 241-262
19. Briand, L., Wust, J.: Empirical studies of quality models in Object-Oriented systems, Advances in computers, Academic Press (2001).
20. Kanmani.S., Prathibha.V.: Object-Oriented Metric Calculator, Technical report, Pondicherry engineering College, 2001.

Appendix:

Table 7. Cohesion measures

Name	Definition	Source
LCOM1(lack of cohesion in methods)	The number of pairs of methods in the class using no attribute in common	[8] & [9]
LCOM2	LCOM2 is the number of pairs of methods in the class using no attributes in common, minus the number of pairs of methods that do. If this difference is negative LCOM2 is set to zero.	[9]
LCOM3	Consider an undirected graph G, where the vertices are the methods of a class, and there is an edge between two vertices if the corresponding methods use at least an attribute in common. LCOM3 is then defined as the number of connected components of G.	[11]
LCOM4	Like LCOM3 where graph G additionally has an edge between vertices representing methods m and n, if m invokes n or vice versa.	[11]
Co(Connectivity)	Let V be the number of vertices of graph G from measure LCOM4, and E the number of its edges. Then $Co = 2\,\frac{|E_c| - (|V_c| - 1)}{(|V_c| - 1).(|V_c| - 2)}$	[11]
LCOM5	Consider a set of methods $\{M_i\}(i=1,\ldots m)$ accessing a set of attributes $\{A_j\}$ $(j=1,\ldots,)$. Let $\mu(A_j)$be the number of methods, which reference attribute A_j.	[10]

	Then $\text{LCOM5} = \dfrac{\frac{1}{a}\left(\sum_{j=1}^{a}\mu(A_j)\right) - m}{1-m}$	
Coh	A variation on $\text{LCOM5}.\ \text{Coh} = \dfrac{\sum_{j=1}\mu(A_j)}{1-m}$	[5]
TCC(Tight class cohesion)	Besides methods using attributes directly (by referencing them) this measure considers attributes "indirectly" used by a method. Method *m* uses attribute *a* directly, if *m* directly or indirectly invokes a method *m'* which directly uses attribute *a*. Two methods are called *connected* if they directly or indirectly use common attributes. TCC is defined as the percentage of pairs of public methods of the class which are connected, i.e. pairs of methods which directly or indirectly use common attributes	[2]
LCC(Loose class cohesion)	Same as TCC, except that this measure also considers pairs of "indirectly connected" methods. If there are methods $m_1,\ldots m_n$ such that m_i and m_{i+1} are connected for $i=1,\ldots n\text{-}1$ then m_1 and m_n are indirectly connected. Measure LCC is the percentage of pairs of public methods of the class which are directly or indirectly connected	[2]
ICH(Information flow based cohesion)	ICH for a method is defined as the number of invocations of other methods of the same class, weighted by the number of parameters of the invoked method. The ICH of a class is the sum of the ICH values of its methods.	[13]

Table 8. Coupling measures

Name	Definition	Source
CBO(Coupling between object classes)	According to the definition of this measure a class is coupled to another, if methods of one class use methods or attributes of the other or vice versa. CBO for a class is then defined as the number of other classes to which it is coupled. This includes inheritance based coupling (coupling between classes related via inheritance)	[9]

CBO'	Same as CBO, except that inheritance based coupling is not counted.	[8]
RFC	Response set for class. The response set of a class consists of the set M of methods of the class, and the set of methods directly or indirectly invoked by methods in M. In other words the response set is the set of methods that can potentially be executed in response to message received by an object of that class. RFC is the number of methods in the response set of the class.	[8]
RFC1	Same as RFC, except that methods indirectly invoked by methods in M are not included in the response set.	[9]
MPC	Message passing coupling. The number of method invocations in a class.	[15]
DAC	Data abstraction coupling. The number of attributes in a class that have another class as their type	[15]
DAC'	The number of different classes that are used as types of attributes in a class	[15]
ICP	Information flow based coupling. The number of method invocations in a class weighted by the number of parameters of the invoked methods.	[13]
IH-ICP	As ICP, but counts invocations of methods of ancestors of classes (i.e. inheritance – based coupling) only	[13]
NIH-ICP	As ICP, but counts invocations to classes not related through inheritance	[13]
IFCAIC ACAIC OCAIC FCAEC DCAEC OCAEC IFCMIC ACMIC OCMIC FCMEC DCMEC OCMEC IFMMIC AMMIC OMMIC	These coupling measures are counts of interactions between classes. The measures distinguish the relationship between classes (friendship, inheritance, none) different types of interactions and the locus of impact of the interaction. The acronym for the measures indicates what interactions are counted: The first or first two letters indicate the relationship (A: coupling to ancestor classes, D: Descendants, F: Friend classes, IF: Inverse friends (classes that declare a given class c as their friend) O: others, i.e. none of the other relationships). The next two letters indicate the type of interaction: CA: There is a Class-Attribute interaction between classes c and d if c has an attribute of type d. CM: There is a class-Method interaction between classes c and d, if c invokes a method of d, or if a method of	[6]

FMMEC DMMEC OMMEC	class d is passed as parameter (function pointer) to a method of class c. The last two letters indicate the locus of impact: IC: Import coupling, the measure counts for a class c all interactions where c is using another class. EC: Export coupling count interactions where class d is the used class.	
IC	Inheritance coupling. Number of parent classes to which a class is coupled.	

Table 9. Inheritance related measures

Name	Definition	Source
DIT	Depth of inheritance tree. The DIT of a class is the length of the longest path from the class to the root in the inheritance hierarchy.	[8]
AID	Average inheritance depth of a class. AID of a class without any ancestors is zero. For all other classes AID of a class is the average AID of its parent classes increased by one.	[10]
CLD	Class to leaf depth. CLD of a class is the maximum number of levels in the hierarchy that are below the class.	[18]
NOC	Number of children. The number of classes that directly inherit from a given class.	[8]
NOP	Number of parents. The number of classes that a given class directly inherits from.	[14]
NOD	Number of Descendants. The number of classes that directly or indirectly inherit from a class (i.e. its children, grand children and so on)	[18]
NOA	Number of ancestors. The number of classes that a given class directly or indirectly inherits from.	[18]
NMO	Number of methods overridden. The number of methods in a class that override a method inherited from an ancestor class.	[14]
NMI	Number of methods inherited. The number of methods in a class that the class inherits from its ancestors and does not override.	[14]
NMA	Number of methods added. The number of new methods in a class not inherited, not overriding	[14]
SIX	Specialization index. SIX is NMO*DIT/(NMO+NMA+NMI)	[14]
OVO	Overloading in stand-alone classes.	[1]
DPA	Dynamic polymorphism in ancestors	[1]
DPD	Dynamic polymorphism in descendants	[1]
SPA	Static polymorphism in ancestors	[1]
SPD	Static polymorphism in descendants	[1]

Table 10. Size measures

Name	Definition
Stmts	The number of declaration and executable statements in the method of a class
NM (Number of methods)	The number of all methods (inherited, overriding and non-inherited) methods of a class
NAI (Number of attributes)	The number of attributes in a class (excluding inherited ones). Includes attributes of basic types such as strings, integers..
NMpub (Number of public methods)	The number of public methods implemented in a class
NMNpub (Number of non-public methods)	The number of non-public (i.e. protected or private) methods implemented in a class

Table 11. Size measures-results

Metric	System 1 results			System 2 results		
	Maximum Minimum	Mean and median	Standard Deviation	Maximum Minimum	Mean and median	Standard Deviation
NAI	5,0	1.14,1.00	1.10	22,0	5.4,5	4.30
NM	13,0	3.79,4.00	2.51	46,2	13.3,11	9.33
NMNpub	3,0	0.05, 0	0.40	10,0	1.58,0	2.76
NMpub	5,0	1.91,2.0	1.01	46,0	11.8,9	10.16
Stmts	82,3	17.52,15	12.01	554,2	112.05,100	84.5

An Empirical Investigation on the Impact of Training-by-Examples on Inspection Performance

Atiq Chowdhury and Lesley Pek Wee Land

The University Of New South Wales
School of Information Systems, Technology and Management
Sydney 2052, NSW, Australia.
z2275486@student.unsw.edu.au, l.land@unsw.edu.au

Abstract. Software inspection is often seen as a technique to produce quality software. It has been claimed that expertise is a key determinant in inspection performance particularly in individual detection and group meetings [33]. Uncertainty, among reviewers during the group meetings due to lack of expertise is seen as a weakness in inspection performance. One aspect of achieving expertise is through education or formal training. Recent theoretical frameworks in software inspection also support the idea of possible effects of training on inspection performance [33]. A laboratory experiment was conducted to test the effects of training by examples on requirements inspection. Our findings show that the trained group performs significantly better than the group which received no training. However, the 'experienced' reviewers did not outperform those with no experience. The results have implications the use of a repository of defect examples for training reviewers.

1 Introduction

Software inspection or review has emerged as a popular technique for the development of quality software [13],[16],[19]. Research shows that inspection applied to the early stages of the software development life cycle incurs significant savings in terms of cost and time [4]. Therefore it is worthwhile to try and understand ways to improve the performance of requirements inspection.

There are various factors that have a significant impact on the number of defects detected [21]. The impact of inspector's expertise/experience on inspection performance is an area in which limited empirical work has been done. In fact, most empirical studies have focused on other factors such as reading techniques (Ad-hoc, Checklist, Scenario Based Reading, Usage Based Reading and Perspective Based Reading) [1],[29] and group meetings [22],[31]. Nevertheless Sauer et al's work [33] suggests that most important factor in determining the effectiveness of software inspection is the level of expertise of individual reviewers. They suggest that work in this area may highlight ways to improve inspection performance via the selection of reviewers.

A majority of the defects are detected during individual detection and group meetings. Group and individual reviewers fail to make any decisive decisions as to whether or not a defect should be carried forward to the overall list as well as not knowing whether to discuss a potential defect. This could be due to many possible

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 203–219, 2004.

factors such as lack of confidence, lack of time, production blocking, free riding (see [22]), and inadequate expertise and experience [22]. If such losses can be minimized if not eliminated, then collectively there is a possibility a more complete quality software product produced. This leads to the question of how we can increase the level of individual task expertise or develop ways to improve individuals' expertise. Thus expertise is a key area that needs more attention such that individual and groups can make better decisions with an increased degree of confidence.

One aspect of expertise is the technical education or training of inspectors. In the field of education, training can be used to increase individual's expertise level and improve learning [36],[38]. The inspection literature claims that training is crucial to improve individual's expertise [15], however the nature or type of training required has not been formally tested. Training in the inspection literature commonly refers to high level teaching about the "what" and "why" regarding inspections. "How" the defect detection task should be executed is usually embedded in the various inspection aids recommended (e.g. perspectives, scenarios, checklists). We argue in this paper that inspectors' training in defect detection needs to be more explicit and comprehensive. Proper task training may likely contribute to the "expertise" level of inspectors and thus improve inspection outcome.

The focus of this paper then is to test the impact of training by (defect) examples on inspection performance. The main motivation for this paper comes from the extended behavioral theory of performance for software inspections where it has been proposed that characteristics of inspectors such as technical and non-technical factors have an impact on inspection performance [23]. This paper will empirically test part of this framework via a formal inspection experiment conducted on a piece of requirements document. The impact of such training in software practices may have possible affects on how reviewers can be trained for the review task. We speculate that appropriate reviewer training may help improve review performance particularly where reviewers lack appropriate expertise to perform the task.

Section 2 summarizes the literature relevant to this research. It includes the theoretical and empirical work in the inspection literature, and also presents work done in the field of training in education. Section 3 explains the research questions and Section 4 presents the experimental study in detail. Section 5 reports the results of the experiment, followed by the discussion and conclusion in Sections 6 and 7, respectively.

2 Literature Review

2.1 Inspector's Experience

Fagan (1976), initially raised the issue that people needed to be taught or prompted to find errors effectively. It was suggested that one approach to getting started was to make a preliminary inspection of a design or a code that is representative of the actual program to be inspected. However, not enough empirical research has been conducted or been tested to prove the theory that experience result in better defect detection. Researchers have suggested that the selection of inspectors based on their characteristics for examples software development experience and domain experience

can impact the defects found during the inspection process [28],[30],[33]. In fact a study conducted by NASA engineers showed that there were some indications that inspector's experience had an impact on number of defects detected [1]. Some of the characteristics that have been investigated in the past concern the experience of the inspectors in different tasks such as writing requirements, using requirements, testing software and writing uses cases to name a few.

One proposed method for gaining process experience is to have subjects work in pairs and gain experience through observation [5]. Carver et al (2003), believes that subjects could gain inspection experience by observing another subject performing an inspection as previous study showed that observers were able to gain useful insight into the execution of a technique [35]. Carver et al (2003), executed an experiment in training students how to use Perspective Based Reading (PBR) technique. Training was done through the observation medium. The results indicated that the observation was beneficial to students in terms of gaining understanding and confidence. Furthermore, it demonstrated that inspectors who observed performed better than non-observant subjects in low experience group for one requirements document. However in another requirements document the result was contradictory. Thus the results are not conclusive and further experiments need to be carried out to allow researchers to have confidence that observation is an appropriate method of training.

It must be noted that there is no definition of what constitutes an expertise particularly in relation to software inspection. Is expertise defined as programming experience, number of years in software inspection or expertise in reading requirements specification and code? Or are there any other aspects to it? This key determinant needs to be addressed so it can provide valuable practical guidelines on the selection of inspectors in the software industry.

Biffl (2002), measured inspector capability factors as inspector development skills, quality assurance experience and ability to conduct inspection with a particular reading technique on different document with known defects. Surprisingly their empirical research indicated that number of years of experience has no significant influence on inspection performance. One would think that individuals with plenty of experience would be able to perform better in defect detection than individuals with little or limited experience. Other studies indicated that different types of experience with inspection such as comfort level, experience with reviewing requirements and experience in software inspection techniques could be important factors in the effectiveness of an inspection [6]. However a more substantial research and experiments need to be carried out regarding the effect of experience in inspection performance.

While the effects of expertise on software reviews have not been directly studied, there are substantial theory that indicates the importance of expertise in review performance. One of the reasons that such effect cannot be studied is because it is not easy to control for expertise level where subjects are not experienced software practitioners. Initially subjects start at different experience level and needs to be balanced up in order to correctly manipulate expertise. Otherwise contradictory results will form and the true effect of expertise will not be recognized. We see training by worked examples of different defect types as a method to balance the expertise variable between the subjects as it will provide less experienced members to gain the additional skills required and knowledge to perform the inspection with the more experience reviewers.

We need an underlying theory that emphasizes the importance of individual task expertise in the performance of a review. Thus a behaviorally motivated theoretical framework is needed which will assist in directing this research in a more focused manner.

2.2 The Extended Behavioral Theory of Performance for Software Inspections

Sauer et al (2000), formulated a behavioral theory on software technical reviews, based on the 2-staged defect detection task. The 2-staged task model includes the individual defect detection (preparation) stage, followed by the group meeting (defect collation) stage. Recently, this theory has been extended by [23] to address the limitations posed in the original theory (depicted in Figure 1).

The original theory highlights the performance of a group can be closely modeled by the performance of individual members and combining it to give a greater weight to the performance of the most expert individual in the group [33]. This acknowledges that group performance is determined by expertise. A group's decision based on majority of responses may not guarantee that the correct decision was made, but an expertise influence on the decision making is likely to produce a more accurate decision.

The extended theory addresses a number of limitations posed in the original theory. Given the recent argument against inspection meetings, the extension focuses on the additional factors impacting individual defect detection, which includes the inspection process, software artifact type and the individual characteristics of inspectors [23].

This model as shown in Figure 1 below states that a relationship exists between characteristics of inspectors (technical and non-technical) and inspection performance; performance is maximized when inspectors of the right expertise are selected for the task. The type of artifact inspected also affects both the characteristics of inspectors as well as the inspection process, and hence the final performance outcome [23]. There is scarce evidence of the impact of technical factors on inspection performance, especially in a controlled laboratory setting. Both the original and extended theory maintains that expertise as exhibited by individuals is a key determinant of performance.

Land et al (2003), model shows that there is a relationship between individual characteristics and inspection performance. Boehm (1981), in the COCOMO model which can predict the cost of a software system being developed included Analyst capability, Programmer capability, Application experience, Platform experience, Language and tool experience and Personal continuity as factors that influence the effort estimate produced by the model. Boehm (1981), also highlighted that application specific experience are also vital issues to consider. It is thus interesting to see what other variables in software engineering in individual characteristics play a part in inspection performance. A better understanding of this knowledge will enable us to determine what specific characteristics of individuals have significant impact on inspection performance. Some of the characteristics of individuals that have been studied are Application Domain Knowledge, Software Development Experience and Inspection Process Experience.

We argue in this research that individual's defect detection performance can be improved by increasing task expertise through education and training. In the next

section, we review the relevant literature in this area to find supporting evidence for this idea.

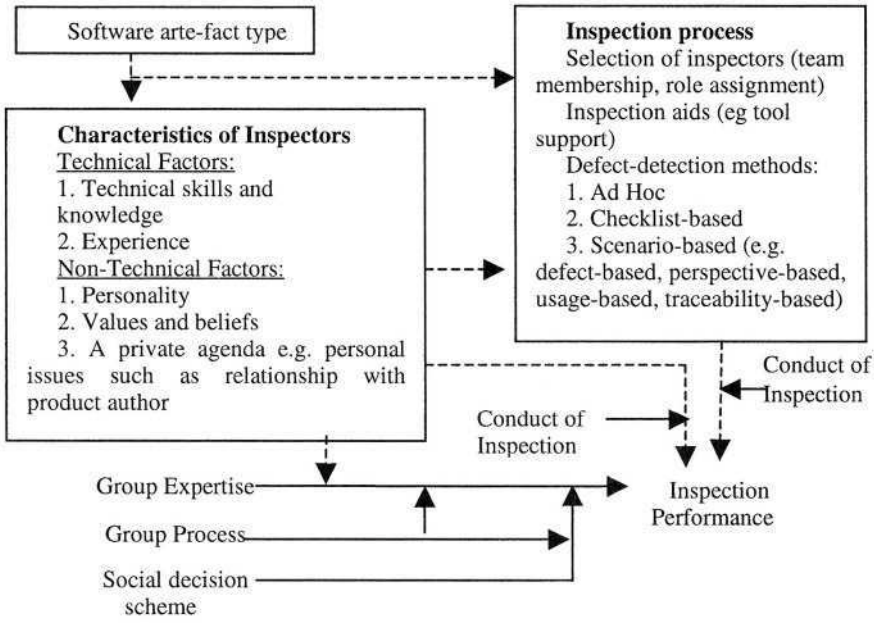


Fig. 1. The extended behavioral theory of performance for software inspections [23]

2.3 Education and Training

Education and training are most effective when high quality instruction is presented under conditions that are conducive to learning [27]. One approach to facilitate the effectiveness of education and training programs is to help students develop better learning strategies. Effective learning strategies results in less time consuming in practice, enhance self esteem for the learner and fewer demands on the instructor's time [27]. Activities student engage in when encountering learning task can be modified through instruction and training to make them more effective and efficient.

Empirical study has been executed in evaluating the impact of training in terms of learning performance. Dansereau (1978), used training materials such as short lectures, practical exercises and discussions which showed that training improved the student's learning behaviour and attitude. On a more recent note [17] investigated the impact of the type of training and the duration of training on a transfer task. The results indicated that length of training and type of training had bearing on improving student's learning performance.

2.3.1 Worked Examples as a Training Method

Most researchers are aware of the importance of worked examples to students assimilating new material. The ability to process worked examples correctly is an essential part of training [37]. Worked examples provide additional information as

they entail schema acquisition (ability to recognize appropriate circumstances and actions required by those circumstance) and rule automation (information being processed with minimal conscious). This was evident in the experiment conducted by [36] using algebra equations which exhibited that groups with worked examples made significantly less errors up to nine times less than the groups without any worked examples. This encompasses the notion that worked examples are a more efficient mode of learning than solving conventional problems. However when faced with a different problem type, students will not be able to recall with what's been learnt in the worked examples due to being immersed with a schema. Furthermore experimental results by [11] showed that 75% of worked examples group, but none of the conventional problem groups were able to solve the transfer problems, clearly highlighting worked examples can facilitate transfer performance.

On the contrary [8] results indicated that problem solving is superior to worked examples. One major disadvantage of using worked examples is student's inability to learn how to solve other problems apart from the ones shown in the example. A heavy use of worked examples can provide learners with solution patterns such that they fail to solve new or different types of problems. As a result mechanisms need to be implemented to ensure individuals fully process the contents of an example.

It is quite convincing that the use of worked examples may be beneficial to learning outcomes and the transfer of the knowledge gained into solving new problems. This is due to learners viewing worked examples as the primary and natural source of learning material [24]. Numerous empirical studies in the education field have focused on worked examples and instruction. These studies have shown that such practices produces better performance in learning. This type of work has not been done yet in software inspection. While the inspection task is vastly different to those task specified in the education literature, especially in terms of the task complexity, it is still an open question whether similar performance improvements can be achieved through the use of worked examples as a training method. Inspection training [15],[16] is typically targeted at a high level (e.g. explaining what and why, process variables), "how" to execute defect detection is usually explained via the use of inspection aids such as checklist and scenarios. The possibility of improving defect detection through an alternative way via training poses an interesting option. As a result, this paper will particularly address the impact of training by examples on inspection performance.

3 Research Model and Hypothesis

In Figure 2, we present the research model of this study. It is important to note that this model is an elaboration of a part of Land et al's model. This model purports that training using worked examples of various defect types affect individual's characteristics (inspector's technical knowledge, skills, and experience), which in turn can affect inspection performance. Our focus in this study is on the importance of individual's characteristics as a determinant of performance, rather than inspection process. This does not mean that process is not an important determinant of performance.

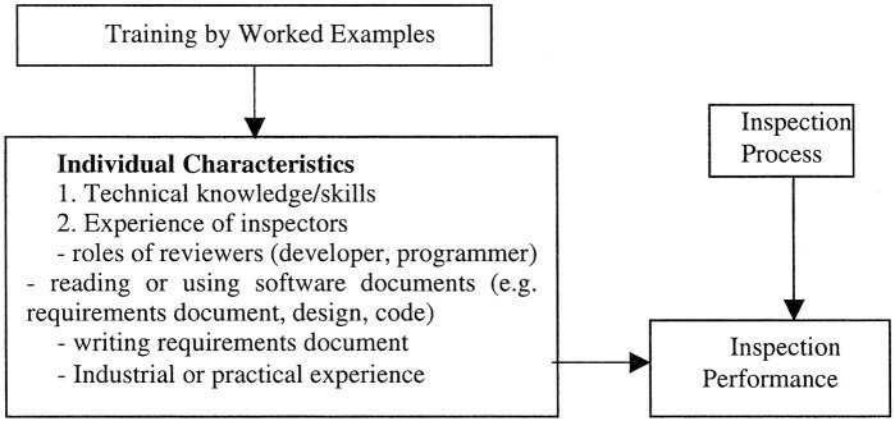


Fig. 2. Research Model

Individual characteristics are composed of technical skills and experience of inspectors. Experience of inspectors is accumulated through individuals working in an IT environment such as a programmer, analyst, developer, tester or manager; reading or using software documents such as requirements, design or code; writing requirements document; practical experience to aspects of inspection task and any involvement in industry projects relating to software development.

Training by worked examples may enhance the existing experience of inspectors and learning outcomes due to the transfer of knowledge acquired through training to the new task at hand. Experience in various roles and practical experience in software development could possibly add another dimension to the inspector's ability to perform the task. Being equipped with technical skills and knowledge through training by examples, it is expected that there will be an improvement in inspection performance. Inspection performance is determined by the number of true defects detected (defects that need to be repaired for the software to work), classification of defects in the correct category and number of false positives reported (non-true defects).

This leads to the following research questions that will be addressed in this experimental study.

Research Hypothesis 1

$(H_{0,1})$ – There is no difference in finding the number of true defects between an inspector who has been trained with worked examples of defect classes and an inspector who has not been trained.
$(H_{1,1})$ - An inspector who has been trained with worked examples of defect classes finds more true defects than the one with no training.

An inspector who has been trained with how to look for defects and what defects to look for, using concrete examples will increase in the knowledge and skill of defect detection. Worked examples are realistic defect instances, and thus will likely improve familiarity and confidence when inspectors actually performance the task

themselves afterwards. By contrast, an inspector who has no such exposure to concrete examples may find it hard to visualize how to look for defects, where they may occur, or what defect descriptions look like.

Research Hypothesis 2

$(H_{0,2})$ - There is no difference in classification of true defects in the correct category between an inspector who has been trained with worked examples of defect classes and an inspector who has not been trained.
$(H_{1,2})$ - An inspector who has been trained with worked examples of defect classes is able to classify more true defects in the correct category than the one with no training.

Concrete defect examples are better illustrators of what defect look like than defect class descriptions. Therefore, an inspector who has been trained will get a better insight into the different types of defects that exist, compared with an untrained inspector. Trained inspectors will be able to better classify defects perhaps by simulating or applying the new expertise learnt. An untrained inspector may resort to making irrational classifications or random guessing due to insufficient knowledge about defect classifications.

Research Hypothesis 3

$(H_{0,3})$ - There is no difference in reporting of false positives between an inspector who has been trained with worked examples of defect classes and an inspector who has not been trained.
$(H_{1,3})$ - An inspector who has been trained with worked examples of defect classes finds fewer false positives than the one with no training.

Since worked examples during training illustrate cases of true defects, we do not expect that training will impact false positive identifications. False positive raised are most likely due to incorrect knowledge or understanding of the artifact. Training is unlikely to affect faulty understanding, and thus we hypothesize no difference in false positive detections when training is applied.

Research Hypothesis 4

$(H_{0,4})$ – There is no difference in finding true defects between an experienced inspector and a non-experienced inspector.
$(H_{1,4})$ - An inspector who is experienced finds more true defects than the one with no training.

Research Hypothesis 5

$(H_{0,5})$ – There is no difference in classification of true defects in the correct category between an experienced inspector and a non-experienced inspector.
$(H_{1,5})$ - An inspector who is experienced can classify more true defects in the correct category than a non-experience inspector.

Research Hypothesis 6

$(H_{0,6})$ – There is no difference in reporting of false positives between an experienced inspector and a non-experienced inspector. $(H_{1,6})$ - An inspector who is experienced reports fewer false positives than a non-experienced inspector

An expert's skill and knowledge is most likely acquired through past experience, after many years of experience in their domain [9]. Inspectors who have acquired significant knowledge through practical IT experience as a programmer, developer, tester or analyst would collectively be able to recall all the tasks that they have been associated with and use this past occurrence to positively impact their inspection performance. Skill acquisition is due to gradual improvement during extended experience in a domain [32] and because experience inspectors have acquired such skills through continuous involvement in numerous projects over a period of time, they would be more attentive and make more accurate judgments in defect detection and in classifying defects correctly.

4 A Controlled Laboratory Experiment

A controlled experiment was conducted in an academic environment to test the hypotheses posed in the previous section. The main purpose of this study is to test the impact of training on inspection performance.

4.1 Experiment Context

The experiment was part of a university postgraduate students studying Master of Commerce specialising in various disciplines such as Information Systems Management (ISM), Accounting, Finance and Information Science.

There were 44 students who participated in this experiment. The students had a varying degree of IT experience in which some had practical experience in roles such as a programmer, analyst and a developer for a significant number of years. The inspection task was integrated into the course structure. Student consent, confidentiality agreement, etc were obtained prior to the experiment, in order to fulfill the requirements of the ethics committee of the university.

4.2 Experiment Variables

The experiment uses one treatment variable – training of subjects. There are three dependent variables:

1) number of true defects detected correctly
2) number of defect types classified correctly
3) number of false positives reported

4.3 Experimental Material

The inspected material is a requirements document describing an Automatic Teller Machine (ATM)[1], which has been used widely in other inspection experiments worldwide. The document is broken down into introduction, general description and specific requirements that consist of functional and performing requirements. There is also graphical diagram to aid the student in understanding the document. The size of the document is 16 pages with 30 seeded defects. The document was also reviewed by two experienced reviewers prior to the experiment, to validate the instrument, and to check that the size of the artifact was not too large for the inspection duration.

A defect in the requirement document is classified as Ambiguous, Incorrect Fact, Extraneous, Inconsistent, Omission and Miscellaneous. Each student was given a definition of the defect types to help them identify the types of errors detected. The distribution of defect types in the list of seeded defects is as follows: 9 Ambiguous, 1 Extraneous, 3 Inconsistent, 8 Incorrect Fact, 1 Omission and 10 Miscellaneous. In addition a questionnaire was created that recorded the level of experience students had in practical IT, Software Development Life Cycle (SDLC) as well as any experience in using, reading or writing requirements document. The questionnaire also recorded their background such as what course they are undertaking at university.

4.4 Experiment Procedure

There were three lectures conducted each on a different day and location. The experiment consisted of a half an hour lecture that included background information about inspection, two stage phases of defect detection and defect collection, definition of a defect and concluded with definition of different defect types (omission, incorrect fact, inconsistent, ambiguous, miscellaneous and extraneous). This lecture was given to every student who participated in the experiment. The training session itself took place immediately after the lecture. A questionnaire was first filled out by the students. The training session involved students reading a short general description of a video system with samples functional description that consisted of seeded defects. The defects were explained to the students in relation to the requirements document as part of the training process.

The inspection task itself was conducted one week after the training session. Each student was issued with a requirements document on Automatic Teller Machine (ATM), a defect form to record their individual findings and a definition of the different defect type. Each of the three inspection sessions went for one hour. Immediately after the inspection task, all experiment materials were collected.

It must be noted that this training of students with worked examples of different defect types was provided to students in the 1st lecture. The students in the 2nd lecture did not receive any training at all. In the third lecture, the class was required to be divided up to make the total number of students in both trained and not trained groups equal. This division of the class was totally volunteered by the students themselves without the knowledge of why they were being divided up to create a balanced treatment. In total there were 16 students in the first lecture that were trained, 11 in the second lecture with no training and 6 trained and 11 not trained students in the

[1] http://www.cs.umd.edu/projects/SoftEng/ESEG/manual/pbr_package/manual.html

third lecture respectively. This training of how to identify defects was only given to randomly selected students and this was the difference among students participating in this experiment.

4.5 Data Collection and Analysis

To investigate the influence of training by examples on the number of defects detected, the type of defect class detected correctly and number of false positives reported, all subjects recorded their findings in a defect form. Each student's data was compared with a master defect form from which a tabular result is created consisting of each student's defect performance in addition to their background to requirements documents. From this data, it was calculated the impact of training by examples on the number of defects detected, the type of defect class detected correctly and number of false positives. In evaluating using and writing requirements document, firstly two groups were established with one group consisting of subjects with less than or equal to 2 years of experience and the other with more than 2 years of experience respectively. In addition, when assessing Software Development Life Cycle (SDLC) impact on inspection performance, three groups were formed. One group had no experience in SDLC at all, second group with theoretical experience only and the third with practical experience. Reported syntax errors and unclear description were not included in the data collection and analysis. It must also be noted that an outlier was present in the data collected in the training group and was removed in the final analysis.

Different tests were used depending on whether the variables have normal distribution or not. The normality of data distributed using Shapiro-Wilk statistic due to the sample being below one hundred. In normal distribution, independent t-test was used and non parametric test such as Mann Whitney and Kruskal Wallis test were used if the distribution was not normal. The significant criterion is set at 0.05 as recommended for empirical research in Software Engineering [26].

4.6 Threats to Validity

This section exhibits a number of internal and external validity threats in this experiment.

4.6.1 Threats to Internal Validity

Selection refers to any pre-existing differences between individuals in the different experimental condition that can influence the dependent variable. However the subjects in our experiment had a varying degree of development IT experience. It was not determine whether this pre-existing difference had any impact on the results obtained as students were not divided o classified according to their skill level. In this experiment, we minimized such occurrences by randomly assigning students to the trained and not trained group. The questionnaire is also included in the analysis to reduce this pre-existing difference. Thus it is believed that the likelihood of selection threat in this experiment is minimized.

Maturation involves any naturally occurring process within persons that could cause a change in their behaviour. Changes may occur due to boredom, tiredness or learning. The experimental lectures were held late in the afternoon and evening where the students already attended few other classes on the day. Due to the length of the questionnaire and the requirements document, slight maturation effects could occur during the once-only test. This may include fatigue or boredom, which may induce some participants to look for defects without as much thought or motivation as earlier ones. However the grading of students such as awarding a small percentage of mark as a motivation tool may have reduced the impact of this threat.

History refers to any event that coincides with the independent variable and could affect the dependent variable. Since the experiment was carried over 7 days in three different locations with various experimental environment or conditions will form a threat to validity. In addition major historical event such as the war in Iraq was taking place at that time such that subjects could have affected by it. However this is not seen as a threat as no relationship exists between the subjects attitude to external events with inspecting a requirements document. Furthermore there were no noise disruptions, power failures or any other unwanted interruptions during the experiment.

Mortality threat arises when some subjects do not continue through the experiment. Because the subjects were highly motivated by grading, there were not many cases of subject drop-outs. In fact three people dropped out of the training group and six in the no training group before the inspection task began, not during the task. It was decided to use only data of subjects who completed the entire inspection.

Finally there was no instrumentation effect as each group was given same experimental materials.

4.6.2 Threats to External Validity

Representative subjects. The students participated in the experiment may not be representative of the population of software professionals. Since the subjects are university students, the skill level will differ significantly in comparison to software professionals. All subjects have a university degree of various disciplines prior to their current degree which will result in a large variance in level of expertise. However a report on estimation task showed that there is no significant difference between student and professional subjects [18].

Artifact. The requirements document inspected in the experiment may not be a true representation of industrial requirements document. In this experiment the document was simpler and less complex than industrial ones. However the artifact has been widely used in other empirical study particularly in examining reading techniques [1].

All these threats are liable in conducting classroom experiments and can be overcome by conducting replication with people that are representative of software professionals, artifact and process that represent industrial practice.

5 Results

Research Hypothesis 1 and 4
A trained inspector finds more true defects than one with no training (means 2.21 vs. 1.00, p=0.037). Thus null hypothesis 1 is rejected, alternate hypothesis is supported. Null Hypothesis 4 is not rejected based on the results obtained (independent sample t-test, using requirements document p=0.983>0.05, means 1.50 vs. 1.69, independent sample t-test, writing requirements document p=0.822>0.05, means 1.50 vs. 1.68). Thus there is no difference in finding true defects between an experienced and a non experienced inspector.

Research Hypothesis 2 and 5
Null Hypothesis 2 is rejected based on the significance value obtained through Mann Whitney test (p=0.043<0.05, means 1.16 vs. 0.38). Thus inspector who has been trained is able to classify more true defects in the correct category than the one with no training, alternative hypothesis 2 is supported. Null Hypothesis 5 is not rejected (Mann Whitney test, using requirements document p=0.949>0.05, means 0.50 vs. 0.86, Mann Whitney test, writing requirements document p=0.437>0.05, means 0.25 vs. 0.87 respectively). Thus the null hypothesis 5 cannot be rejected highlighting that there is no difference in classification of defects correctly between an experienced inspector and a non-experienced inspector

Research Hypothesis 3 and 6
There was no difference in terms of which group identified more false positives. In fact the training group had a slightly higher mean than the no training group, but the value is negligible. Null Hypothesis 3 is not rejected based on t-test result (p=0.619>0.05, means 11.5789 vs. 11.0000). There is no difference in reporting of false positives between training and no training group.

On a different note, experienced subjects had a slightly fewer mean than no experienced subjects (independent sample t-test, using requirements document p=0.796>0.05, means 10.8333 vs., 11.4138, independent sample t-test, writing requirements document p=0.504>0.05, means 9.750 vs. 11.560 respectively). Thus the null hypothesis 6 cannot be rejected, that there is no difference between experienced and non experienced subjects in reporting of false positives.

On a further note, Friedman test was carried out to see whether theoretical, practical or no experience in SDLC have any impact on inspection performance. The result indicated that significant difference did exist in inspection performance (0.01<0.05) across the three groups and that theoretical experience appeared to have a lesser impact.

6 Discussion

It was initially proposed those individuals with training and individuals with experience in SDLC, requirements document and any practical technical experience

would perform better in terms of defect detection, classification of defects correctly and reporting fewer false positives.

It has been shown that training by examples had made a positive difference in individual's performance regardless of being experienced or not. We believe the reasons of why individuals with training performed significantly better is due to a few explanations. Firstly, training enabled individuals to be familiar with requirements document such as the structure and what to expect in such an artifact. This may have increased their confidence level and minimized any certainty or confusion that positively affected their inspection performance. Secondly, during the lecture and training phase, the notion of false positives was not explained to the subjects at all nor was any examples shown. This may be the central reason as to why there was no significant difference between trained and non-trained inspectors in reporting false positives. Finally, subjects did not have enough practical or theoretical technical experience to be able to review a requirements document. In fact the subjects in this experiment were a combination of disciplines from Accounting, Information Systems and Information Science. However, the majority of students were undertaking a Information Technology subject for the first time in which they had very little exposure to programming, developing applications or even being part of a computing environment. As a result the selection of subjects accounts to why experience in SDLC and requirements document did not have significant impact on individual's inspection performance. Furthermore this is reflected in the low means obtained across all levels. This is seen as a limitation of this experiment.

7 Conclusion

Inspections have become an imperative tool in achieving high quality software. Much of the inspection research has been done on evaluating the performance of reading techniques. Interestingly the impact of experience of inspectors has received considerable less attention. As a result an empirical research was carried out in examining the impact of training by worked examples of defect types on inspection performance.

From our analysis of the experimental data, several conclusions were drawn. As with any experiment, these conclusions apply only to the experimental settings from what they were drawn. Threats to external and internal validity must be considered carefully before generalizing these results. The findings of the experiment are the following:

1) An inspector who has been trained finds more true defects than the one with no training.
2) An inspector who has been trained is able to classify more true defects in the correct category than the one with no training.
3) There is no difference in reporting of false positives between a trained inspector and a non-trained inspector.
4) There is no difference in finding true defects between an experienced inspector and a non-experienced inspector.
5) There is no difference in classification of true defects in the correct category between an experienced inspector and a non-experienced inspector.

6) There is no difference in reporting of false positives between an experienced inspector and a non-experienced inspector.

The results of this work have important implications for software practitioners in performing inspection. The implication of the findings is to create a repository of defect examples. A storage of defect examples can be developed such that inspectors can familiarise themselves with the different defect classes. This will particularly assist novice inspectors, as is the case with our experimental subjects

One serious limitation of this study is the low level of IT experience of the subjects. Therefore, the extent to which we can generalize these results to IT professionals is questionable. Further work is required before we can conclusively state that training by examples has positive effect in inspection performance. We also suggest using subjects with more industry experience, and providing more worked examples.

Future work can also address the method in which training is imparted (e.g. web simulations). Group performance was not studied, it will be interesting to investigate the impact of training on group meetings.

References

1. Basili, V.R., Green, S., Laitenburger, O., Lanubile, F., Shull, F., Sorumgard, S., Zelkowitz, M. (1996), "The Empirical Investigation of Perspective-Based Reading" Empirical Software Engineering: An International Journal, 2(1):133-164, Kluwer Academic Publishers
2. Biffl S., Halling M., (2002), "Investigating the Influence of Inspectors capability factors with Four Inspection Techniques on Inspection Performance", Proc. IEEE Int'l Software Metrics Symp.
3. Biffl S., Halling M., (2003), "Investigating the Defect Detection Effectiveness and Cost Benefit of Nominal Inspection Teams", IEEE Transaction on Software Engineering, vol. 29, no. 5, pp. 385-387
4. Boehm, B.W. (1981), "Software Engineering Economics", Advances in Computing Science and Technology.Prentice Hall
5. Carver J., Shull, F., & Basili, V., March (2003), "Investigating the Effects of Process Experience on Inspection Effectiveness.", University of Maryland Technical Report CS-TR-4442.
6. Carver J., April 2003, "The Impact of Background and Experience on Software Inspections.", PhD Thesis, University of Maryland
7. Carver J., and Victor Basili., October 2003, "Identifying Implicit Process Variables To Support Future Empirical Work.", *Proceedings of the 17th Brazilian Symposium on Software Engineering (SBES 2003).* p. 5-18.
8. Charney, D.H. & Reder, L.M (1986),"Designing interactive tutorials for computer users: Effects of the form and spacing of practice on skill learning", Human-Computer Interaction, 2, 297-317
9. Chase W.G. Simon H.A., (1973), "The mind's Eye in Chess. In Visual Information Processing". Ed. WG Chase. New York: Academic, p215-81.
10. Cheng, B., and Jeffery R., (1996), "Comparing Inspection Strategies for Software Requirements Specifications", In Proceeding of the 1996 Australian Software Engineering Conference, pp. 203-211

11. Cooper, G. & Sweller, J. (1987), "The effects of schema acquisition and rule automation on mathematical problem-solving transfer", Journal of Educational Psychology, 79, 347-362
12. Dansereau, D, F, (1978), "The development of a learning strategies curriculum". In H.F. O'Neil, Jr. (Ed.), Learning Strategies. New York; Academic Press
13. Fagan, M.E. (1976), "Design and code inspections to reduce errors in program development", IBM Systems Journal, 15(3):182-211
14. Fowler, P.J., (1986), "In-process inspections of work-products at AT&T, At&T Journal, March/April, 102-112"
15. Freedman, D.P., Weinberg G.M. (1990), "Handbook of Walkthroughs, Inspections and Technical Reviews: Evaluating Programs, Projects, and Products", Third Edition, Dorest House Publishing
16. Gilb, T., Graham. D. (1993), "Software Inspection", Addison Wesley
17. Goska R., Ackerman P., (1996) "An Aptitude-Treatment Interaction Approach to Transfer With-in Training", Journal of Educational Psychology, Vol 88, no. 2, p 249–259
18. Host, M., Regnell, B., & Wohlin, C., (2000), "Using Students as Subjects – A Comparative Study of Students and Professionals in Lead-Time Impact Assessment", Empirical Software Engineering, vol 5, pp. 201-214
19. Humphrey W.S, (1995), "A Discipline for Software Engineering", Addison-Wesley Publishing Company
20. Kim, L.P. W., Sauer, C., Jeffery, R., (1995), "A Framework for software development technical reviews, Software Quality and Productivity: Training, Practice, Education and training, Eds. Matthew Lee, Ben-Zion Barta and Peter Juliff, IFIP, Chapman and Hall, pp294-299"
21. Laitenburg O., & DeBraud J., (2000), "An Encompassing Life Cycle Centric Survey of Software Inspection", Journal of Systems and Software, 50(1): 5-31
22. Land L., (2000), "Software Group Reviews and the Impact of Procedural Roles on Defect Detection Performance", Thesis, University of New South Wales
23. Land L., Wong B., Jeffery R., (2003), "An Extension Behavioural Theory of Group Performance in Software Development Technical Reviews" APSEC
24. Lieberman, H. (1986). "An Example Based Environment for beginning Programmers." Instructional Science, 14(3), 277-292
25. Melo, W., Shull F., and Travassos, G.H., (2001), Software Review Guidelines Technical Report ES-556/01 Systems Engineering and Computer Science Program. COPPE. Federal University of Rio de Janeiro, September
26. Miller, J., Daly, J., Wood, M., Roper, M., Brooks, A. (1997), "Statistical power and its subcomponents – missing and misunderstood concepts in empirical software engineering research", Information and Software Technology, 39:285-295
27. O'Neil, H.F. & Spielberger,C., (1979), "Cognitive and Affective Learning Strategies", Academic Press
28. Parnas, D.L. and Weiss, D.M., (1985), "Active Design Reviews: Principles and Practices", Proc of the 8th International Conferences on Software Engineering, pp. 418-426
29. Porter, A. A., Votta, L.G., Basili, V.R. (1995), "Comparing detection methods for software requirements inspections: A replicated experiment", IEEE Transactions on Software Engineering, March, 21(6):563-575
30. Porter, A. A., and Johnson, P.M., (1997), "Assessing Software Review Meetings: Results of a Comparative Analysis of Two Experimental Studies", IEEE Transaction on Software Engineering, vol 23, no. 3, March, pp. 129-145
31. Porter, A. A., & Votta, L.G. (1997), "What makes Inspection Work?", IEEE Software, pp. 99-102
32. Proctor, R. W., and A. Dutta, 1995, "Skill Acquisition and Human Performance". Thousand Oaks, CA: Sage

33. Sauer C., Jeffery R., Land L., Yetton P., January (2000), "The Effectiveness of Software Development Technical Reviews: A Behaviourally Motivated Program of Research", IEEE Transaction on Software Engineering, Vol. 26, no. 1
34. Schneider, G.M., Martin J., Tsai, W.T., (1992), "An experiment study of fault detection in user requirements document", ACM Transactions on Software Engineering and Methodology, April, 1(2): pp. 188-204
35. Shull F., Carver J., and Travassos, G., (2001), "An Empirical Methodology for Introducing Software Processes." In *Proceedings of the Joint 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9),* Vienna, Austria, Sept. 10-14,p. 288-296.
36. Sweller, J. & Cooper, G.A. (1985), "The use of worked examples as a substitute for problem solving in learning algebra", Cognition and Instruction, 2, 59-89
37. Sweller, J., (1999), "Instructional Design in Technical Areas", ACER Press
38. Wangari M., Sweller J., (1998), "Learning to Solve Compare Word Problems: The Effect of Example Format and Generating Self Explanation", Cognition and Instruction, Vol. 16, no 2, p173-199

Refactoring Support Based on Code Clone Analysis

Yoshiki Higo[1], Toshihiro Kamiya[2], Shinji Kusumoto[1], and Katsuro Inoue[1]

[1] Graduate School of Information Science and Technology, Osaka University,
Toyonaka, Osaka 560-8531, Japan
Phone:+81-6-6850-6571, Fax:+81-6-6850-6574
{y-higo,kusumoto,inoue}@ist.osaka-u.ac.jp
[2] PRESTO, Japan Science and Technology Agency
Current Address: Graduate School of Information Science and Technology, Osaka University,
Toyonaka, Osaka 560-8531, Japan
Phone:+81-6-6850-6571, Fax:+81-6-6850-6574
kamiya@ist.osaka-u.ac.jp

Abstract. Software maintenance is the most expensive activity in software development. Many software companies spent a large amount of cost to maintain the existing software systems. In perfective maintenance, refactoring has often been applied to the software to improve the understandability and complexity. One of the targets of refactoring is code clone. A code clone is a code fragment in a source code that is identical or similar to another. In an actual software development process, code clones are introduced because of various reasons such as reusing code by 'copy-and-paste' and so on. Code clones are one of the factors that make software maintenance difficult. In this paper, we propose a method which removes code clones from object oriented software by using existing refactoring patterns, especially "Extract Method" and "Pull Up Method". Then, we have implemented a refactoring supporting tool based on the proposed method. Finally, we have applied the tool to an open source program and actually perform refactoring.

1 Introduction

It is well known that software systems are evolving by adding new functions and modifying existing functions over time. On the other hand, through the evolution, the structure of the software becomes more complex. Then, the understandability and maintainability are deteriorating. So, perfective maintenance [9], that is defined as a modification to a software product after delivery to improve its performance and/or maintainability, is an important maintenance activity.

Refactoring is an effective technique to conduct the perfective maintenance. It is defined as "the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure[7]." In [7], several refactoring patterns are described. It is necessary to

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 220–233, 2004.

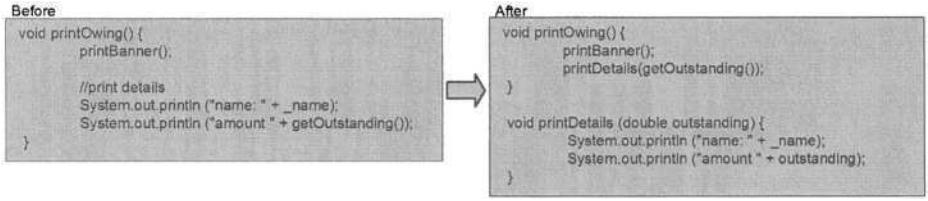


Fig. 1. Example of Extract Method

identify the refactoring candidates that contain "bad-smells" in order to apply refactoring patterns.

Code clone is one of the typical bad-smells that make software maintenance very difficult [7]. A code clone is a code fragment in a source file that is identical or similar to another. Code Clones are introduced because of various reasons such as reusing code by 'copy-and-paste' and so on. Code clones make the source files very hard to be modified consistently. Hence effective code clone detection will support the refactoring activities of perfective maintenance. Up to the present, several code clone detection methods have been proposed [2] [3] [6] [8].

In this paper, we show that the existing refactoring patterns[7] can be used to remove code clones. Then, we propose a method to support refactoring activity by applying code clone detection techniques. Furthermore, we implement a tool supporting our proposed method. This tool uses CCFinder[10], which is a code clone detection tool, and Gemini[16], which is code clone analysis environment. The function of this tool is to find certain code clones to which the refactoring patterns can be applied to. User can get code clone information for refactoring graphically.

2 Preliminaries

In this section, we briefly explain two refactoring patterns, "Extract Method" and "Pull Up Method" that are related to the code clone. Also, we define the code clone.

These patterns can be regarded as one typical activity to remove code clones. In other words, these patterns get code clones into common routine like method by using distinctive functions of object oriented programming language.

2.1 Refactoring Pattern

Extract Method. To put it plainly, "Extract Method" means extraction of a part of existing method as a new method, and the extracted part is replaced by a new method caller shown in Figure 1. In general, this pattern is applied to the case that there is a too long method. In applying the pattern to code clones, a new method, that is a code fragment of code clone, is defined and the original

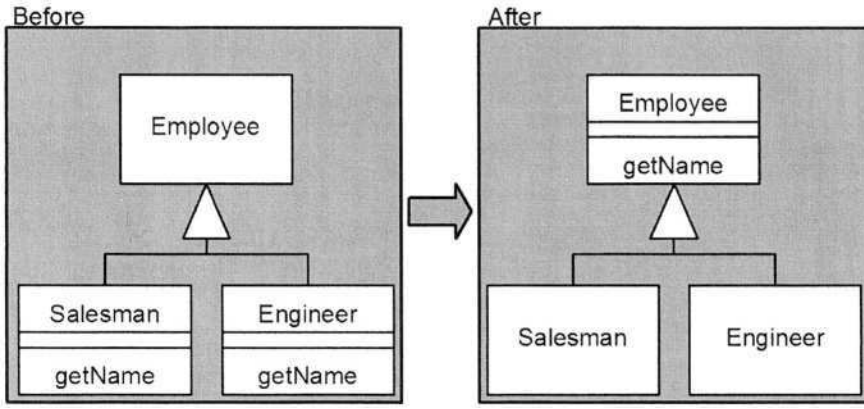


Fig. 2. Example of Pull Up Method

code clones are replaced by the new method caller. As the result, we can remove the code clones.

Pull Up Method. "Pull Up Method" is a simple refactoring pattern. It means pulling up a method which defined in child class to its parent class. If the parent class has several child classes and some of them have the same method (that is, code clone), pulling up the method can remove the code clone.

2.2 Code Clone

Definition of code clone. A clone relation is defined as an equivalence relation (i.e., reflexive, transitive, and symmetric relation) between two code fragments [10]. A clone relation holds between two code fragments if (and only if) they are the same sequences. (Sequences are sometimes original character strings, strings without white spaces, sequences of token type, and transformed token sequences.) For a given clone relation, a pair of code fragments is called a clone pair if the clone relation holds between the fragments. An equivalence

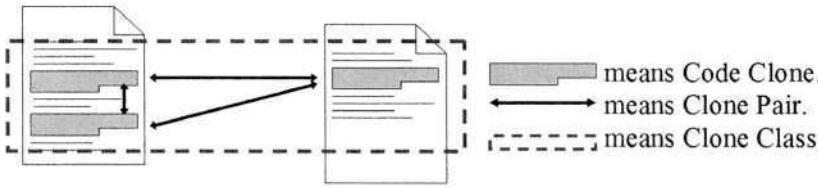


Fig. 3. Clone Relationship

class of clone relation is called a clone class. That is, a clone class is a maximal set of code fragments in which a clone relation holds between any pair of the code fragments as shown in figure 3.

A code fragment in a clone class of a program is called a code clone or simply a clone.

Code Clone Detection Tool: CCFinder. CCFinder detects code clones from program source codes and outputs the locations of the clone pairs in the source codes. The length of minimum clone is specified by a user beforehand. The length of minimum clone is the minimal size of the code fragment that CCFinder detect as a code clone.

Clone detection process of CCFinder consists of the following four steps:

Step1 Lexical analysis: Each line of source files is divided into tokens according to a lexical rule of the programming language. The tokens of all source files are concatenated into a single token sequence so that finding clones among multiple files is performed in the same way as a single file analysis.

Step2 Transformation: The token sequence is transformed, i.e., tokens are added, removed, or changed based on the transformation rules that aim at regularization of identifiers and identification of structures. Then, each identifier related to types, variables, and constants is replaced with a special token. This replacement makes code fragments including different variable names clone pairs.

Step3 Match Detection: From all the sub-strings on the transformed token sequence, equivalent pairs are detected as clone pairs.

Step4 Formatting: Each location of the clone pair is converted into line numbers on the original source files.

Figure 4 illustrates the types of the code clones. CCFinder extracts the following two types of code clone corresponds to a code fragment *C:*

Exact code clone: *E* is a code fragment that is the same as *C* except for the difference about blank, new line and comments.

Renamed code clone: *R* is a code fragment that is the same as *C* except for the difference about the corresponded names of user-defined identifier (name of variables, constant, class, method and so on). Also, the reserved words and the sentence structures are the same between *R* and *C*.

3 Extraction of Refactoring-Oriented Code Clones

3.1 Filtering Approach

As described in Section 2.2, the clone detection process of CCFinder is very fast but only lexical analysis is performed. Since code clones detected by CCFinder are sequences of tokens, they are not necessarily appropriate to be directly

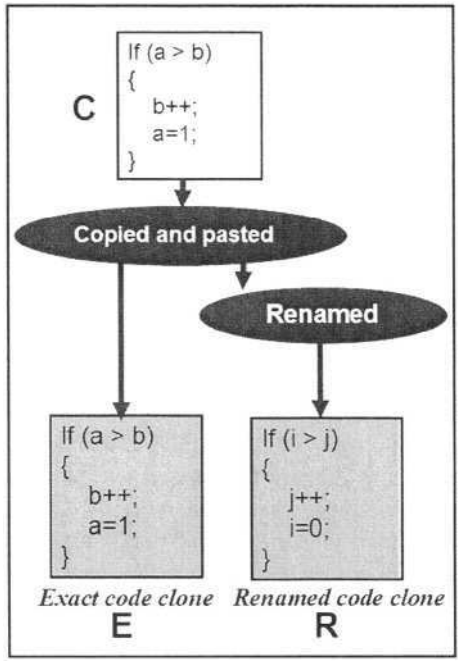


Fig. 4. Exact and renamed code clone

merged into one module (subroutine, function etc.). Some of them are not suitable for refactoring. In order to deal with the problem, we propose a method that extracts refactoring–oriented code clones from the whole set of code clones detected by CCFinder.

We have already proposed the key idea of this approach in [15]. In this paper, we extend the idea, and propose more practical approach to detect distinctive code clones to which these refactoring patterns are easy to apply. In the followings, we explain the detail of the approach.

The key idea is to find a kind of cohesive code fragment (like compound block or method bodies) from the code clone fragments. Figure 5 illustrates an example. In Figure 5, there are two code fragments A and B from a program, and the code fragments with hatching are maximal clones between them. In code fragment A, some data are substituted to list data structure from the head successively. In code fragment B, they are done so from the tail successively. The `for` blocks in A and B have a common logic that handles a list data structure. There are, however, sentences before and after `for` block, that are not necessarily related with the `for` block from semantic point of view. Such semantically unrelated sentences often obstruct refactoring. In other word, extracting only `for` block as a code clone is more preferable from refactoring viewpoint in this example.

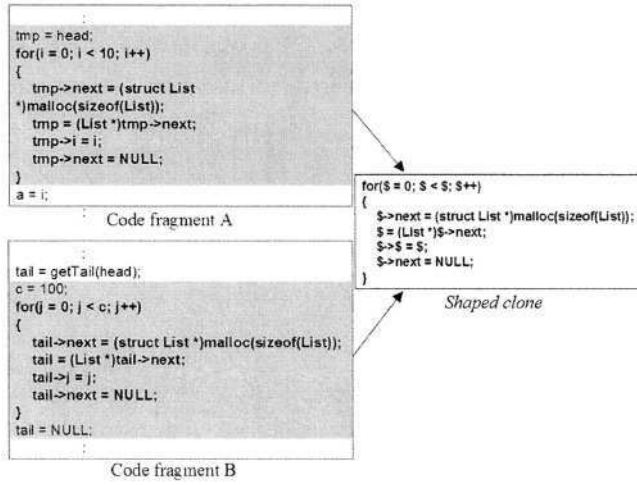


Fig. 5. Example of merging two code fragments

The proposed method is implemented as a filter for the output of CCFinder. We named the filter CCShaper. Currently CCShaper can be applied to only Java language. The extracting process using CCShaper consists of the following three steps:

Step1: Detect clone pairs using CCFinder
Step2: Provide semantic information (body of method, loop and so on) to each block by parsing the source files where clone pair are detected in Stepl and investigating the positions of blocks.
Step3: Extract structural blocks in the code clone using the information of location of clone pairs and semantics of blocks. Intuitively, structural block indicates the part of code clone that is easy to move and merge.

CCShaper performs Step 2 and Step 3. CCShaper extracts the following kinds of code clone as a refactoring-oriented code clone.

`Declaration` : class { }, interface { }
`Method` : method body, constructor, static initializer
`Statement` : if-statement, for-statement, while-statement, do-statement, switch-statement, try-statement, synchronized-statement
`Block` : range surrounded with '{' and '}'

3.2 Application of Refactoring Patterns

Here, we explain how these refactoring patterns are applied to the extracted code clones by CCShaper. At first, we have to say that there are two types of

code clone in which we are interested. One is “method-unit clone”, ant the other is “statement-unit code clone”.

For example, if all fragments of a given clone class are in the same class and the type is statement-unit, we can extract the clone statements as a new private method. In the other case, if all classes which have some fragments of a given clone class succeed to the same parent class and the type is method-unit, pulling up these fragments to the common parent class removes the code clones. In the similar case, if all classes which have some fragments of a given clone class succeed to the same parent class and type is statement-unit, each fragment could be extracted as a new method by applying “Extract Method”, and in addition, each new method could be pulled up to the common parent class.

4 Case Study

4.1 Target Software

In order to evaluate the usefulness of the proposed method, we have applied it to a famous Java software:ANTLR(Version 2.7.1)[1]. ANTLR(ANother Tool for Language Recognition) is a language tool that provides a framework for constructing recognizers, compilers, and translators from grammatical descriptions containing C++ or Java actions. ANTLR includes 189 files and the size is 42000LOC.

4.2 Code Clone Analysis Environment: Gemini

In this case study, we used Gemini[16], which is graphical code clone analysis environment. Gemini uses CCFinder as a code clone detector, and greatly helps user to analysis code clone and modify source code. Currently, the function of CCShaper is included in Gemini, and the three steps of CCShaper(written in 3.1) are performed automatically. The followings explain some functions of Gemini, which were used in this case study.

Scatter Plot. Figure 6 shows an example of scatter plot. Both the vertical and horizontal axes represent code portions of source files. The following two sequences are used as sample code portions in the scatter plot.

code portion X: “ABCDCDEFBCDG”,
code portion Y: “ABCEFBCDEBCD”

Here, symbols “A”, “B”, “C”,. . . are code portions in an unit such as character, token, line, statement, function, etc (In Gemini, it is token). In Figure 6, each small black square means that corresponding two elements on the two axes are the same. So, a clone pair is shown as a diagonal line segment. If the same code portions are arranged on the two axes, naturally, a diagonal line from the upper left to the lower right is drawn since such dot means comparison of token with itself, and the dots are symmetrical with a diagonal line.

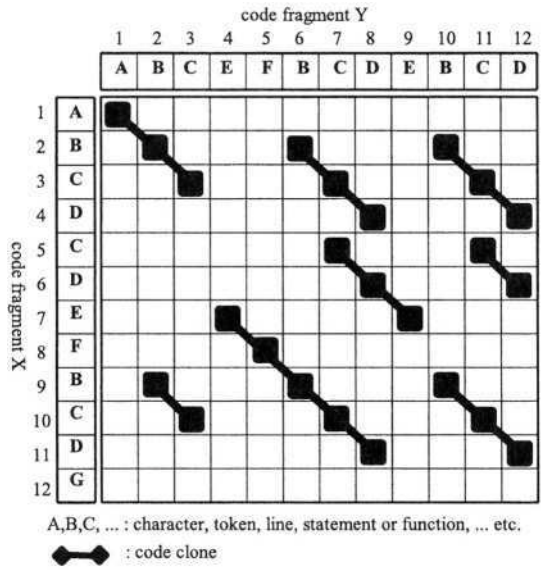


Fig. 6. Scatter Plot Model

The state of distribution of code clone can be grasped at a glance. However, as for large scale software in which there are many code clones, it is very difficult to decide which plot (that is code clone) in the huge scatter plot should be kept our eyes on. That is, if many files are located on the axis of coordinate in naive order, such as alphabetical order with file name, the distribution of code clones is occasionally spread widely without conspicuous deviation. So, Gemini has the function to sort the order of files on the two axes. It causes code clones not to distribute all over a scatter plot as much as possible. As a basic idea, the more code clones exist among two source files, the nearer the files are to be located in each axis. The details are described in [16].

Metric Graph. Figure 7 shows the model of metric graph. A polygonal line is drawn per a clone class. Besides, each of five vertical bars represents the each metric. The followings are the metrics.

RAD: Represent the range of the source code fragments of a code clone in the directory hierarchy, when the source code is supposed to be stored in the hierarchical directory. When all the code fragments of a clone class are located in one source file, the RAD value of the clone class is equivalent to 0. When the code fragments of the clone class are located in multiple source files stored in one directory, the RAD is 1. If those sources files are stored in different directories, then RAD is the maximum depth of those sources measured from their common parent director.

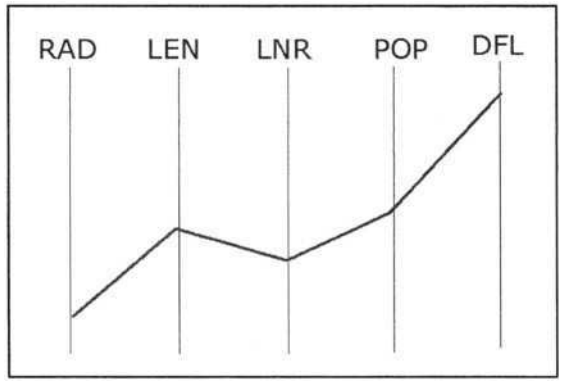


Fig. 7. Metric Graph Model

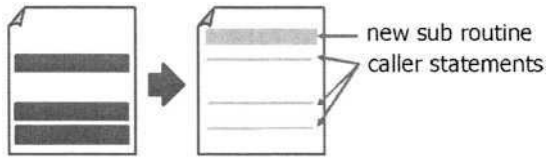


Fig. 8. Replace cloned portions with new identical routine

LEN: Represent the maximum length of element(code fragment) in a given clone class.
LNR: Represent the number of tokens without repeated part of the clone code fragments. each LNR is the same or smaller than LEN.
POP: Represent the number of element(code fragment) for a given clone class. If this value is high, it means that the similar code fragment appears in many places.
DFL: Represent an estimation of how many tokens would be removed from source files when all element(code fragment) of a given clone class are replaced with caller statements of a new identical routine like figure 8.

Metric graph allows user to set upper and lower bound on each metric, which enables user to select arbitrary clone class which he or she is interested in.

Source Code View. User can browse source codes of code clones which are selected in scatter plot or metric graph. In this view, code clones are highlighted, so he or she can easily recognize the range of them.

4.3 Refactoring Activity

First, we have applied CCFinder to ANTLR. In this case study, we specified 50 as the length of minimum clone of CCFinder and 30 as the length of minimum

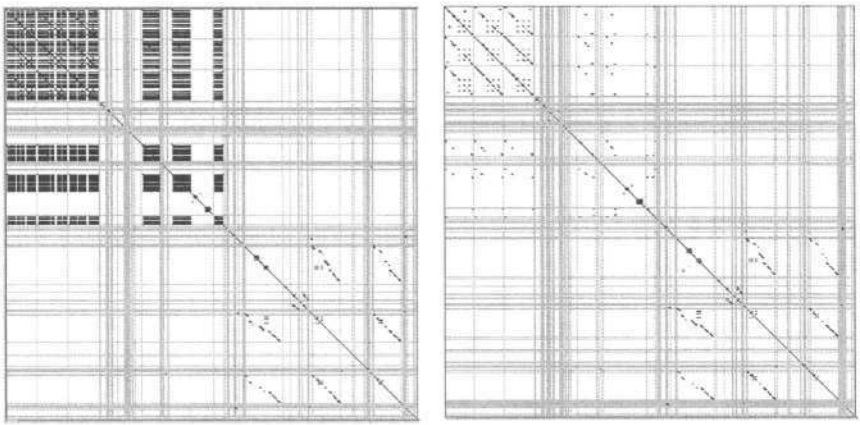


Fig. 9. Comparing on scatter plot

block of CCShaper. The result of scatter plot is shown in Figure 9(a). We can see there are many code clones in ANTLR. Totally, there are 276 clone classes and 13290 clone pairs.

Then, we extracted several code clones using CCShaper and investigated their appropriateness for refactoring. In this selection, we used metric graph. As a result, we identified the following two types of such code clones.

A clone class (C1), which was one of the types, included 24 code fragments. Each code fragment of C1 included 82 tokens and implemented the same algorithm to parse different lexical entities, such as the comma, the semi-colon, or the operators. All code fragments of (C1) were methods, and 17 code fragments of them were in same class. So, we applied "Extract Method" pattern [7] to (C1) and merged the 17 code fragments into a single method. Figure 10 shows some of the code fragments of C1 and Figure 11 shows the merged method.

As for the second type, a clone class (C2) included 150 code fragments and each of them included 33 tokens. All code fragments of (C2) were if-statements, and appeared in three classes. Furthermore these three classes inherited the same class as its parent. So, we firstly applied "Extract Method" pattern to (C2), which extracted the cloned portions as new methods. And then, we applied "Pull Up Method" to the new method. As the result, cloned if-statements are pulled up to common parent class as a new identical method shown in Figure 12.

The refactoring process to (C1) and (C2) resulted in 1000LOC reduction of the source code.

```
    public final void mBANG(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = BANG;
        int _saveIndex;

        match('!');
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
            _token = makeToken(_ttype);
            _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mSEMI(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = SEMI;
        int _saveIndex;

        match(';');
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
            _token = makeToken(_ttype);
            _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mCOMMA(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = COMMA;
        int _saveIndex;

        match(',');
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
            _token = makeToken(_ttype);
            _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mRCURLY(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = RCURLY;
```

Fig. 10. Source code of C1

Finally, we have to confirm that the functionality is not changed by the above refactoring process to ANTLR. We have checked the behavior of ANTLR after refactoring using all sample programs included in ANTLR package. For the 84 sample programs, the outputs from ANTLR before and after refactoring are exactly the same ones.

Figure 9(b) shows the scatter plot of ANTLR after refactoring. You can see that most of the clones located on the upper left side of Figure 9(a) have been removed.

5 Related Works

For the purpose of procedure extraction, code clone detection method for semantically cohesive ones using PDG(program dependence graph) have been proposed[12][14]. Also, Baxter et al. proposed a method to detect code clones using control/data flow dependencies from AST(abstract syntax trees) [3]. However, it is difficult to apply their approach to large scale softwares since the cost to create PDG/AST is very high.

Other interested approach is proposed by Balazinska et al[5]. They find function level clones by using 21 metrics[8], and apply different analysis[4][13] and context analysis[5] to detected clones, which means an estimation of refactoring applicability for each clone.

```
public final void ExtractedMethod(boolean _createToken, int _new_int, char _new_char)
        throws RecognitionException, CharStreamException, TokenStreamException {

        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = _new_int;
        int _saveIndex;

        match(_new_char);
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
}
```

Fig. 11. Extracted method for C1

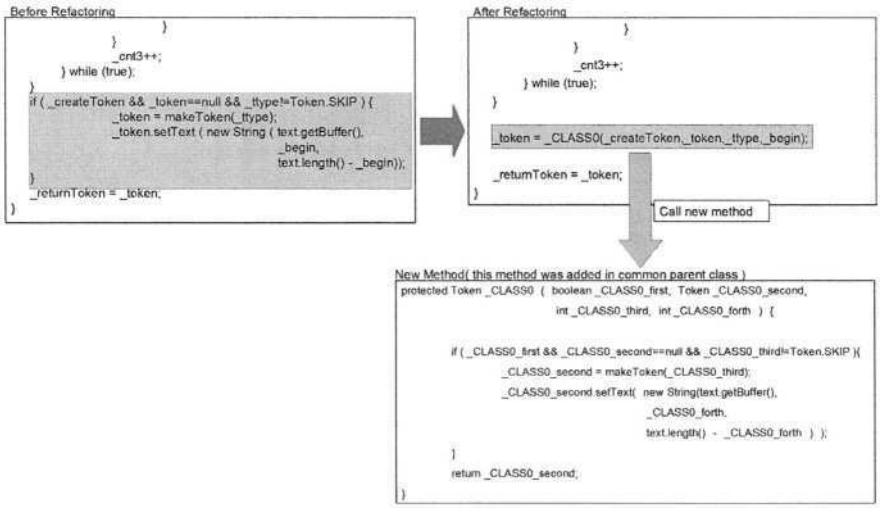


Fig. 12. Refactoring to C2

6 Conclusion

We have applied CCShaper with CCFinder to a practical Java software ANTLR. We found two clone classes that .refactoring patterns can be applied and actually conducted refactoring to them. As the result, we could reduce the code size by 1000 LOC without changing its original functionality.

But, since the current approach just extracts structural code clones from detected ones by CCFinder, it does not guarantee that all extracted structural code clones can be removed. So, as one of the future works, we will perform two types of analyses to get more precise result. One is to analyze variables which are referred in the code clone portions. For example, if all referred variables are declared in the same code clone portion, the portion can be easily moved to the

parent class and so on. Otherwise, some referred variables are declared outside of the code clone portion, it is difficult to move the cloned portion to other class.

The other is to analyze the portion relationship between code fragments belonging to same clone class in class hierarchy. For example, all code fragments of a given clone class are in the plural classes, if these classes inherit the same parent class, this clone class may be removed. But, if they don't have common parent class, it is difficult to remove them. We consider that above two types of analyses can help us to perform refactoring.

It is also necessary to evaluate the refactoring effect[11] after detecting the bad-smell part in the actual refactoring process. Without knowing the effect, we cannot judge whether we should go for refactoring or not because we have to be cost sensitive. We are going to examine it in the refactoring process based on the code clone information.

References

1. ANTLR, http://www.antlr.org/, (2000).
2. B. S. Baker: "Parameterized Duplication in Strings: Algorithms and an Application to Software Maintenance," SIAM Journal on Computing, 26, 5, pp. 1343-1362 (1997).
3. I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier: "Clone Detection Using Abstract Syntax Trees," Proc. of ICSM98, pp. 368-377 (1998).
4. M. Balazinska, E. Merlo, M.Dagenais, B. Lagu, and K. Kontogiannis: "Measuring clone based reengineering opportunities," Proc. of METRICS99, pp. 292-303 (1999).
5. M. Balazinska, E. Merlo, M. Dagenais, B. Lagu, and K. Kontogiannis: "Advanced Clone-Analysis to Support Object-Oriented System Refactoring," Proc. of WCRE2000, pp. 98-107 (2000)
6. S. Ducasse , M. Rieger, and S. Demeyer: "A Language Independent Approach for Detecting Duplicated Code," Proc. of ICSM99, pp. 109-118 (1999).
7. M. Fowler: Refactoring: improving the design of existing code, Addison Wesley (1999).
8. J. Mayland, C. Leblanc, and E. Merlo: "Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics," Proc. of ICSE96, pp. 244-253 (1996).
9. IEEE Std 1219-1992: Standard for Software Maintenance, (1992).
10. T. Kamiya, S. Kusumoto, and K. Inoue: "CCFinder: A multi-linguistic token-based code clone detection system for large scale source code," IEEE Trans. on Software Engineering, 28, 7, pp. 654-670 (2002).
11. Y. Kataoka, T. Imai, H. Andou and T. Fukaya: "A quantitative evaluation of maintainability enhancement by refactoring," Proc. of ICSM2002, pp. 576-585 (2002).
12. R. Komondoor and S. Horwitz: "Using slicing to identify duplication in source code", Proc. of the 8th International Symposium on Static Analysis, pp. 40-56 (2001).
13. K. Kontogiannis, R. DeMori, E. Merlo, M. Galler, and M.Bernstein: "Pattern Matching Techniques for Clone Detection," Journal of Automated Software Engineering, Kluwer Academic Publishers, Vol. 3. pp.77-108, 1996.

14. Jens Krinke: "Identifying Similar Code with Program Dependence Graphs", Proc. of the 8th Working Conference on Reverse Engineering, pp. 562-584(2001).
15. Y. Higo, Y. Ueda, T. Kamiya, S. Kusumoto and K. Inoue: "On software maintenance process improvement based on code clone analysis," Proc. of Profes 2002, pp. 185-197 (2002).
16. Y. Ueda, T. Kamiya, S. Kusumoto, K. Inoue, *Gemini: Maintenance Support Environment Based on Code Clone Analy sis,* 8th International Symposium on Software Metrics, June 4-7, 2002.

Introducing the Next Generation of Software Inspection Tools

Henrik Hedberg

Department of Information Processing Science, University of Oulu
P.O. BOX 3000, FIN-90014 University of Oulu, Finland
henrik.hedberg@iki.fi

Abstract. The area of tool support for software inspection has been under active research since the early 1990's. Although numerous implementations exist and development is still taking place, no tool has achieved a break-through. The main reason is that one tool usually demonstrates only one new idea, neglecting other features. A different approach must be taken, and software inspection tools should be seen as integral parts of the development environment. This paper categorises the existing tools into four generations based on the transition from traditional meeting support to asynchronous distributed inspections implemented with web technologies. Based on the analysis of 16 tools and our experiences, we summarize the most important features and add two new aspects to be notified when implementing the next generation of inspection tools for use in modern software development, flexibility and integration. The major focus is on comprehension, and we have taken the first steps at achieving this.

1 Introduction

Software inspection [1] is a well known, acknowledged and simple but disciplined method for locating defects in artefacts produced at any phase of software development. There are still surprisingly few organizations that take advantage of this, however, possibly on account of wrong picture that inspections would be uneconomical and not suitable for modern distributed development. When using software inspection tools, routine tasks can be automated and meetings can be distributed or removed, thus facilitating adoption of the inspection process.
Tool support for software inspection has been under development since the early 1990's, and significant progress has taken place, but new tools are still being introduced. The aim of this development was first to support distributed working and then to change the nature of the process to one that is totally independent of time and place by providing the means to inspect documents and participate in discussions in a distributed and asynchronous manner. The focus more recently has been largely on implementation issues.

Although this area has been studied intensively and numerous implementations exist, no tool has achieved a break-through. It seems that researchers have usually developed their own tools merely to demonstrate and evaluate a single theory, and little attention has been paid to the rest of the features. Actually, a review of tool

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 234–247, 2004.

support for software inspection by Macdonald et al. [2] in 1995 already indicated the same symptoms. In addition, the distribution aspect has become more and more relevant in software development, and the need for tools is now greater than ever. By inventing the wheel again and again, inspection tool research has failed to respond to the growing demands. It is obvious that a different approach to tool research should be adopted.

This paper presents a categorisation of existing tools in terms of generations and gathers together results that could be used as a basis for research and development aimed at new tools. The analysis is based on a review of the literature concerning the most significant tools available to date. By combining this with our own experiences, we arrive at a list of the most important aspects that should be taken into account when implementing the next generation of inspection tools. These ideas are related to our previous work on Virtual Software Inspection [3,4].

Chapter 2 briefly describes the defining features of software inspection tools, Chapter 3 presents the existing four generations of tools and Chapter 4 introduces the next generation. Chapter 5 is discussion, and the last chapter presents our conclusions.

2 Software Inspection Tools

The purpose of software inspection tools is to support software inspections as originally defined by Fagan [1]. The inspection process consists of six phases: planning, overview meeting, preparation, inspection meeting, rework, and follow-up. The most obvious task to automate is data collection, including the logging of defects and calculation of data, as is mainly done at inspection meetings and in the follow-up phase. If the participants are geographically scattered, material sharing is the next important feature. A sophisticated tool should nevertheless provide support for all tasks.

Macdonald et al. [2] have identified the features that are suitable for tool support. These are document handling, individual preparation, meeting support, and data collection. The focus was still on traditional Fagan-style inspection, with synchronous meetings enhanced with distributed meeting support to allow a geographically scattered team to co-operate. Based on the analysis of five implementations, they concluded that a successful inspection tool should incorporate support for all types of document, including diagrams, the linking of annotations to the parts of the document to which they refer, checklists and the compilation of metrics, which could also be used to ensure that each inspector is adequately prepared. In addition, automatic defect detection and polls were suggested, but strictly speaking these are not part of an inspection.

Extending the view of Macdonald, Tenhunen and Sajaniemi [5] added support for the process and interfaces as important viewpoints in software inspection tool support. Both these frameworks can be used to evaluate and develop inspection tools. Neither Macdonald et al. in 1995 nor Tenhunen and Sajaniemi in 2002 could find any tool that possessed all the above features.

Ellis et al. [6] classified groupware, i.e. software intended to support group work, into four categories based on the characteristics of distribution and synchronisation. This is an important viewpoint for software inspection tools, too, since the original inspection process specifies, for example, that preparations should be made privately

at the best time for each participant but that the findings should be collected together at a joint meeting. There are many variations on the software inspection process, however, and tools have added their own characteristics to it. One objective is to make inspections totally meetingless, in order to alleviate resource and timing problems [7]. The original inspection contained meetings and the first tools were developed to support these, but since then support for distributed working has been developed, and finally participation with total freedom of time and place. This shift of emphasis is depicted in Figure 1.

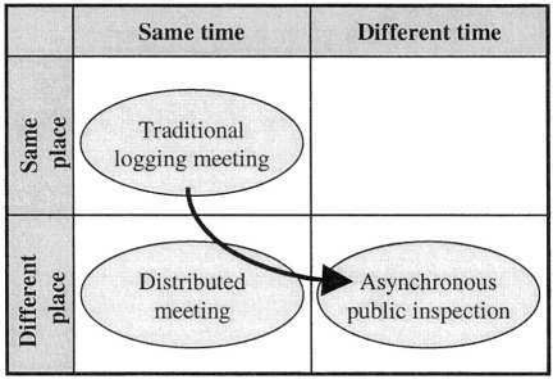


Fig. 1. Evolution of tools support from synchronous same place meetings to asynchronous different place working.

The inspection tool is a special case of groupware intended to support a strictly defined process handling a predefined set of material. There are also studies that have adapted generic groupware such as electronic meeting systems [8] for inspection purposes. By comparison with actual inspection tools, generic software does not provide support for the process, or for specific characteristics such as checklists or detailed metrics. Group work aspects should be taken into account, but specialized support must be incorporated, too. Data collection tools are also used in many companies, but these are not considered in the present paper. Some of them are commercial and some developed in-house, but they are rarely documented outside. Although mentioned as inspection tools and being important for supporting inspections, those tools do not cover the whole process and thus they are not strictly speaking software inspection tools.

3 Existing Generations of Tools

Tool support for software inspection evolved in the 1990's. The first inspection tools were aimed at helping to collect issues, and soon tools were being developed to support distributed working. Later researchers have improved these tools by adding other valuable features. Major evolutionary steps occurred during this period.

Based on the classification of groupware [6] and technological transitions, four generations of software inspection tools can be identified. These are:

- Early tools
- Distributed tools
- Asynchronous tools
- Web-based tools

The first three generations follow directly the degrees of freedom in the classification of groupware. The first transition is from early same-place, same-time tools to distributed but still same-time tools. Total independence of time and place was implemented in the third generation, asynchronous tools. Web-based tools are still asynchronous, but are distinguished from the previous generations by their technological base. The generations are described in more detail in the following subsections.

3.1 Early Tools

One of the first inspection tools was ICICLE (Intelligent Code Inspection Environment in a C Language Environment) [9]. This was intended to be used when inspecting source code. The authors identified preparation and the logging meeting as the most difficult and time-consuming phases. ICICLE uses the X Window System and provides for both private preparation and a shared view of the same material. The meeting had to be carried out in one room, because a separate voice communication channel was needed.

The main idea of InspeQ (Inspecting software in phases to ensure quality) [10] was to support phased inspections, although the tool was not limited to this. The material under inspection was plain text, and inspectors made free-form comments. The tool also provides checklists and windows for related standards. No form of meeting was possible, and files used and generated by the tool had to be transferred by e-mail, for example.

The early tools launched the development of software inspection tools, and although the main purpose was to support data collection, checklists and standards were incorporated as well and some kind of meeting support was considered.

3.2 Distributed Tools

Distribution set out from the idea that inspectors should be able to participate in an inspection meeting using their own computers. The first tool to make this geographically dispersed work possible was Scrutiny [11], which combined both synchronous and asynchronous phases. After the leader had planned the inspection, the inspectors went through the material privately. The authors also put forward the idea of a public inspection phase, in which comments made by inspectors could be shared by participants, and integration into other software development environments was also pursued. This could have enabled continuous fetching of the newest versions of the material, and also the exporting of defects to an error tracking system.

Tool support for the whole distributed inspection process was implemented in CSI (Collaborative Software Inspection) [12], which enabled both the material and the

defects found in it to be shared. In addition to asynchronous phases, the process also contained a synchronous logging meeting with video conferencing support.

The first distributed tools contained both asynchronous and synchronous phases, requiring the participants to meet at some time. Distribution was based on the Unix environment, or else the material was transferred between users in plain files. The idea of integration with other software development tools was first presented at this stage, as well.

3.3 Asynchronous Tools

The change to pure asynchronous inspections without a synchronous meeting took place before the mid-1990's, and research interest in this area also increased at this time. The result was that at least five totally different new inspection tool prototypes are documented in the literature.

One of the first tools implementing a totally asynchronous inspection process was CAIS (Collaborative asynchronous inspection of software) [13], in 1994. This was a successor of the CSI tool. Strictly speaking, the process was not Fagan-style inspection, but contained an additional problem resolution phase in the logging meeting. The discussion needed to accomplish this often lasted several days and participants would come back again and again to comment on each other's ideas.

The CSRS (Collaborative Software Review System) [14] presented the idea of a configurable inspection process, as originally used in the totally asynchronous FTArm (Formal, Technical Asynchronous review method) process, with both private preparation and public inspection meeting phases. In addition, special attention was paid to metrics collection. Defect logs with timestamps, for example, could be analysed in a variety of ways to produce empirical data as a basis for research and process improvement. The purpose was not only to support paper-based inspection, but mostly to focus on the benefits afforded by tool support.

The idea of inspecting documents that also contained graphics was first presented in TAMMi [15]. The tool followed the metaphor of correcting paper documents by marking a suspect item with a vertical line in the margin. This was essential in order to enable the inspection of design documents, for example, since earlier tools were limited to plain text. Although this was a distributed tool that supported an individual preparation phase, document and process management had to be implemented separately, e.g. by e-mail.

ASSIST (Asynchronous / Synchronous Software Inspection Support Tool) [16, 17] made it possible to adapt a tool to the needs of an organisation. This was achieved by using the IPDL (Inspection Process Definition Language) to model process phases. Extensive metrics collection facilities also existed, and the tool calculated estimates to support inspections and exit criteria, for example. One participant could control everybody's line cursor during the group work, but unfortunately the tool could handle only plain text documents.

InspectA [18] used e-mail as its distribution technology. The authors explained the choice by advancing the hypothesis that a view shared by means of the web could be detrimental to the motivation of slower inspectors. The importance of source documents was emphasised, but the tool still supported only textual documents. The process began with individual preparation, followed by defect log construction.

Before the document was inspected once again, the participants could check the defect list and discuss it asynchronously.

The legacy of this era is certainly important. The idea of asynchronous meetings and a configurable process must be acknowledged when defining the key features of software inspection tools. In addition, the importance of documents containing graphics was demonstrated.

3.4 Web-Based Tools

The beginning of the web era marked a significant transition for software inspection tools as well, as it became possible to implement tools that were totally independent of time and place. Web-based tools could be developed using existing structures and were easy to employ.

AISA (Asynchronous Inspector of Software Artefacts) [19] was one of the first web-based software inspection tools. The material under inspection is textual, with the possibility of contain some graphics, and it is published in the dedicated web site divided into pages, like one class in a class diagram. Comments are connected into pages. After the preparation phase, the author removes duplicates and the participants can then discuss possible solutions for the defects they have found. The tool collects metrics such as the number of comments and the duration of sessions.

HyperCode [20] is targeted for use in code inspections. The material is delivered via the web, and inspectors can comment on it on a line by line basis. The process contains only asynchronous phases, and other participants are able to see the comments, so there is no logging meeting. WiP (Web inspection Prototype) [21] is very similar to HyperCode but is implemented as a Java applet. After individual inspection, the results are collected together into a report containing links from the original material to the comments.

New tools are also available that are based on web technologies. WiT (Web inspection Tool) [22] and its lightweight version In Win (Inspection Window) [7] use the side note as a metaphor for issue marking. The process consists of two phases, individual and public inspections. Although the tool presents material in the form of web pages, the linking of related material is not possible.

IBIS (Internet-Based Inspection System) [23] is a lightweight web-based inspection tool that uses the RealServer/RealPlayer G2 software to provide audio and possible video connections between inspectors. During the distributed meeting, material is presented with Microsoft PowerPoint and participants' views are synchronised with the RealPresenter G2. Web-IPSE (Web-based Inspection Process Support Environment) [24], which is intended for educational purposes, provides a means for monitoring several groups and facilitates reinspection by linking comments to artefacts in a version management system.

The web-based tools mostly stabilized the technologies used to implement software inspection tools, following the general trend from desktop to web-based applications. Although the basic idea of the web is hypertext, hyperlinking capabilities are not actually used in the tools. Links are only seen as an enabling technology for annotations, but the documents under inspection cannot usually contain links. AISA is the only exception from this pattern, because the material is divided into multiple pages that are linked together. This is only an implementation issue, however, and is not used for reference to related material.

3.5 Summary

The four generations of software inspection tools have shifted the focus of tool support from traditional meeting support to asynchronous distributed inspections implemented with web technologies. The tools are presented on a time-scale by generations in Figure 2. Examination on this evolution shows that once a new method or implementation has proved its feasibility, it has been adopted and research has been focused on further development.

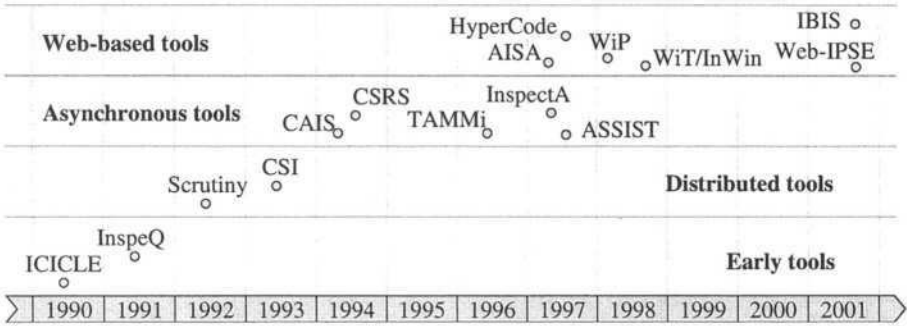


Fig. 2. Four generations of software inspection tools.

The most significant improvements took place during the transitions to distributed and asynchronous tools, for once the basic set of features, document management, annotation handling, checklists and metrics, had become established, research mostly became stuck on implementation issues. Although these are interesting and important technologically, the new tools have unfortunately provided practically nothing new for software inspection itself. Tool research has not adopted the new ideas of inspections generally, such as perspective-based reading [25].

Being limited by implementation issues, inspection tools usually support only the inspection of code or documents containing text. Although the real benefits of inspections can be appreciated best by inspecting design documents, only a couple of tools can handle both text and graphics such as figures, tables and charts. Most of the tools focus on the individual preparation phase, and a couple of them provide support for public inspection or a virtual logging meeting. Every tool contains administrative tasks that the leader must perform at the beginning, and the resulting data consist mostly of comments used to correct the document, whereas more sophisticated metrics gain little attention.

Macdonald et al. [2] summarized the state of inspection tools in 1995 by claiming that "Each has its own unique approach and unique features. However, no single system exists that provides all of the features outlined above, which are necessary to effectively support the entire inspection process." The area of software inspection tool support is still subject to active research and the new tools are making use of modern technologies provided by the current web infrastructure. Unfortunately, after all these years and new tools, the situation is still the same. Researchers have usually developed their own tools in order to demonstrate and evaluate a single theory, but there is no tool that supports the whole inspection process. As software development has changed and new technologies make it possible to resolve the issues that arise,

there is still a need to fit the pieces together to form a comprehensive software inspection tool.

4 The Next Generation of Tools

The next generation of software inspection tools must embrace the support of software inspection as a whole. This means that all the important features already found that have proved feasible should be taken into account when developing new tools. The next generation could be referred to as *"comprehensive tools",* because the main objective will be to combine the important features into a single tool or platform.

The basic idea of the inspection tool is to help inspectors to handle *artefacts* based on a predefined *process* that yields annotations and metrics as *results.* The resulting data can be used not only when correcting the original target, but also when analysing the inspection process itself. It is not obvious, however, whether current inspection tools can conveniently provide support for these functions in various situations, e.g. when inspecting UML charts or analysing the overall performance of the inspection process.

To meet the actual needs of modern software development, we bring in two new aspects of software inspection tools: *flexibility* and *integration.* We claim that by implementing these, the tool can conform to the requirements of real users. We will attempt next to give a detailed description of what these aspects mean in the context of artefacts, the process and the results.

4.1 Artefacts

The target of software inspection is an artefact, such as a requirements document, a design document, or source code. Plain text is the simplest form of documentation, and is sufficient for code inspections, but design documents also contain figures and tables, for example. Document management has been a major feature of tools since the early days, but still more attention should be given to the flexibility aspect in order to enable the inspection of various types of document.

To ensure effortless administration of the tool and the inspection, the tool must be aware of how the artefacts have been produced, where their revisions are located and how they can be accessed. If the material has to be copied into a tool repository, there is a risk that the wrong version may be inspected or that two copies of the artefact will begin to evolve concurrently. Artefact management is thus an obvious example of the need for integration.

4.2 Process

Classical Fagan inspection with both asynchronous and synchronous aspects is not the only process model used in industry. Rather than a fixed process model, inspection tools should provide capabilities for customizing the process for an individual organization or project. One process model will not fit for every organisation or

situation. The workflow of the tool has to be flexible enough to support different types of inspection while still being rigorous. In this way small, busy projects can employ pair inspection [22], for example instead of traditional inspection.

There should not be any need to copy or generate new workflow data for the tool, as the data should already be written into the project plan in the form of tasks and resources (people and time). If the tool can utilize the project management data, administration tasks will be minimized and the inspection leader will be able to focus on the inspection itself. The inspections are sometimes marked as milestones, however, and not the tasks, which means that those should be performed in zero time. By insisting that the inspection is marked as a task with adequate start and stop points, the tool also forces an improvement in the overall process. In addition, information goes in the other direction, too, because the project plan has to be updated as the inspection tasks are completed. Inspections usually produce new action points (for reworking), which it would be convenient to read off directly from the inspection tool database and insert into the task list in the project management tool.

4.3 Results

The comments written down by the participants during the inspection should also contain information about the severity of errors, their classification, and references to checklist items. In addition, Gilb and Graham [26] list 42 metrics that should be collected during the inspection, and 14 variables that should be calculated. An inspection tool should unquestionably support these metrics and automate the collection, storing and analysing of the necessary data.

Most of the current inspection tools operate with a copy or some kind of image of the original data. The results should be linked to the original document, however, in order to enable seamless analysis and to ensure that all the corrections will be made. To encourage improvements in the process, it must be possible to calculate the basic metrics automatically, and the set of metrics must be flexible enough to focus on the most important aspects in certain situations.

In addition to the built-in metrics, the raw data must be available for further analysis. Manual transfers are in any case laborious and unnecessary, as well as being error prone. At the same time, they can cause the metrics to be buried somewhere in the tool database, because the time taken to handle the data is not thought valuable enough. Thus the inspection tool should be integrated into generic repositories so as to build up an organisation-wide knowledge base. The results should also be used to improve the inspection process itself.

4.4 Summary

Finally, the flexibility aspect must be extended to integration issues, too. The tool should not guess anything about the degree of integration or the repositories used in the organization. This level of independence is hard to implement, but development should be focused on structures that can be effortlessly adapted to the organization's needs.

The relations between the most important features of software inspection tool support are summarized in Figure 3, which points to three characteristics related to inspections: artefacts, results, and the process. The outermost columns and rows summarize the integration and flexibility aspects.

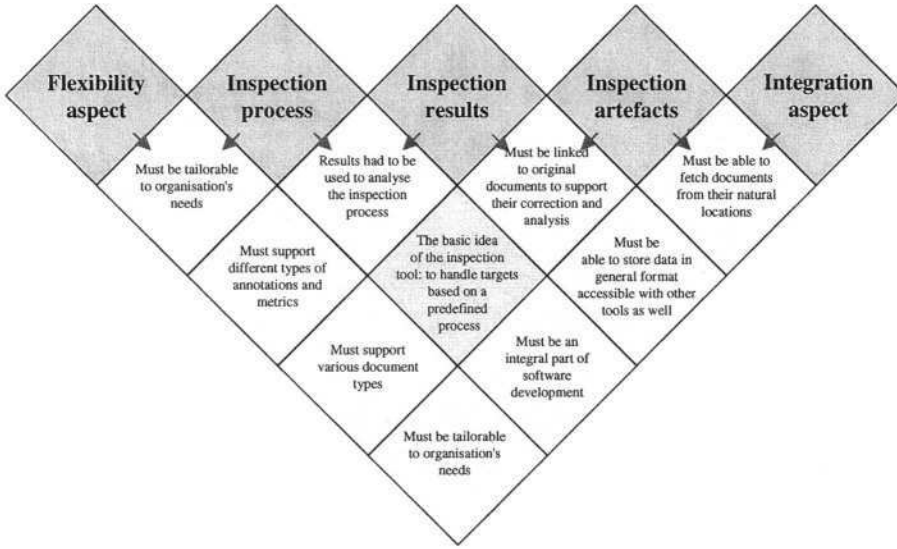


Fig. 3. Interrelationships between the three basic aspects of software inspection tools and two new aspects.

Our vision of the next generation of tools may be crystallised in the definition of Virtual software inspection [3]. The process can be lightweight or all-inclusive, comprising both asynchronous and synchronous phases. It will usually be carried out using network techniques, but traditional meetings can be included if necessary. Distribution will be carried out with tools that enable efficient running of the process. Independence of time and place, on-line recording of issues and data management can be achieved through networked tools. One important aspect will be flexibility, covering both the process and supporting tools, to ensure tolerable levels of adoption effort and acceptance. In addition, interoperability of the processes and tools will be required to enable convenient everyday use of the method and to improves the effectiveness of inspections. These characteristics should make inspections more attractive and widely used by easing the implementation effort required.

5 Discussion

Distributed software engineering projects cannot make use of the same traditional methods as do co-located teams, although their communication and quality assurance

needs are the same. This is especially apparent in software inspections that have a heavy accent on teamwork. Although tools are enablers for distributed team working, those are identified as a main problem in development [27]. Achieving a uniform tool environment could be difficult because of the cultural differences and various tools used in separate offices.

Interoperability is the key for this problem. However, it should be seen largely, as process integration merely than only technological issue. Inspection as a team work can be a method to share ideas, knowledge and work between team members in various locations. It can function as a glue to bind together process pieces scattered around the world.

As Porter and Johnson state, face-to-face meetings do not make the defect detection process significantly less effective [28], and several other sources [29, 30] show that there should be no substantial differences in efficiency between traditional and computer-supported inspections. Thus, if tools are implemented properly, distributed teams lost nothing as compared with co-located teams.

To demonstrate and evaluate the flexibility and interoperability aspects, we have implemented a new annotation tool, XATI (XML Annotation Tool for Inspection), placing emphasis on ease of use and deployment. XATI is based on the analysis of existing tools and experiences gathered during the development and use of our earlier tools, such as WiP, WiT and SATI (Site Annotation Tool for Inspection).

We started with a student experiment that focused on the usefulness and ease of use of the XATI tool. According to Davis, these are the key factors that affects to the adoption of a new technology [31]. The results were encouraging, as the students felt XATI to be both useful and easy to use. An obvious majority preferred it to paper-based forms for performing inspections and predicted that they would use it on a regular basis in the future if available.

Although the concept of virtual software inspection was formulated in co-operation with our partner companies, we are transferring the generalized results back to the industry. The next step will be to evaluate the XATI tool and the concept of Virtual Software Inspection in industry projects to obtain feedback from real users. We are especially interested in how feasible the aspects of virtual inspections are and whether we have succeeded in implementing them in our tool. We thus aim to seek answers to at least the following questions:

- Are the virtual inspection process and tools supporting it as flexible as they should be? Especially, what changes should be made to the processes when using our tool, or what changes should be made to the tool if following the original processes at the organization?
- Is the organization able to make efficient use of the interoperability of the processes and tools? Especially, where are the bottlenecks when creating seamless operation between inspection and other phases?

These questions should raise some important factors or limitations in the deployment and usage of computer-supported inspections. While inspection is only one part of the larger software development cycle, as a strictly defined process it is a good object of research and can give hints to the research of distributed software development in general.

6 Conclusion

Inspection tool research has produced numerous tools over a period of more than a decade, but no tool has been widely adopted for practical use or contains all the important features that have proved feasible. This is largely a consequence of the fact that the purpose of developing the tools has been to test single ideas, and little attention has been paid to their completeness.

Analysis of the inspection tools documented in the literature from the early days to the newest implementations enables us to identify four generations: early tools providing the means for collecting data, distributed tools enabling co-operation between geographically scattered inspectors, asynchronous tools that increased this freedom even further by eliminating meetings and finally web-based tools that have stabilised the technologies used in implementation by following the general trend towards web-based applications. The most important transition of all was to eliminate meeting with asynchronous tools.

Development and research must not become stuck on implementation issues only, or focus on single aspects of a tool. What is really needed is an implementation that acknowledges the legacy of previous generations and combines all the well-known features. This could be called the generation of comprehensive tools.

In addition to the key characteristics of inspection tools, such as material handling, process support and data collection, two new aspects must be taken into account: integration, implying that inspection should be an integral part of software development and that the tool should rely on existing document and project management systems, as well as flexibility, implying that the inspection tool should support documents, annotations and metrics of different types and that both the process and the level of integration must be tailorable to the organisation's needs. The tool should share the resulting data, for example, so that other tools can access items if needed.

The XATI tool, as of the first stage in an extensible inspection tool platform, shows that it is possible to construct a comprehensive tool. Its flexibility and integration with other software development tools and the fact that those cover both the data and the process are in our experience promising features. We really believe that these characteristics will make software inspection tools more attractive and widely used by facilitating their implementation and reducing the effort required to adopt them.

Acknowledgements. This work is being supported by the Academy of Finland under grant 74780.

References

1. Fagan, M.: Design and code inspection to reduce errors in program development. IBM System Journal, Vol. 5., no. 3. (1976) 182-211
2. Macdonald, F., Miller, J., Brooks, A., Roper, M. & Wood, M.: A review of tool support for software inspection. In: Seventh International Workshop on Computer-Aided Software Engineering (1995) 340-349

3. Harjumaa, L., Hedberg, H. & Tervonen, I.: A Path to Virtual Software Inspection. In: Proceedings of Procediings of Asia-Pacific Conference Quality Software (APAQS) (2001) 283-287
4. Hedberg, H. & Harjumaa, L.: Virtual Software Inspections for Distributed Software Engineering Projects. In: Proceedings of the Internation Workshop on Global Software Engineering, International Conference on Software Engineering (ICSE) (2002)
5. Tenhunen, V. & Sajaniemi, J.: An Evaluation of Inspection Automation Tools. In: Kontio, J. & Conradi, R. (eds.) Proceedings of Qality Connection - 7th European Conference on Software Quality (ECSQ) (2002) 351-361
6. Ellis, C., Gibbs, S. & Rein, G.: Groupware - Some issues and experiennces. Communications of the ACM, Vol. 34, no. 1 (1991) 38-58
7. Tervonen, I., Harjumaa, L. & Iisakka, J.: The Virtual Logging Meeting: a web-based solution to resource problems in software inspection. In: Proceedings of the Sixth European Conference on Software Quality. ADV Handlesgesellschaft m.b.H., (1999) 342-351
8. Genuchten, M., Cornelissen, W. & Dijk, C.: Supporting Inspections With an Electronic Meeting System. Journal of Management Information Systems, Vol. 14, no. 3., (1997) 165-178
9. Brothers, L.R., Sembugamoorthy, V. & Muller, M.: ICICLE: Groupware for code inspections. In: Proceedings of the 1990 ACM Conference on Computer Supported Cooperative Work (1990) 169-181
10. Knight, J. C. & Meyers, E. A.: An improved inspection technique. Communications of the ACM, Vol. 36, no. 11 (1993) 51-61
11. Gintell, J., Arnold, J. Houde, M., Kruszelnicki, J., McKenney, R. & Memmi, G.: Scrutiny: A collaborative inspection and review system. In: Proceedings of the Fourth European Software Engineering Conference (1993)
12. Mashayekhi, V & Drake, J., Tsai, W.-T. & Reidl, J.: Distributed, collaborative software inspection. IEEE Software, Vol. 10 no. 5 (1993) 66-75
13. Mashayekhi, V., Feulner, C. & Reidl, J.: CAIS: Collaborative Asynchronous Inspection of Software. In: Proceedings of the Second ACM SIGSOFT Symposium on the Foundations of Software Engineering (1994)
14. Johnson, P.: An instrumented approach to improving software quality through formal technical review. In Proceedings of the 16th International Conference on Software Engineering (1994)
15. Putaala, M. & Tervonen, I.: Inspecting Postscript Documents in an Object-Oriented Environment. In Proceedings of the 5th European Conference on Software Quality - Additional Papers, Dublin, Ireland (1996)
16. Macdonald, F. & Miller, J.: A Comparison of Tool-based and Paper-based Software Inspection. Empirical Software Engineering, no. 3 (1997)
17. Miller, J. & Macdonald, F.: ASSISTing exit decisions in software inspection. In: Proceedings of the 13th IEEE International Conference on Automated Software Engineering (1998) 281-284
18. Murphy, P. & Miller, J.: A process for asynchronous software inspection. In: Proceedings of the 8th International Workshop on Software Technology and Engineering Practice (1997) 96-104
19. Stein, M., Riedl, J., Harner, S. & Mashayekhi, V.: A Case Study of Distributed, Asynchronous Software Inspection. In: Proceedings of International Conference on Software Engineering (ICSE) 97, ACM (1997) 107-117
20. Perpich, J., Perry, D., Porter, A., Votta, L. & Wade, M.: Anywhere, Anytime Code Inspections: Using the Web to Remove Inspection Bottlenecs in Large-Scale Software Development. In: Proceedings of the 1997 International Conference on Software Engineering. IEEE Computer Society (1997) 14-21

21. Harjumaa, L. & Tervonen, I.: A WWW-based tool for software inspection. In: Proceedings of the Thirty-First Hawaii International Conference on System Sciences. Vol. 3 (1998) 379-388
22. Tervonen, I., Iisakka, J. & Harjumaa L.: Software Inspection - a blend of discipline and flexibility. In: Kusters R.J., Cowderoy A., Heemstra F.J. & Trienekens J.J.M. (eds.) Project Control for 2000 and beyond (Proceedings of ESCOM-ENCRESS'98), Maastricht: Shaker Publishing B.V. (1998) 157-166
23. Caivano, D., Lanubile, F., Visaggio, G.: Scaling up Distributed Software Inspections. In Maurer, F., Dellen, B., Grund, J., Kötting, B. (eds.): 4th ICSE (International Conference on Software Engineering) Workshop on Software Engineering over the Internet (2001) 5-8
24. Hazeyama, A. & Nakano, A.: Web-based Inspection Process Support Environment for Software Engineering Education. In Maurer, F., Dellen, B., Grund, J., Kötting, B. (eds.): 4th ICSE (International Conference on Software Engineering) Workshop on Software Engineering over the Internet (2001) 13-18
25. Basili, V.R., Green, S., Laitenberger, O. Lanubile, F., Shull, F., Sormgård, S. Zelkowitz, M.V.: The Empirical Investigation of Perspectiove-based Reading. Journal of Empirical Software Engineering 2 (1) (1996) 133-164
26. Gilb, T. & Graham D.: Software Inspection. Essex, England: Addison Wesley Longman Limited (1993)
27. Komi-Sirviö, S. & Tihinen, M.: Great Challenges and Opportunities of Distributed Software Development - An Industrial Survey. In: The 15th International Conference on Software Engineering and Knowledge Engineering (SEKE'03), San Francisco Bay, USA (2003)
28. Porter, A.A. & Johnson, P.M.: Assessing software review meetings: Results of a comparative analysis of two experimental studies. IEEE Transactions on Software Engineering, vol 23, no 3 (1997) 129-145
29. Johnson, P.M. & Tjahjono D.: Does every inspection really need a meeting? Empirical Sotware Engineering, no 3 (1998) 9-35
30. Laitenberger, O. & Dryer, H.M.: Evaluating the usefulness and the ease of use of a web-based inspection data collection tool. In: Proceedings of the Fifth International Software Metrics Symposioum (1998) 122-132
31. Davis, F.: Perceived Usefulness, Perceiced Ease of Use and User Acceptance of Information Technology. MIS Quarterly, vol. 13, no 3 (1989) 318-340

Intelligent Support for Software Release Planning

Amandeep[1], Günther Ruhe[1] , and Mark Stanford[2]

[1] University of Calgary, 2500 University Drive NW,
Calgary AB T2N 1N4, Canada
{amandeep, ruhe}@cpsc.ucalgary.ca
[2] iGrafx, Corel Inc., 7585 S.W. Mohawk St., Tualatin,
Oregon USA 97062
mark.stanford@corel.com

Abstract. One of the most prominent issues involved in incremental software development is to decide upon the most appropriate software release plans taking into account all explicit and implicit objectives and constraints. Such decisions have become even more complicated in the presence of large number of stakeholders such as different groups of users, managers, or developers. However, early involvement of customers and understanding of their real needs is one of the core success factors of software business [16].
This paper introduces a six step process model for release planning. It is inspired by the Quality Improvement Paradigm [2], as release planning is a learning and improvement process as well. Emphasis is on proposing the tool support implementing this process. The use of the intelligent decision support tool ReleasePlanner™ is presented by comparing a baseline scenario reflecting current state-of-the practice of release planning with a supposed improvement scenario obtained after usage of the tool. Initial experience from a real-world environment at iGrafx Corel Inc. is used to validate the improvement scenario.

1 Introduction

Software development has become a highly complicated and critical activity because of the increasing market demand, extensive use of high technology and increasing awareness for the use of software products. Unfortunately, the software engineering practices have failed to keep pace with such a tremendous progress. Intelligent decision support is required for the software systems related decisions to be based not only on the human judgment but also on sound models, knowledge, and methodology [11].

Requirements engineering is one of the key activities is software engineering. It is a multi-disciplinary and human centered process of discovering the purpose of the intended software product, identifying different stakeholders and their needs, documenting the requirements in an appropriate manner, communicating them and subsequently implementing them [10]. It is a decision-centric process [1]. Unfortunately, this process is not understood enough and is not given its due importance. The situation is worsened in today's dynamic incremental software development environment where the requirements are not static but are constantly changing as the project progresses [5].

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 248–262, 2004.

One of the most prominent issues involved in incremental software development is to decide upon the most appropriate software release plans taking into account all explicit and implicit objectives and constraints. Amongst the informal approaches claiming to handle this issue, one of the most well known is "planning games" used in agile development [9]. The goal of this approach is to deliver maximum value to the customer in least time possible. In half to one day long sessions, customers write story cards describing the features they want, while developers assign their estimates to those features. The customers then choose the most promising story cards for the next increment by either setting a release date and adding the cards until the estimated total matches the release date, or selecting the highest value cards first and setting the release date based on the estimates given on them.

This simplistic approach works well in smaller project. However, as the size and the complexity of the projects increases, the decisions involved in release planning become very complex. Various factors come into play, such as the presence of stakeholders having different relative importance and different preferences, the presence of constraints about sequencing and coupling of requirements in various increments, and the need to take into account resource allocation issue for implementing the requirements. Considering problems involving several hundreds of requirements and large number of widely scattered stakeholders, it becomes very hard to find appropriate solutions without intelligent support tools. The goal of such support is to account for all these factors in order to come up with a set of most promising release plans.

Carlshamre [4] provides a tool support for release planning by just concentrating on the model ended up being an overly simplistic approach due to the wicked nature of the problem. Realizing this, we provide a process centric approach considering different stakeholders involved in this task. In this paper, we present a comprehensive environment for intelligent support for software release planning. The technical details of this work are described in [5], [13], and [14].

The emphasis of this paper is to present a six step Quality Improvement Paradigm (QIP) based process model for release planning. In addition to that, the paper describes tool support implementing this process and initial experience on using the tool. In Section 2, we briefly introduce the problem and present main ideas of the core solution approach called EVOLVE*. Section 3 explains the process of release planning from the perspective of systematic learning and improvement following the QIP [2]. In Section 4, the actual use of our new tool ReleasePlanner™ is presented by comparing a hypothetical baseline situation reflecting current problems in release planning with the usage of our intelligent tool. Initial experience from a real-world environment (Corel Inc.) is presented as well. Finally, Section 5 presents the summary and the future work.

2 Problem Statement and Solution Approach

Release planning for incremental software development includes the assignment of requirements to releases such that all technical, risk, resource, and budget constraints are fulfilled. The overall goal is to find an assignment of requirements to increments that maximizes the sum of all (weighted) priorities of all the different stakeholders (see [5], [13], [14]).

The question of what actually constitutes a good release plan needs careful consideration. Intuitively, we are expecting most important and most urgent requirements first. However, 'most important' might mean different things for different types of stakeholder. The user is expecting features that he or she would need first to get started. But there are different types of users such as novice, advanced and expert users having different types of expectations and preferences. The actual challenge is to balance all those competing perspectives and to customize the objective function to the real user needs.

Effectively solving the problem of release planning involves satisfying the needs of a diverse group of stakeholders. Stakeholder examples are:

- Sales representative
- User (novice, advanced, expert)
- Investor
- Shareholder
- Project manager
- Product manager
- Developer

Understandings of who the stakeholders are and their particular needs are key elements in developing an effective solution for the software release planning problem. Typically, stakeholders will have different perspectives on the problem and different needs that must be addressed by the release plans. According to their main interest, they will focus more on quality, time, or business value.

Release planning is a highly constrained process. Typically, we encounter technological constraints expressed as dependencies between requirements. In addition to that, effort and resource constraints are addressing limitations on the amount of requirements that can be assigned to a certain release. In addition to effort, (bottleneck) resources are considered whenever specific tasks can be performed by specific types of resources only. For details on these modeling aspects we refer to [14].

As a result of the planning process, different increments will be composed out of the given set of requirements. These increments are planned up-front allowing for the possibility of re-planning after any increment. This re-planning may involve changing some requirements, priorities and constraints and/or introducing new ones. It necessitates a reassignment of requirements (not already implemented in former releases) to increments.

The proposed solution approach called EVOLVE* [14] combines the computational strength of genetic algorithms with the flexibility of an iterative solution method. At all iterations, a genetic algorithm is applied to determine most promising solutions of constrained release planning.

EVOLVE* differs from previous prioritization techniques [7], [8], [17] in its focus on the incremental software process. Main strengths of EVOLVE* are:

- It takes into account stakeholder priorities as well as effort constraints for all releases;

- Consideration of inherent precedence, coupling and resource constraints;
- Ability to easily handle changes to requirements and the project attributes;
- It caters for conflicts in stakeholders' priorities while recognizing that stakeholders opinions are not always equal; and
- It generates a set of most promising solutions (instead of just one solution).

EVOLVE* is an evolutionary approach. At iteration k, a decision is made about the next immediate increment $\mathbf{Inc}^{k}$. In addition, a solution is proposed for all subsequent increments $\mathbf{Inc}^{k+1}$ and $\mathbf{Inc}^{k+2}$. The reason for the iterative part in EVOLVE* is to allow all kinds of late changes in requirements, prioritization of requirements by stakeholders, effort estimation for all requirements, effort constraints, precedence and coupling constraints as well as changes in the weights assigned to stakeholders. This new information is used as an input to iteration k+1 to determine the next increment $\mathbf{Inc}^{k+1}$ as final and all subsequent ones $\mathbf{Inc}^{k+2}$ and $\mathbf{Inc}^{k+3}$. This approach is illustrated in Figure 1.

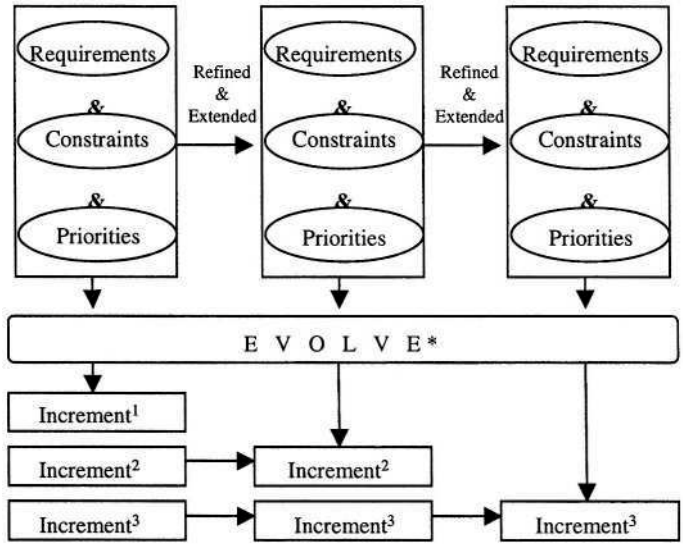


Fig. 1. EVOLVE* approach to assign requirements to increments

Solutions generated by EVOLVE* are optimal or near optimal. Genetic algorithms, in general, cannot guarantee optimality. There is a great variance in the solutions generated by different runs of the solution algorithm. This variation concerns both changes in mutation and crossover rate. However, as known from empirical evaluations, there is a great likelihood to have the optimal solution among the ones generated, especially if a larger number of computations with varying parameters are conducted.

Maximization of the objective function (defined based on the client needs) is the main purpose of conducting crossover and mutation operations. This function is computed as a fitness value for each optimization step of the genetic algorithm. The

algorithm terminates when there is no further improvement in the solution. This is calculated as no improvement in the objective function value achieved within 0.5% deviation over six hundred simulations. The algorithm works by producing orderings of requirements and, if valid, calculating this fitness score.

For each solution generated by the genetic algorithm, each of the constraints is checked. The effort constraint is handled by a greedy-like increment allocation algorithm, which is applied on every ordering produced by the genetic algorithm. This can be discrete effort estimation or, alternatively, the model allows for the possibility that effort estimates are represented by probability distributions.

3 The Process of Release Planning from a QIP Perspective

Our process of release planning is based on the ideas of constant learning and improvement as manifested in the notion of a learning software organization [12]. The Quality Improvement Paradigm (QIP) is a methodological framework for goal-oriented systematic improvement. It was originally developed for (but is by no means restricted to) software engineering projects and organizations [2]. Since release planning strives for learning and improvement at the project level as well as at the organizational level, the QIP can very well guide the activities and goals of release planning. QIP was designed to explicitly identify improvement goals, to capture relevant knowledge, and to evaluate and systematically reuse that knowledge. Therefore, a strong goal-orientation of the entire process as well as quality control of the captured knowledge is an important built-in feature of the QIP.

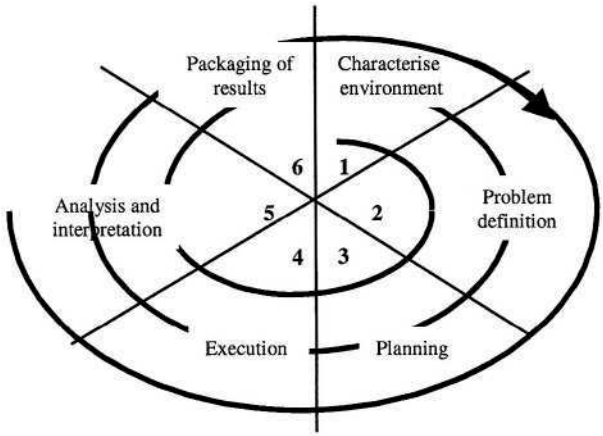


Fig. 2. Release planning cycle

The underlying process of using ReleasePlanner™ can be considered from the perspective of the QIP. Systematic learning and reuse of models and experience is an essential prerequisite for release planning. The spiral curve as shown in Figure 2 indicates the release planning methodology and the learning process between different and subsequent applications of this process. Higher stages of the spiral curve refer to higher maturity in this process. From QIP perspective, the release planning process

comprises six steps (Figure 2). In the following, we will explain these steps in more detail.

Step 1: Characterize and Understand

In order to build an efficient software system, it is imperative to characterize the project environment as well as the organizational environment in which the project is embedded. The project environment is influenced by many internal factors such as the type of the product and the process to be used, number and the quality of the professionals available, and many external factors such as market demand of the intended product. A comprehensive evaluation of these factors lays a solid foundation for a software development project.

The characteristics of the project at hand are studied and classified. This is necessary in order to find reusable experience from the knowledge base. One can retrieve similar product models from the knowledge base to support the project manager in making some realistic estimates about various process parameters. Some of the factors used while characterizing a project are: number, skills and availability of people involved, developed artifacts, size of the project, maturity of processes, and main quality criteria of the products.

Step 2: Problem Definition

After getting a reasonable understanding of the project context, the project manager describes the various parameters of the project. This task can be subdivided into five sub-tasks that are described in the following:

Step 2.1 Identify the Stakeholders

Stakeholders are the people who directly or indirectly, influence or are influenced by the software project plan. They may include developers, managers, customers etc. Identifying the stakeholders is a crucial step as they are the ones who set the track of the project and decide the evaluation criteria for its success. This task is performed essentially by the project manager.

Step 2.2 Elicit and Specify the Requirements

The task of eliciting, analyzing and managing the requirements is very tedious [17]. This is the inception phase of the project and there is high probability that the situation would be quite uncertain and dynamic. Stakeholder negotiations have to be instigated to increase the transparency and the understandability of the requirements. Some of the popular techniques used for this purpose are questionnaires, surveys, and interviews. Latest groupware systems based techniques such as Easy WinWin [6] also present a sound methodology for requirement elicitation and stakeholder negotiation [3].

Once the requirements have been specified in the tool, the project manager, with the consultation of the stakeholders, provides his effort estimates for the requirements. Furthermore, an estimated business impact is provided to distinguish between requirements (or feature) of high, medium, and low impact.

Step 2.3 Enter Preferences among Stakeholders
In an incrementally delivered system, one has to plan the increments very carefully so that the requirements from the most important stakeholders and the ones with the highest business values are implemented in the earlier increments. Thus, identifying the relative importance of the stakeholders and assigning them appropriate weights, is a crucial step. This task is essentially performed by the project manager.

Step 2.4 Enter Constraints
While deciding the release plans, the project manager has to consider various constraints regarding the structure of the increments such as precedence of some requirements to others (precedence constraint), coupling of some requirements with others (coupling constraint), or resource bottlenecks for some requirements (resource constraint). Currently the tool handles first two types of constraints. The project manager identifies these constraints in consultation with the stakeholders.

Step 2.5 Stakeholders Assign Priorities to the Requirements
This task is meant for the stakeholders. Each stakeholder has a separate login to access the system. Once logged in, they see the whole set of requirements specified by the project manager in Step 2.3. They can read the requirement specification, add their comments and assign the appropriate priority to them.

At the end, the project manager makes various estimates about the project such as number of increments required to deliver the full functionality and maximum effort available for various increments. These parameters are used later on when the ReleasePlanner™ is launched.

Step 3: Planning

Planning of the release planning process includes the following tasks:

- Link to other information systems (to retrieve existing data sources; link to project management and enterprise resource planning tools to synchronize proposed solutions).
- Planning scenarios (variation of parameters like effort, business values, capacity for releases, risk) to better reflect uncertainty of data.
- Planning of feedback session with stakeholders to discuss the proposed solutions (to be actually done in step 5).
- Usage of other tools (such as groupware support system for negotiation) to facilitate communication between stakeholders.
- Planning of a sequence of release planning steps (update of computations every week or so to reflect changing project parameters).

Step 4: Execution

After all the necessary information required for successful release planning is collected, ReleasePlanner™ is used to execute the process according to the plan. Often, in this step, experiments are performed; analyzing the solutions for a range of variation of parameters, and a set of solutions is generated. The project manager gets release plan proposals for all those parameters involved. This plan is then discussed with the stakeholders and after thorough negotiation the plan is either finalized or re-calculated by changing some of the involved parameters.

Step 5: Analyze Experience

The data are collected during the implementation of the tool, analyzed and evaluated to interpret the results. The analysis helps to characterize and to understand project performance. For example, such data may show that some of the stakeholders could not understand the requirements appropriately or the effort estimates for some requirements were not appropriate.

The feedback data from iteration are collected using some popular techniques such as questionnaires, interviews, discussions, and regular feedback sessions etc. and analyzed by comparing the results with the tentative goals for the increments.

Step 6: Package Experience and Results

At this step, the data generated from various tool runs is packaged for reuse. Such preparation would include documenting, explaining, structuring, and linking the gained information to the context where it was obtained. The experience is characterized and classified, documented, explained and linked to the existing models, knowledge, and packaged in the experience base. For additional understanding of the data, it can further be compared and linked to similar projects in the knowledge base. For example, before running a project for release planning, a project manager can check the experience base to see how similar projects were executed in the past. This would help him understand various patterns of variations in different project parameters that resulted in the most appropriate release plans.

4 Release Planning Scenarios

In this section, we discuss scenarios comparing a tentative baseline situation reflecting the current state-of-the practice in software release planning with the improved situation of using intelligent tool support. We present two scenarios. The first one describes a hypothetical baseline situation reflecting the current state-of-the practice in software release planning. The second scenario presents how our approach is supposed to alleviate those problems.

4.1 Scenario 1: Baseline

We assume a software organization that applies the incremental development paradigm to better react on flexible customer requirements. The main problems faced by the organization for incremental delivery of the software are enumerated as follows:

Size and complexity of the problem: One major problem the project manager faces while deciding the release plans is the size and the complexity of the problem. The number of requirements in some projects may go up to hundreds or even thousands. The situation is worsened by the fact that there are usually a large number of stakeholders involved in the project who are either not consulted at all or are not given appropriate relative weights. In addition to that, the stakeholders usually have conflicting interests and opinions about allocating the requirements to increments. The size and the complexity of the problem (known to be NP-complete) and the tendency for not involving all the contributing factors, makes the problem prohibitively difficult to be solved by individual judgment or trial and error type methods. Greedy-type heuristics are faster, but are not able to provide good enough solutions.

Changing requirements and other parameters: If a large number of requirements increase the complexity of the project, their dynamic nature poses another challenge. The requirements are always changing as the project progresses. In addition to that, other parameters such as number of stakeholders, their priorities etc. also change with time.

Requirements are not well specified and understood: There is usually no formal way to describe the requirements. Non-standard format of requirement specification often leads to incomplete description and makes it harder for stakeholders to understand the requirements properly.

Other constraints: The project manager has to consider various constraints while allocating the requirements to various increments. For example, due to some technical constraints or customer preferences, some requirements may have to be implemented in precedence to or in coupling with other requirements. Furthermore, the project manager has to allocate his limited resources efficiently to get the best results as soon as possible. The resources may act as bottleneck for the allocation of some requirements in the same increment.

Efficiency and effectiveness of release planning: Release plans have to be updated quite frequently due to changing project and organizational parameters. The effort for that and the quality of solutions generated ad hoc are far behind objective demands.

Transparency of solutions: It is not clear for all the involved stakeholders why certain plans were suggested. In case of conflicting priorities, it would be useful to know how certain compromises look alike and why they were taken.

No provision to learn from the experience: In an organization where there is no provision to learn from the experience gained from previous release plans, each project is one of its kind. Every time an increment has to be planned, the whole process is started from the scratch.

All these constraints add to the complexity of the problem at hand and if not handled properly, they create huge possibility of project failures. In the next section,

we show how our methodology backed by the tool support is supposed to alleviate this problem.

4.2 Scenario 2: Improvement by Intelligent Tool Support

In the second scenario, we illustrate the individual steps of release planning supported by the intelligent tool ReleasePlanner™. To make the benefits more transparent, we assume a sample project with of 25 requirements. Five stakeholders are involved in prioritizing the individual requirements. Furthermore, additional technological and resource constraints are assumed.

In our prototype tool, we have the provision to specify the requirements briefly in our main database and in detail using the standard templates for requirement specification (SRS: Software Requirement Specification) in the knowledge base. To achieve greater flexibility and objectivity in determining priorities between stakeholders, we assume the application of pair-wise comparison following the Analytical Hierarchy Process AHP [15].

To remove the bias and to save the stakeholders from external influences as much as possible, the tool provides separate logins for the project manager and the stakeholders. Hence, the stakeholders are not able to see the weights assigned to them by the project manager.

In what follows, we will illustrate the usage of ReleasePlanner™ by two screen shots. In Figure 3, the stakeholders have assigned the priorities to different requirements. The project manager logs in his account and assigns the weights to the stakeholders. The first row in Figure 4 also shows the stakeholder weights calculated using AHP. Moreover, the project manager enters the constraints such as precedence, coupling and resource constraints. After entering all this information, the project manager runs the data through the tool and gets result in terms of a proposed release plans.

Requirement	0.211	0.211	0.421	0.05	0.105
1: Should be able to save as PDF files	1	1	1	1	1
2: Improve display of data	3	1	2	2	1
3: The shape of parent should be linked directly to that of children	4	3	5	5	3
4: Export Table should export XML properties	3	5	3	5	5
5: Tables should have attaching ability	1	3	2	2	3
6: Table should be made on multiple diagrams	4	5	3	4	4
7: Redo the coding in Java and make some improvements	5	5	3	5	4
8: Table manuplation should be faster	5	5	5	3	4
9: Improvetitle creation methodology for projects	3	1	2	2	2
10: Output - Ctrl Click should open linked object	3	2	3	4	2
11: Large increments should be broken down into iterations	2	1	2	3	1
12: Publish to Web - improve publishing of documents.	4	3	2	3	4
13: Manage document templates properly	3	3	2	4	3
14: Tables should have fields for Department names	1	2	2	2	2
15: Table should support any type of diagram object	2	2	2	2	1
16: Clearly defined layout option	2	1	2	1	2
17: Field changes should be persistent	3	1	2	2	1
18: Improved accumulation of data in the table	3	2	1	4	3
19: Improved data formatting	2	4	3	2	4
20: Integrate the Chart editor into the Task Bar	5	2	1	1	1
21: Improved Histogram bars	1	5	3	4	4
22: Improve line graphs	5	5	5	5	5
23: Put symbols on pie slices	1	2	3	1	1
24: Report tool tips should work	1	2	3	4	5
25: There should be a provision to save in the word document	3	4	5	4	3

Fig. 3. Stakeholder priority table (Step 2.5)

Figure 4 shows three release plans produced by the tool for three different values of α, the parameter that in indicates whether the solution is more trying to maximize stakeholder benefit or to minimum penalty for them in case that the actual assignment of requirements is not in accordance to the assigned priority. For details we refer to [5].

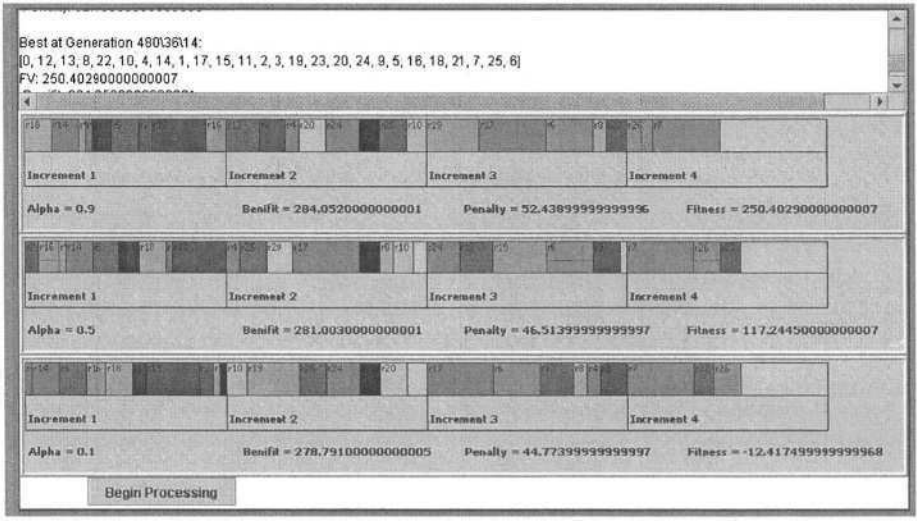


Fig. 4. Example release plan (Step 4)

In this way, ReleasePlanner™ takes into account various factors influencing a release plan and comes up with quality solutions. However, these solutions are by no means final; they just give the project manager the scenarios of what the release plan would look like for the particular values of the parameters he chose. He can change these values to get different scenarios. The final choice is made after evaluating several such scenarios.

To account for problems such as change in the requirements and volatility in other constraints of the project, this process is repeated whenever a new increment has to be planned. The benefits of using the above-mentioned approach are as follows:

Efficiency: It is clear from the example above that the effort to generate solutions related to their quality is much lower than that of when done manually or with any other method.

Effectiveness: Quality of solutions generated is near optimal and in general superior to any solution generated ad hoc.

Transparency: The underlying fitness score function of EVOLVE* guarantees optimal balancing between different stakeholder preferences, and this makes the proposal transparent.

Stability: The stability of the proposed solutions can be judged from the different runs of the tool (eventually, with varying crossover and mutation rates).

Flexibility: The parameters involved in deciding the release plans such as customer requirements, stakeholder preferences for the requirements etc. are dynamic in nature [5]. ReleasePlanner™ allows investigation of any change in requirements, priorities or

other involved parameters. This enables the generation of new solution alternatives. Furthermore, variations of the weighting parameter in the objective function result in offering a set of most promising candidate solutions.

Knowledge management support: In a system like this where the user background can vary from a very technical to a novice user, an experience repository supports storage of experiences from the varied set of stakeholders. With large amount of information involved in the system, such as various artifacts, product models produced after each run of the tool, and the lessons learned from these runs; the need of a good knowledge base support conies natural [2]. In the current version of the tool, we recognize the following three entities to be included in the knowledge base:

Document templates: These are the text documents provided to standardize the task of documentation for the artifacts used in the process. Our knowledge base contains a range of standard document templates used in the process and other information resources helpful for the users. Typical examples of such resources would be newsgroup/message boards/e-mail postings, and other technical issues.

Product models: A product model in our knowledge base contain the project specification for that product, the values of the various parameters involved in it and the results of ReleasePlanner™ tool runs while designing the release plans of the product. The analysis of the data collected from these models gives a general idea of how the system responds to a range of variation of the parameters involved in it.

Lessons learned: These are the documents containing the experience of the users in terms of lessons learned while using the tool in the past. We go a step further in it by generalizing the project specific lessons to process related lessons and suggesting the improvements in the approach based on them.

4.3 Release Planning for iGrafx (a Product Group of Corel Inc.)

Before using ReleasePlanner™, iGrafx releases were planned in an informal fashion using a Lotus Notes database of collated customer requirements. Records in the database were created for each distinct requirement received from customers (existing and prospective) and internal sources. In addition to title, description, and source, records included a priority ranking between 1 (high) and 4 (low). An effort field estimated the development time (in days) to complete requirements.

The Lotus Notes database was effective for providing a fast, simple, central storage of requirements available to program management (release planners) and software engineers (the developers). Lotus Notes was also good for creating a diverse set of reports and summaries to assist implementation.

The planning process, however, was mostly ad hoc. Program management scanned the requirements attempting to identify the most commonly requested feature requests to add to the next release. The development team would then estimate how many requirements could be completed in the time allocated and the product plan would gradually come together. This process had several difficulties:

- Many requirement descriptions were not fully described.
- Stakeholders (other than program management) had no synchronized involvement in ranking.

- Coupling and precedence relationships between requirements were not formally defined.
- For many requirements, the effort was not defined early enough in the requirement selection process.
- Without a clear prioritization process, it was hard to achieve buy-in on the product plan.
- It was difficult to change the product plan when effort estimates changed or new "must-have" requirements arrived.

4.3.1 An Improved Planning Process

These planning deficiencies led to the discovery of ReleasePlanner™. Program management was quickly excited about the possibilities these new tools provided but software engineering was initially skeptical. They believed that ReleasePlanner™ would only add unnecessary complexity to the release process. This negative perception has disappeared. Now there is universal agreement among the iGrafx team that ReleasePlanner™ is improving the iGrafx product planning process and ReleasePlanner™ will deliver long-term value.

To date, we've run tests on a subset (about 250 requirements ranked by three stakeholders) of our database and the results are encouraging. For the first time, we are defining a plan that looks three releases into the future instead of only one (our past practice). In addition, we feel that the contents of each release are maximizing benefits to the stakeholders. Soon, we will define releases using the full set of 800 requirements ranked by seven stakeholders.

Recently, other Corel product groups (e.g. XML Solutions and WordPerfect) have asked for information about ReleasePlanner™. They are keen to build on the experiences and success of the iGrafx team.

4.3.2 Use and Benefits of ReleasePlanner™

Our initial challenge was importing our Lotus Notes database into ReleasePlanner™. We concluded that we must retain our Notes database because the software engineers use it to track code changes, describe implementation plans, and chart development status of work-in-progress. Having two databases means that we must upload our Notes database periodically to ReleasePlanner™ and then rank new requirements before running the tool. Eventually, we'd also like to export the ReleasePlanner™ rankings to Lotus Notes.

We are confident that ReleasePlanner™ is helping us define a better product plan and providing the following benefits:

- Engineers are more careful to accurately estimate implementation effort. A good estimate is now required because the effort field directly determines release contents. In addition, by defining all precedence and couplings early in the planning process, the release contents are more certain and subject to fewer revisions.
- Requirement descriptions have improved. Stakeholders will not rank requirements with poorly written descriptions or will rank them lower. Poorly described requirements will probably not land in any release.

- Stakeholders are providing earlier and more valuable feedback. Before ReleasePlanner™, stakeholders were asked for advice after the initial product plan was announced. Now, they are involved earlier and are more likely to endorse final release plans.
- Program Management can more easily identify and develop release themes that can later be used by marketing.
- Customers can be given a "road-map" of future plans extending beyond the next release.

5 Summary and Future Work

We have presented a powerful approach for solving a broad class of software release planning problems for incremental software and product development. In addition to an innovative core algorithm that is based on evolutionary computing, we have developed process-centered and knowledge management support for using the tool. A six-step approach following the QIP is used to achieve that goal.

From comparison between a baseline scenario (ad hoc planning without tool support) and an improvement scenario using ReleasePlanner™, we have demonstrated the potential benefits of our environment. Initial experience at Corel Inc. confirms those hypotheses. Future developments and researches are directed into mainly three directions:

Enhancements of methodology EVOLVE*: Broader modeling scope by including risk and resource constraints as well as aspects of task scheduling.
Enhancements of the tool environment: To better support decision-making for release planning, we are working on an integration of ReleasePlanner™ with a knowledge management component that is supporting the structuring, organization, retrieval and maintenance of relevant knowledge and experience.
Validation: Controlled and field study experiments are planned to further validate the proposed hypotheses. This will result in knowledge about the expected Return-of-Investment and further feedback to improve the overall tool.

Acknowledgements. One of the authors would like to thank the Alberta Informatics Circle of Research Excellence (iCORE) for the financial support for this research. Many thanks are due to Des Greer, An Ngo-The and Sebastian Maurice for their collaboration and stimulating discussions. Kenny Tsang and Erik Bauld contributed to main parts of the implementation of EVOLVE*.

References

1. Aurum A., Wohlin C.: The Fundamental Nature of Requirement Engineering Activities as a Decision-Making Process. Information and Software Tech, Vol. 45 (2003), pp 945-954.
2. Basili, V., Caldiera, G., Rombach, D.: Experience Factory. In: J. Marciniak: Encyclopedia of Software Engineering. Vol 1, pp 511-519, 2001.

3. Boehm, B., Grunbacher, P., Briggs: Developing Groupware for Requirements Negotiation: Lessons learned. IEEE Software, May/June, pp 46-55., 2001.
4. Carlshamre, P.: Release Planning in Market-Driven Software Product Development: Provoking an Understanding, Requirement Engineering Vol 7 (2002), pp 139-151.
5. Greer D., Ruhe G.: Software Release Planning: An Evolutionary and Iterative Approach. Accepted for publication in Information and Software Technology, 2004.
6. Gruenbacher, P.: Collaborative Requirement Negotiation with Easy WinWin. IEEE Software, pp 954-958, 2000.
7. Karlsson, J, Wohlin, C and Regnell, B.: An evaluation of methods for prioritizing Software Requirements, Information and Software Technology, Vol 39 (1998), pp 939-947.
8. Karlsson, J.: Software requirements prioritizing, Proceedings of the Second International Conference on Requirements Engineering, pp 110 – 116, 1996.
9. Larman, C.: Agile & Iterative Development, A Manager's Guide. Addison-Wesley, 2003.
10. Nuseibeh, B., Easterbrook, S.: Requirements Engineering: a Roadmap. Proceedings of "The Future of Software Engineering", pp 35-46 May 2000.
11. Ruhe, G.: Software Engineering Decision Support: Methodology and Applications. Innovations in Decision Support Systems (Ed. By Tonfoni and Jain), International Series on Advanced Intelligence Vol 3, pp 143-174, 2003.
12. Ruhe, G.: Learning Software Organizations. In: Handbook of Software Engineering and Knowledge Engineering (S.K. Chang. Ed.), World Scientific Publishing 2001, Vol 1, pp 663-678.
13. Ruhe, G. and Greer, D.: Quantitative Studies in Software Release Planning under Risk and Resource Constraints. Proceedings of the IEEE International Symposium on Empirical Software Engineering (ISESE 2003), pp 262-271.
14. Ruhe, G., Ngo-The, A., Hybrid Intelligence in Software Release Planning. Appears in: IJHS, Vol 1 (2004).
15. Saaty, T.L.: The Analytical Hierarchy Process. Wiley, New York, 1980.
16. Standish Group Research: What are your requirements? http://www.standishgroup.com/
17. Wang, Q. and Lai, X.: Proc. Requirements Management for the Incremental Development Model. 2nd Asia-Pacific Conference on Quality Software, pp 295-301, 2001.

An Empirical Evaluation of Predicting Runaway Software Projects Using Bayesian Classification

Osamu Mizuno[1], Takanari Hamasaki[1], Yasunari Takagi[2], and Tohru Kikuno[1]

[1] Osaka University, 1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan,
o-mizuno@ist.osaka-u.ac.jp
[2] OMRON Corporation, 2-2-1 Nishikusatsu, Kusatsu, Shiga 525-0035, Japan
yasunari_takagi@omron.co.jp

Abstract. Since software development projects often fall into runaway situations, detecting signs of runaway status in early stage of development has become important. In this paper, we propose a new scheme for the prediction of runaway projects based on an empirical questionnaire. We first design a questionnaire from five viewpoints within the projects: requirements, estimations, planning, team organization, and project management activities. Each of these viewpoints consists of questions in which experience and knowledge of software risks are included. Secondly, we classify projects into "runaway" and "success" using resultant metrics data. We then analyze the relationship between responses to the questionnaire and the runaway status of projects by the Bayesian classification. The experimental result using actual project data shows that 33 out of 40 projects were predicted correctly. As a result, we confirm that the prediction of runaway projects is successful.

1 Introduction

Recently, software development projects are required to produce highly reliable systems within a short period and at low cost. In other words, software development projects have been put in a very risky situation. Thus, detecting signs of problems at an early stage of the software project is important. If the detection of a problem is delayed, it becomes more difficult to fix a problem since the effort of coping with a problem increases exponentially [1].

Much research has been carried out about the detection of problem signs of a software development project [2, 3]. Concerns for risk management are increasing for early detection of such problem signs in software projects. The Software Risk Evaluation method (SRE) is a risk-management technique for a software development project [4]. In the SRE, the project's risks are identified using the taxonomy table of software risks. The risk taxonomy table is very useful for systematically identifying risks of a project. However, since many risk attributes exists in the identification of risks using the taxonomy table, the extraction of a risk takes time. Therefore, SRE recommends to carrying out the tailoring of the taxonomy table for each project.

Risks of a software development project are influenced by environments such as the domain, the business style, the culture of the organization, and by human characteristics. In the projects with similar environments, an approach to prevent recurrence of problems

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 263–273, 2004.

by analyzing past problems is usually taken. Such an approach is easily understood by the project members, since it is based on the actually occurring problem. However, if a problem factor is not arranged appropriately, the number of factors will increase, and handling will become difficult. Efficiency is most important in a project; therefore, software must be developed in a short period of time and at the lowest possible cost. Such efficiency also holds true for project management activity.

This study shows an empirical approach to predict "runaway" projects in an organization. The proposed approach has the following features:

Use of a questionnaire: We utilize a simple questionnaire to identify the characteristics of projects.

Empirical evaluation: We validate our constructed model with actual project data empirically.

Firstly, we investigate the problems on various project using the questionnaire. In developing our questionnaire, we made use of lessons learned from past experience of our projects, and related works in the literature [5,6,7,8]. From the results of the questionnaire and past project performance, we identified "runaway" projects and their problems.

Secondly, we analyze the relationship between the responses of the questionnaire and the results of projects using the naive Bayesian classification approach. The naive Bayesian classification is a well-known approach in the data mining, and it is widely used to classify data into several discrete categories. We therefore apply the naive Bayesian classifier to the collected data of the questionnaire in order to classify them into "runaway" and "success"projects.

Finally, we evaluate this approach by the 10-fold cross validation technique. The results of this evaluation sufficiently confirmed the validity of our approach.

2 Background and Objective

The main products of Social Systems Business Company (SSBC) in OMRON consist of embedded software in ticket vending machines, automated teller machines, and in point of sales systems. Such systems for different customers use the same type of hardware and operating systems, but incorporate specific customer requirements, such as customer-specific user interfaces, printed forms, and operation sequences. The development process for these projects is the overlapping waterfall model [9].

In order to promote a process improvement initiative, the Software Engineering Process Group (SEPG) was established in 1992. Various Software Process Improvement (SPI) activities were undertaken by the SEPG, such as;

- Collecting and analyzing a project's actual result data such as quality, cost, and duration
- Monitoring of project status and troubleshooting support
- Developing process standards and procedures in the organization
- Delivering training of the organizational standard process
- Facilitating software process improvement initiatives throughout the organization

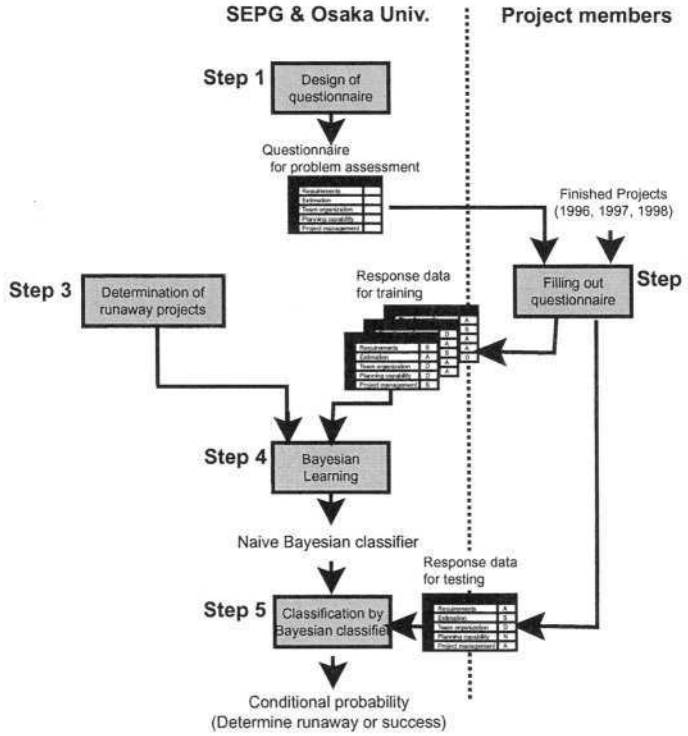


Fig. 1. Outline of prediction of runaway projects

For this reason, the number of runaway projects has decreased yearly. However, the runaway projects still occur every year although the number of them is quite small. In order to reduce such runaway projects, the SEPG has tried to analyze their characteristics. If such characteristics are identified, we can predict runaway-prone projects (also called "risky") and we can deal with such projects before they turn to runaway projects. A similar type of project has been dubbed the *death march project* [10].

In this paper, a "runaway project" is defined as follows:

- A project whose cost and duration is out of a certain range, and,
- A project falling into an uncontrollable situation during development.

3 Outline of Our Approach

Figure 1 shows the outline of our approach for predicting runaway projects. First, in Step 1, we designed a questionnaire to be distributed to project managers and leaders in order to collect the assessment data. Fortunately, in SSBC, actual resultant data and lessons learned for every development project have been stored. The questionnaire consists of

five viewpoints each of which is further divided into several risk factors. Next, in Step 2, SEPG distributed the questionnaire to project managers and leaders, and asked them to fill out the questionnaire. After they finished filling out the questionnaire, the SEPG collected them. At the same time, in Step 3, SEPG determined the runaway projects from available project data. Here, we assume that the final status of a project becomes either "runaway" or "success". In Step 4, we apply the naive Bayesian learning to the responses to the questionnaire that is prepared for the model learning, and we obtain the probabilistic model to identify the runaway projects. Finally in Step 5, using the obtained model, we predict the runaway project based on the calculated probabilities.

The questionnaire consists of five viewpoints: requirements, estimations, planning capability, team organization, and project management activities. (The details of the design will be described in Section 5.) Each sub-item regarding risk factors in the questionnaire must be filled in according to the Likert scale [11]: "Strongly Agree", "Agree", "Neither agree nor disagree", or "Disagree".

The naive Bayesian classification is the optimal method of supervised learning if the values of the attributes of an example are independent given the class of the example. Although this assumption is almost always violated in practice, recent research has shown that naive Bayesian learning is also effective in practice.

In our previous research [12], we constructed a logistic model with five candidates used for the parameters. In this paper, we change the modeling technique with more generic model, that is the naive Bayesian classifier, since it is more robust approach to apply more empirical data from various industries.

4 Naive Bayesian Classifier

The naive Bayesian classifier is one of the most common approach to classify categorical data into several classes. We introduce several fundamental concepts briefly.

4.1 Bayesian Learning

Let $Q_1, Q_2, \cdots, Q_n$ be the parameters with discrete values to predict a discrete class C. For example, Q_i denotes the questionnaire and C denotes the final status of a project: runaway and success. Suppose that the values $q_1, q_2, \cdots, q_n$ are given to these parameters, and the optimal prediction is the class value C = runaway such that $P(C = \text{runaway} | Q_1 = q_1 \wedge Q_2 = q_2 \wedge \cdots \wedge Q_n = q_n)$ is maximum. By the Bayes' theorem, this probability is expressed as follows:

$$\frac{P(Q_1 = q_1 \wedge Q_2 = q_2 \wedge \cdots \wedge Q_n = q_n | C = \text{runaway})}{P(Q_1 = q_1 \wedge Q_2 = q_2 \wedge \cdots \wedge Q_n = q_n)} \times P(C = \text{runaway}).$$

The probability $P(C$ = runaway) can be easily estimated from training data. Furthermore, $P(Q_1 = q_1 \wedge Q_2 = q_2 \wedge \cdots \wedge Q_n = q_n)$ is irrelevant to the class variable C. Therefore, learning is reduced to the problem of estimating $P(Q_1 = q_1 \wedge Q_2 =$

$q_2 \wedge \cdots \wedge Q_n = q_n | C = \text{runaway})$ from training data. Using Bayes' theorem again, this conditional probability can be written as follows:

$$P(Q_1 = q_1 | Q_2 = q_2 \wedge \cdots \wedge Q_n = q_n, C = \text{runaway}) \times P(Q_2 = q_2 \wedge \cdots \wedge Q_n = q_n | C = \text{runaway})$$

The second factor of above formula is recursively formulated as follows:

$$P(Q_2 = q_2 | Q_3 = q_3 \wedge \cdots \wedge Q_n = q_n, C = \text{runaway}) \times P(Q_3 = q_3 \wedge \cdots \wedge Q_n = q_n | C = \text{runaway})$$

Here, we assume that Q_i's $(1 \leq i \leq n)$ are independent each other. In other words, we assume

$$P(Q_1 = q_1 | Q_2 = q_2 \wedge \cdots \wedge Q_n = q_n, C = \text{runaway}) = P(Q_1 = q_1 | C = \text{runaway})$$

and so on. Then, $P(Q_1 = q_1 \wedge Q_2 = q_2 \wedge \cdots \wedge Q_n = q_n | C = \text{runaway})$ equals

$$P(Q_1 = q_1 | C = \text{runaway}) \times P(Q_2 = q_2 | C = \text{runaway}) \times \cdots \times P(Q_n = q_n | C = \text{runaway}).$$

$P(Q_i | C = \text{runaway})$ $(1 \leq i \leq n)$ can be estimated from training data. The process of Bayesian learning is thus performed.

4.2 Bayesian Classification

Using the result of learning, we can classify and predict the value of C if the values of q_i are given.

$$P(C = \text{runaway} | Q_1 = q_1 \wedge \cdots \wedge Q_n = q_n) = (\prod_{i=1}^{n} P(Q_i = q_i | C = \text{runaway})) \times P(C = \text{runaway})/z$$

where z is a normalizing constant. We can thus calculate the conditional probability for any given values of parameters q_i's using this equation. From the calculated probability, we classify the data into either class, runaway or success. Here, we determine a project is in class "runaway" if $P(C = \text{runaway} | Q_1 = q_1 \wedge \cdots \wedge Q_n = q_n) \geq 0.5$.

5 Design of the Questionnaire

5.1 Five Viewpoints

In this study, we have investigated various works [5,6,7,8] regarding risk management and the experience of the SSBC. Based on the results of this investigation, we have summarized all key risk factors and classified them into the following five viewpoints: (1) Requirements, (2) Estimations, (3) Team organization, (4) Planning capability, and (5) Project management activities. The overview of the questionnaire is shown in Table1.

Table 1. Problem Assessment Questionnaire

1. **Requirements**	
1.1	Ambiguous requirements.
1.2	Insufficient explanation of the requirements.
1.3	Misunderstanding of the requirements.
1.4	Lack of commitment regarding requirements between the customer and the project members.
1.5	Frequent requirements or specification changes.
2. **Estimations**	
2.1	Insufficient awareness of the importance of estimation.
2.2	Insufficient skills or knowledge of estimation methods.
2.3	Insufficient estimation for the implicit requirements.
2.4	Insufficient estimation for the technical issues.
2.5	Lack of stakeholders commitment for estimation.
3. **Planning**	
3.1	Lack of management review for the project plan.
3.2	Lack of assignment of responsibility.
3.3	Lack of breakdown of the work products.
3.4	Unspecified project review milestones.
3.5	Insufficient planning of project monitoring and controlling.
3.6	Lack of project members' commitment for the project plan.
4. **Team Organization**	
4.1	Lack of skills and experience.
4.2	Insufficient allocation of the resources.
4.3	Low morale.
5. **Project Management Activities**	
5.1	Lack of resource management of project managers throughout a project.
5.2	Inadequate project monitoring and controlling.
5.3	Lack of data needed to keep track of a project objectively.

5.2 Requirements

The *Requirements* viewpoint includes factors which are related to the understanding and commitment of the requirements among the project members. The factors for the requirements viewpoint are distinguished as follows:

(1.1) Ambiguous requirements
This item checks whether or not the requirements are clear and consistent. It is important for the project members to understand what the customer wants in order to achieve clear and consistent results.

(1.2) Insufficient explanation of the requirements
This item checks whether or not the customers have a sufficient explanation of the requirements regarding the system and/or software.

(1.3) Misunderstanding of the requirements
This item checks whether or not the developers have sufficient skills and/or knowledge to understand the requirements. The developers must have not only sufficient

technical skills and/or knowledge for the project, but must also have specific knowledge regarding the customers' domain.

(1.4) Lack of commitment regarding requirements between the customer and the project members

This item checks whether or not a commitment is obtained by both the project members and the customer. In order to confirm a commitment, it is important to have meetings to review requirements with project members and customers.

(1.5) Frequent requirement changes

This item checks whether or not changes in requirements or specifications are appropriately managed and whether or not project members are kept informed.

5.3 Estimations

The *Estimations* viewpoint includes factors related to the estimation itself, the technical methods for carrying out the estimation, and the commitment between project members and customers. The factors for the estimation viewpoint are distinguished as follows:

(2.1) Insufficient awareness of the importance of the estimation

This item checks whether or not the project members are aware of the importance of estimations. If they are not aware of the importance of the estimation, project members may tend to accept unreasonable requirements.

(2.2) Insufficient skills or knowledge of the estimation method

This item checks whether or not the project members have sufficient skills or knowledge for the estimation methods. In order to show the rationale of estimates, estimation methods must be utilized effectively.

(2.3) Insufficient estimation for the implicit requirements

This item checks whether or not the implicit requirements are considered and estimated. "Must-be" functions in the business area of the customer, or functions implemented in the previous system tend to be the implicit requirements.

(2.4) Insufficient estimation for the technical issues

This item checks whether or not the project members have estimated technical issues sufficiently. For example, technical issues include the selection of the programming language and the development environment.

(2.5) Lack of stakeholders' commitment for estimation

This item checks whether or not the commitment between project members and stakeholders is established. Stakeholders include customers, the sales division, and subcontractors. If a commitment is insufficient and the project members yield to political pressure, unrealistic estimations will be produced.

5.4 Planning

The *Planning* viewpoint includes factors related to the planning or scheduling activity and the commitment for the project plan among project members. The factors for the planning viewpoint are distinguished as follows:

(3.1) Lack of management review for the project plan

This item checks whether or not the project manager reviews the project plan. Management review includes reviewing the project plan to check for feasibility, etc.

(3.2) Lack of assignment of responsibility
This item checks whether or not the project has been systematically divided into activities by using the WBS, and whether or not the responsibility for each technical activity has also been clearly specified. The plan of the project should include not only engineering activities but also project management activities.

(3.3) Lack of breakdown of the work products
This item checks whether or not the work products to be produced by development have been correctly specified. The degree of breakdown should be determined for each development project.

(3.4) Unspecified project review milestones
This item checks whether or not sufficient project review milestones are set up. In the project review, project status, such as progress, quality of work products, etc., is reviewed.

(3.5) Insufficient planning of project monitoring and controlling
This item checks whether or not the plan of monitoring and control the project activities, issues, risks, and work products, is specified correctly.

(3.6) Lack of project members' commitment for the project plan
This item checks whether or not the plan has been reviewed by all of the project members. All engineers engaged in the project must recognize the project plan, and understand the concrete goals of the project.

5.5 Team Organization

The *Team organization* viewpoint includes factors related to the staffing of the projects, the fundamental skills and experience and morale of project members. The factors for the team organization viewpoint are as follows:

(4.1) Lack of skills and experience
This item checks whether or not project members have sufficient skills and experience to do their tasks.

(4.2) Insufficient allocation of resources
This item checks whether the resources are well allocated or not.

(4.3) Low morale
This item checks whether the morale of the project members is low or not.

5.6 Project Management Activities

The *Project management activities* viewpoint includes factors related to the project management activities. The factors which distinguish the project management activities viewpoint are as follows:

(5.1) Project manager lack of resource management throughout a project
This item checks whether or not the project members are actually working on the assigned project. Project managers should act as a firewall so that project members can devote themselves to their tasks.

Table 2. Projects used for our experiment

Projects	1: Requirements					2: Estimations					3: Planning						4: Organization			5: Management			Result
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	6	1	2	3	1	2	3	
PR1	D	D	D	D	D	A	S	S	A	D	A	D	D	D	D	D	A	N	D	D	D	D	Success
PR2	D	D	D	D	D	D	D	D	D	D	D	D	A	D	D	D	D	D	D	D	D	D	Success
PR3	D	D	D	D	S	D	D	A	S	D	D	D	D	D	A	D	D	D	D	D	D	D	Success
PR4	S	S	A	A	S	D	D	A	A	D	A	A	D	D	D	N	A	D	D	D	D	D	Success
PR5	D	D	D	D	A	D	D	D	D	D	D	A	D	A	A	D	D	D	D	D	A	D	Success
PR6	D	S	A	D	D	A	A	A	D	A	D	A	D	D	D	D	D	D	D	D	D	A	Success
PR7	D	D	A	S	A	D	D	D	D	D	D	A	D	S	D	D	D	D	D	D	D	D	Success
PR8	D	A	S	S	D	N	D	A	D	D	A	A	D	D	A	A	D	D	N	S	D	D	Success
PR9	D	A	D	A	S	D	D	D	D	D	A	A	D	A	A	D	D	D	D	D	D	A	Success
PR10	D	D	D	D	A	D	A	A	D	D	D	A	D	D	A	D	D	D	D	A	D	D	Success
PR11	D	S	S	A	D	D	D	S	S	D	D	D	D	D	D	D	D	D	A	D	D	D	Success
PR12	D	A	A	A	D	D	A	D	D	D	D	A	D	A	D	D	D	D	D	D	D	A	Success
PR13	D	A	D	A	D	D	D	D	D	D	A	S	S	D	A	A	A	A	D	A	A	N	Success
PR14	D	D	D	D	S	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	Success
PR15	D	A	A	A	A	D	A	A	D	D	D	D	D	D	D	D	S	A	D	S	D	D	Success
PR16	D	D	D	D	A	D	A	D	A	S	S	A	D	A	S	A	S	A	D	A	A	A	Success
PR17	D	D	D	D	D	D	A	D	D	D	A	A	A	S	A	A	D	D	D	A	A	D	Success
PR18	D	D	D	D	N	D	D	D	D	D	D	A	A	D	D	D	D	D	D	D	D	D	Success
PR19	D	D	D	D	S	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	Success
PR20	D	A	S	A	S	D	D	D	D	D	S	D	D	D	S	D	A	D	D	D	S	S	Success
PR21	D	A	A	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	Success
PR22	S	A	S	S	A	A	N	S	A	N	D	A	A	A	D	N	S	N	A	A	A	D	Success
PR23	A	A	D	A	S	D	D	A	S	D	A	D	A	A	S	A	D	D	D	D	A	S	Runaway
PR24	A	A	S	S	S	A	A	S	A	S	S	S	S	A	S	A	S	S	D	A	A	A	Runaway
PR25	S	A	D	D	S	D	D	D	D	D	S	D	D	S	S	D	D	D	D	D	D	D	Runaway
PR26	D	A	S	A	A	S	D	A	A	N	D	A	D	D	A	A	D	A	A	A	D	A	Runaway
PR27	D	A	A	A	A	D	S	A	S	S	D	A	A	D	D	A	A	A	D	D	D	D	Runaway
PR28	A	S	S	A	A	D	D	S	S	A	S	D	S	D	A	S	A	D	A	D	A	A	Runaway
PR29	S	A	S	A	D	S	A	A	A	D	D	A	A	A	S	D	A	D	A	D	S	S	Runaway
PR30	A	A	S	S	A	D	D	A	D	A	A	A	A	A	A	D	S	D	A	D	A	D	Runaway
PR31	D	D	D	D	D	D	D	D	D	D	S	S	S	S	S	S	S	S	D	D	S	S	Runaway
PR32	A	S	S	S	A	A	S	S	S	S	S	S	S	A	S	S	S	S	A	S	S	D	Runaway
PR33	A	D	D	D	D	D	A	D	A	D	D	S	A	D	D	A	S	D	D	A	D	D	Success
PR34	D	D	D	D	D	D	D	D	D	D	D	A	D	D	D	D	D	D	D	D	D	A	Success
PR35	A	A	A	A	S	D	A	D	D	D	D	A	A	D	D	D	D	D	D	D	D	D	Success
PR36	D	D	D	D	A	D	D	D	A	A	A	A	D	D	D	D	D	D	D	D	D	D	Success
PR37	D	D	D	D	N	D	D	D	D	D	D	A	D	D	D	D	D	D	D	D	D	D	Success
PR38	S	D	D	A	A	D	A	A	S	S	D	A	A	D	D	D	A	S	D	S	S	D	Runaway
PR39	A	S	S	S	A	N	N	S	N	N	N	S	S	S	D	A	A	N	S	D	D	D	Runaway
PR40	A	A	A	A	S	A	D	S	D	D	S	S	S	S	S	S	D	D	D	D	A	A	Runaway

(5.2) Inadequate project monitoring and controlling

This item checks whether or not progress monitoring is adequately done, whether or not the progress reporting is actually done, and whether or not corrective action is adequately taken.

(5.3) Lack of data needed to keep objective track of a project

This item checks whether or not the project managers are able to objectively keep track of a project on the basis of the software metrics collected during development. If such data are not available, then it is difficult to recognize the project status correctly to make management decisions appropriately.

6 Application of the Bayesian Classification

In this section, we apply the proposed questionnaire in Section 5 to sample projects, and successively apply the naive Bayesian classifier to the assessment data obtained from the questionnaire.

6.1 Filling out the Questionnaire

We chose 40 projects, which were part of the projects performed from 1996 to 1998 by the SSBC. The SEPG distributed the questionnaires designed in Section 5 to the

project managers or the project leaders of 40 target projects, and explained the details of the questionnaire and the purpose of the trial. The responses to the questionnaire were collected by the SEPG after one month. Table 2 shows the collected responses. In Table 2, the answers "Strongly agree", "Agree", "Neither agree nor disagree", and "Disagree" are shown as characters "S", "A", "N", and "D", respectively.

6.2 Determination of Runaway Projects

Since all of these projects completed their development, the SEPG had already identified the runaway projects according to the decision process mentioned in Section 3. As a result, 13 projects out of 40 were classified as runaway projects. Thus, the column, "Result", in Table 2, shows the actual result of the classification.

6.3 Prediction by Naive Bayesian Classifier

We then applied the naive Bayesian classifier to the response data of all projects, and calculated the conditional probabilities of being runaway. On applying the naive Bayesian classifier, we assume that each response of question is a nominal value.

In order to show the effectiveness of the proposed approach, we perform 10-fold cross validation on the collected data. The evaluation through k-fold cross validation method is one of the most common in machine learning community. The data set is here split into k equally sized subsets, and then in i-th iteration ($i = 1 \cdots k$) i-th subset is used for testing the classifier that has been learned from all other remaining subsets. Notice that each instance of data is classified exactly once. The number of subsets k is usually set to 10.

Let us show an example of 10-fold cross validation. At first, we randomly split 40 projects in Table 2 into 10 subsets. Suppose that a subset contains projects PR3, PR12, PR37, and PR39. Next, we select this subset for testing, and thus rest 9 subsets are used for Bayesian learning. Four projects are then classified by the learned Bayesian model. For each subset, the same procedure is performed. Finally, we obtain 40 testing results to evaluate the overall accuracy of the prediction.

Table 3. Results of 10-fold cross validation

	Predicted	
Actual	Success	Runaway
Success	22	5
Runaway	2	11

Table 3 shows the result of 10-fold cross validation. The rows show the number of projects that are actually runaway or success. The columns show the number of projects that are predicted as runaway or success.

As shown in Table 3, 33 (that is, 22+11) out of 40 projects can be predicted correctly. The predicting accuracy is thus 82.5%. According to Fisher's exact test, we can say that actual and predicted results are not independent with statistical significance $\alpha = 0.01$. We can conclude that it is rather high accuracy.

This result indicates that the prediction works well irrelevant to the selection of projects. For this result, we can say that our proposed approach can be applicable to the prediction of runaway projects.

7 Conclusion

In this research, we tried to construct prediction model of the runaway projects using the naive Bayesian classification method. We showed that a naive Bayesian classification model can be applicable to the response data of questionnaire and the prediction can be done with high accuracy. Since the data used in this study are the information regarding the final status of projects and the responses of the questionnaire about the software risks, this data can be collected easily in actual development fields.

As future work, we will apply the proposed approach to other data set obtained from several companies. Furthermore, in order to make the prediction more accurate, we have to seek the other classification methods suitable for the software engineering.

Acknowledgments. The authors would like to thank Dr. Keishi Sakamoto for his support to our entire process of improvement activities. The authors also would like to express great thanks to the project members of the 40 projects who willingly cooperated with our requests.

References

1. Boehm, B.W.: Industrial software metrics top 10 list. IEEE Software **4** (1987) 84–85
2. Jiang, J., Klein, G.: Software development risks to project effectiveness. Journal of Systems and Software **52** (2000) 3–10
3. Wohlin, C., Andrews, A.A.: Prioritizing and assessing software project success factors and project characteristics using subjective data. Empirical Software Engineering **8** (2003) 285–303
4. Williams, R.C., Pandelios, G.J., Behrens, S.G.: Software risk evaluation (SRE) method description (version 2.0). Technical Report CMU/SEI-99-TR-029, Software Engineering Institute (1999)
5. Conrow, E.H., Shishido, P.S.: Implementing risk management on software intensive projects. IEEE Software **14** (1997) 83–89
6. Fairley, R., Rook, P.: Risk management for software development. In: Software Engineering. IEEE CS Press (1997) 387–400
7. Karolak, D.W.: Software Engineering Risk Management. IEEE CS Press, CA (1996)
8. Sisti, F.J., Joseph, S.: Software risk evaluation method version 1.0. Technical Report CMU/SEI-94-TR-19, Software Engineering Institute (1994)
9. Humphrey, W.S.: A Discipline for Software Engineering. Addison-Wesley, MA (1995)
10. Yourdon, E.: Death March: The Complete Software Developer's Guide to Surviving 'Mission Impossible' Projects. Prentice-Hall Computer Books (1997)
11. Fenton, N.E., Pfleeger, S.L.: Software Metrics : A Rigorous & Practical Approach. PWS Publishing (1997)
12. Mizuno, O., Kikuno, T., Takagi, Y., Sakamoto, K.: Characterization of risky projects based on project managers' evaluation. In: Proc. of 22nd International Conference on Software Engineering. (2000) 387–395

Effort Estimation Based on Collaborative Filtering

Naoki Ohsugi, Masateru Tsunoda, Akito Monden, and Ken-ichi Matsumoto

Graduate School of Information Science, Nara Institute of Science and Technology
Kansai Science City, 630-0192 Japan
{naoki-o, masate-t, akito-m, matumoto}@is.aist-nara.ac.jp

Abstract. Effort estimation methods are one of the important tools for project managers in controlling human resources of ongoing or future software projects. The estimations require historical project data including process and product metrics that characterize past projects. Practically, in using the estimation methods, it is a problem that the historical project data frequently contain substantial missing values. In this paper, we propose an effort estimation method based on *Collaborative Filtering* for solving the problem. Collaborative Filtering has been developed in information retrieval researchers, as one of the estimation techniques using defective data, i.e. data having substantial missing values. The proposed method first evaluates similarity between a target (ongoing) project and each past project, using vector based similarity computation equation. Then it predicts the effort of the target project with the weighted sum of the efforts of past similar projects. We conducted an experimental case study to evaluate the estimation performance of the proposed method. The proposed method showed better performance than the conventional regression method when the data had substantial missing values.

1 Introduction

Many researches have proposed estimation methods of software development effort [1], [2], [6], [7], [17], [18], [20]. They provide systematic procedures to predict the effort (e.g. man-months) of ongoing or future projects based on historical (past) project data. Accurate estimates provide practitioners many advantages in managing the development. These allow project managers to make effective use of valuable resources. Or these give marketers baselines of bid for external contracts; i.e., they can know how many people will be needed to complete the certain works in the particular duration. In fact, estimation methods have not been limited to uses for software development; can be used for any kind of tasks. For instance, a recent study built a method to estimate the effort required to become ISO 9001 certified [13].

These estimation methods usually uses both process and product metrics to characterize each project. Process metrics, which quantify attributes of the development process and of the development environment, are typically collected by hand in forms of progress reports. On the other hand, product metrics, which quantify characteristics of artifacts (e.g. source code) produced during the development process, can be automatically collected by measurement tools.

One of the practical problems in using the estimation methods is that the historical project data usually contain substantial numbers of missing values [4], [9]. Especially,

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 274–286, 2004.

process metrics contain larger numbers of them since they are collected by hand. One reason is that different divisions in an organization may have different policy of data collection, i.e. some project collects a particular metric while other projects do not. And, even if the organization has a unified policy, not all the metrics are collected in each project due to pressing development schedule. Furthermore, the organizational policy to collect a particular set of metrics often changes as the organization's software process matures, thus, older projects tend to have less data than new projects.

Some complementary techniques have been developed for dealing with missing values (henceforth, missing data techniques or MDTs) [10]. Strike et al. [19] conducted experimental evaluation of some MDTs, and found all of the techniques perform well in terms of small biases and high precision. The techniques were: *listwise deletion, mean imputation* and some types of *hot-deck imputation.* Listwise deletion is the simplest technique to ignore data sets that have missing values. Mean imputation is a technique to fill the missing values on a variable with the mean of data sets that are not missing. Hot-deck imputation is alternative forms of imputation that are based on estimates of the missing values using other variables from the subset of the data that have no missing values. They concluded that the simplest technique, listwise deletion, is a reasonable choice.

However, MDTs can give bad influences to the accuracy of estimation, according to the amount and distribution of missing values. Kromrey and Hines [12] analyzed the performance of MDTs with a two predictor regression model. Their results showed that listwise deletion technique did not performed well when the level of missing data was more than 30 percent of all data. And the imputation techniques did not perform well when the distribution of missing values is not uniformly. In the context of effort estimation, the ratio of data that have missing values can be frequently larger than 30 percent [5]. And the distribution of missing values is usually not uniform.

In this paper, we propose *Collaborative Filtering* (CF) based effort estimation method, under the assumption that the (historical) predictor data have a large amount of missing values. CF is one of the estimation techniques using defective data having substantial missing values, in information retrieval research domain. The proposed method first evaluates similarity between a target (ongoing) project and each past project, using vector based similarity computation equation. Then, it predicts the effort of the target project with weighted sum of the effort of past similar projects.

Up to date, there is no method that can effectively estimate software development effort with a large amount of missing values, while CF have been developed for estimating user preferences using defective data that have substantial missing values. Estimating a user's preference is to answer a question such as: which items are likely to be preferred by the user? And, to what degree the user seems to prefer the items? As we consider this is an estimation of a value of a metric (an item), we believe CF can be applied for estimating the effort as well as the user preferences, to improve prediction accuracy. However, conventional CF cannot be directly applied to software metrics since value range of each metric is not constant while that of item ratings in CF are constant.

In this paper, we propose a procedure on how to apply CF for estimating software development effort. We also present results of experimental case study conducted for evaluating the performance of the proposed method. Our evaluation criteria focus on the accuracy of prediction with the absolute and relative error. In the case study, we

used industrial data, which had been collected in a company, including several kinds of process metrics that have large amount of missing values (approximately 60% of all data), as an evaluating data set. We compared the proposed method, vs. the *Stepwise Multiple Regression Models* enhanced by some different MDTs.

In what follows, Section 2 introduces related works. Section 3 proposes an estimation procedure based on CF. Section 4 describes a procedure of an experimental case study for evaluating the proposed method. Section 5 shows the results of our case study and discusses their implications and limitations. In the end, Section 6 concludes the paper with a summary and some future topics.

2 Related Work

CF has been developed as a technique for realizing *Recommender Systems,* by information retrieval research community. Recommender Systems estimate individual user's preferences, for recommending likely preferred items selected from vast amount of items. Resnick et al. [14] developed the *GroupLens* system for recommending preferred Usenet articles extracted from large amount of news articles, to individual users. GroupLens behave as follows, when it recommends particular user u_a. Before the recommending, each user evaluates of the articles which s/he already read in 5 level scales: 5 (prefer) to 1 (not prefer). The system first finds similar users by computing similarities between u_a and the other users, with the vector operation on the users' evaluations. Then, the system estimates u_a's evaluations to every article i_j which u_a has never read, with weighted sums of the other similar users' evaluation to the article i_j. At last, it recommends the articles that will be likely high evaluated by u_a. Our study provides a specific procedure to apply this algorithm for estimating software development effort. To date, many systems and algorithms have been proposed in the CF researches [3], [8], [16]. It is plausible that these algorithms can estimate development effort as well as user preferences.

CF has challenged for accurately estimating user preferences using defective data that have vast amount of missing values. In a typical Recommender Systems most users do not evaluate more than 1 % of all items since existing items are immensity. (e.g., you can't evaluate all Usenet news in using GroupLens.) Sarwar et al. [16] proposed a CF algorithm to accurately estimate users' preferences when the data are extremely sparsity. This algorithm succeeded on the Amazon.com's website with a famous statement "Customers who bought this book also bought these books". We focus on this feature in CF, which can accurately estimate using defective data. The algorithms can provide improved prediction accuracy using incomplete data when we have instantiated procedure on how to apply CF for estimating software development effort.

3 Estimation Procedure

This section describes our estimation procedure based on CF. In the proposed procedure, we use database in form of $m \times n$ matrix as shown in Fig. 1 where $p_i \in \{p_1, p_2, ..., p_m\}$ denotes i-th project, $m_j \in \{m_1, m_2, ..., m_n\}$ denotes j-th metric, and $v_{i,j} \in \{v_{1,1},$

$v_{1,2}$, ..., $v_{m,n}$} denotes value of metric m_j observed in project p_i. Note that some elements in the matrix are actually empty (missing values).

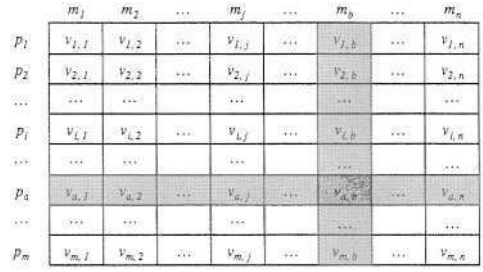


Fig. 1. $m \times n$ table used for estimation .

Here we assume a-th project p_a is a target project in which b-th metric $v_{a,b}$ is to be estimated. We denote $\hat{v}_{a,b}$ as an estimated value for $v_{a,b}$. Below describes 3-step procedure to obtain $\hat{v}_{a,b}$.

Step 1 (normalization of metrics): Since each metric has different value range, this first step normalizes values of metrics so that the value range becomes [0, 1]. The normalized value $normalized(v_{i,j})$ of metric $v_{i,j}$ (of project p_i) is calculated by the following equation.

$$normalized(v_{i,j}) = \frac{v_{i,j} - \min(P_j)}{\max(P_j) - \min(P_j)} \tag{1}$$

where P_j denotes a set of projects in which value of metric m_j was observed (collected), $max(P_j)$ and $min(P_j)$ denotes maximum and minimum value in $\{v_{x,j} \mid p_x \in P_j\}$ respectively.

Step 2 (computation of similarity between projects): In this step, similarity $sim(p_a, p_i)$ between the target project p_a and other projects p_i is computed. Many algorithms for computing $sim(p_a, p_i)$ have been proposed in conventional CF researches [3], [8], [14], [16].

We employ a similarity computation algorithm that was proposed in the field of information retrieval to evaluate the similarity between two documents. The similarity is often evaluated by treating each document as a vector of word frequencies and computing the cosine of the angle formed by the two frequency vectors [15]. We can adopt this formalism to compute $sim(p_a, p_i)$, where projects take the role of documents, metrics take the role of words, and values of the metrics take the role of word frequencies. Formally, we can define the $sim(p_a, p_i)$ between the target project p_a and other projects p_i as:

$$sim(p_a, p_i) = \frac{\sum_{j \in M_a \cap M_i} \left(normalized(v_{a,j}) \times normalized(v_{i,j}) \right)}{\sqrt{\sum_{j \in M_a \cap M_i} \left(normalized(v_{a,j})^2 \right)} \sqrt{\sum_{j \in M_a \cap M_i} \left(normalized(v_{i,j})^2 \right)}} \tag{2}$$

where M_a and M_i denotes a set of metrics observed in project p_a and p_i respectively. The value range of $sim(p_a, p_i)$ is [0, 1]. Note that this computation is suitable for data set containing missing values.

Step 3 (computation of estimation): This step calculates an estimated value $\hat{v}_{a,b}$ of the metric m_b on the target project p_a using $sim(p_a, p_i)$ calculated in previous step. Many algorithms have been proposed to implement this step [3], [8], [14], [16].
We employ weighted sum to compute estimation. The estimated value is computed as the sum of the metrics' values given by the other projects similar to p_a. Each value is weighted by the corresponding the $amplifier(p_a, p_i)$ and the $sim(p_a, p_i)$ between p_a and p_i. Formally, we can define the estimated value $\hat{v}_{a,b}$ as:

$$\hat{v}_{a,b} = \frac{\sum_{i \in k-nearestProjects} \left(v_{i,b} \times amplifier(p_a, p_i) \times sim(p_a, p_i) \right)}{\sum_{i \in k-nearestProjects} sim(p_a, p_i)} \tag{3}$$

where *k-nearestProjects* denotes a set of *k* projects (called *neighborhoods*) that have highest similarity with p_a. The neighborhoods must have m_j as an observed metric. Generally, the neighborhood size *k* affects the estimation accuracy.

To improve accuracy of the estimation, the $amplifier(p_a, p_i)$ calculates an approximate value of the $v_{i,b}$ with comparing the sizes of projects p_a and p_i, i.e. the amplifier indicates what times p_a's value is p_i's value. The *amplifier* derived from the fact that the p_a's value is several times larger (or smaller) than the p_i's value when p_i is a similar to p_a. It's because the similarity is computed by vector operation but not *Euclidean distance. Sim*(*pa, pi*) is computed by comparing tendencies of the values, whereas *Euclidean distance* is computed by comparing absolute values (cf. [286] for *Euclidean distance*-based similarity) . Formally, we can define the $amplifier(p_a, p_i)$ as:

$$amplifier(p_a, p_i) = \frac{\sum_{j \in M_a \cap M_i} \left(\frac{normalized(v_{a,j})}{normalized(v_{i,j})} \right)}{|M_a \cap M_i|} \tag{4}$$

where $|M_a \cap M_i|$ denotes the number of elements of the product set $M_a \cap M_i$.

4 Case Study

4.1 Goal

The goal of this case study is to evaluate the prediction performance of the proposed method under the situation that available data contain a large amount of missing values. In order to evaluate the prediction performance, we used the absolute and relative error. We compared the proposed method, vs. the stepwise multiple regression models enhanced by the techniques for dealing with missing values.

4.2 Data Source

Table 1 shows the data we used for the case study. The data includes 1081 projects and 14 metrics, were collected from a Japanese software company in a decade. The company has developed some software packages that are the reusable large software components. 36% of all projects in the company tailored them to meet the needs of the individual customers. 13% developed individual software without the software packages. The others (51%) were unknown projects with the missing values.

Table 1 also shows the rates of missing values in each metric. 59.83% of all data were missing originally; however, the rates of missing values were quite different in each metric. In m_3, m_4 and m_{14}, there was no missing value. In m_2 there were few missing values (7.49%), whereas in the other 10 metrics there were at least 70% missing values.

Table 1. Data used in case study

	Metrics	Missing value rate
m_1	Mainframe or not (Mainframe 1, Others 0)	75.76 %
m_2	New development or not (New 1, Others 0)	7.49 %
m_3	Total design cost	0.00 %
m_4	Total coding cost	0.00 %
m_5	Design cost for regular (payroll) staffs of a company	86.68 %
m_6	Design cost for dispatched staffs from other companies	86.68 %
m_7	Design cost for subcontract companies	86.59 %
m_8	Coding cost for regular staffs	86.68 %
m_9	Coding cost for dispatched staffs	86.68 %
m_{10}	Coding cost for subcontract companies	86.59 %
m_{11}	# of faults found in the review of conceptual design	83.53 %
m_{12}	# of faults found in the review of functional design	70.77 %
m_{13}	# of faults found in the review of program design	80.20 %
m_{14}	Testing cost	0.00 %

4.3 Stepwise Multiple Regression Model

To construct a multiple regression model, we used a stepwise method to select a combination of predictor variables that strongly affect the objective variable. The stepwise procedure involves (1) identifying an initial model, (2) repeatedly altering the model at the previous step by adding a predictor variable whose coefficient is considered non-zero ($p < 0.05$), or by removing a variable whose coefficient cannot be considered non-zero ($p < 0.1$), and (3) terminating the search when stepping is no longer possible.

We employed the following three types of MDTs. These techniques are widely used for constructing regression models using data with missing values [10], [12], [19].

Listwise Deletion: This method constructs models with the projects which do not contain any missing values. The data of projects which have some missing values are simply excluded.

Pairwise Deletion: This method uses all available values wherever possible, then, it constructs models using the same method to listwise deletion. For example, all observed values are used regardless of whether the projects have missing other metrics, when it calculates the averages of each metrics or the correlation coefficients between the metrics.

Mean Imputation: This method fills each missing value with the mean of observed values. After all of missing values are filled, it then constructs models using the completed data.

4.4 Evaluation Criteria

In this paper we use the following five evaluation criteria. These criteria are commonly used in estimation methods [6], [11], [19]. In the rest of paper, Y denotes actual (observed) objective variable, $\hat{Y}$ denotes its predicted variable, and "operation" indicates an action to compare Y and $\hat{Y}$. Five criteria are computed after the operation has executed t times.

Mean Absolute Error (MAE):

$$MAE = \frac{\sum \left| \hat{Y} - Y \right|}{t} \tag{5}$$

***Variance of Absolute Error* (VAE):**

$$VAE = \frac{\sum \left(\left| \hat{Y} - Y \right| - MAE \right)^2}{t} \tag{6}$$

Mean Relative Error (MRE):

$$MRE = \sum \left| \frac{\hat{Y} - Y}{Y} \right| \times \frac{1}{t} \tag{7}$$

Variance of Relative Error (VRE):

$$VRE = \sum \left(\left| \frac{\hat{Y} - Y}{Y} \right| - MRE \right)^2 \times \frac{1}{t} \tag{8}$$

Pred25: Ratio of operations whose relative errors are under 25%. Generally, estimation is considered accurate when Pred25 is small.

$$Pred25 = \frac{\sum isAccurate\left(\hat{Y} - Y\right)}{t}$$

$$isAccurate\left(\hat{Y} - Y\right) = \begin{cases} 1 & \left|\hat{Y} - Y\right| < 25 \\ 0 & \left|\hat{Y} - Y\right| \geq 25 \end{cases} \tag{9}$$

4.5 Experimental Procedure

The experimental procedure consists of the following 4 steps.

i. We divided the data into 50-50 two sets randomly*: fit dataset* and *test dataset.* In the case study, we used fit dataset for constructing estimation models, and estimated each testing cost m_{14} on the project in the test dataset. With the same way, we made 10 different pairs of fit and test datasets.
ii. We evaluated performance of CF described in the section 3, using 10 different pairs of fit and test datasets. Every evaluation criterion was measured by the averaged values of 10 results with the different pairs of the datasets, for improving reliability of the experimental results. In order to find an appropriate neighborhood size (the value of k in the equation (3)), we measured MRE of each neighborhood size (from k = 1 up to k = 50) and picked up k having smallest MRE. We then measured MAE, VAE, VRE and Pred25 of CF employing the selected neighborhood size.
iii. We evaluated performance of stepwise multiple regression models with three types of MDTs described in section the 4.3, using 10 different pairs of fit and test datasets. In each MDT, all of the criteria were measured by the average of the results with 10 different pairs of the datasets.
iv. We concluded about which method outperformed the others, with comparing the evaluation criteria resulted by the above ii and iii. And we also considered about the implications and limitation observed from our results.

5 Experimental Results

5.1 Overall Results

Table 2 presents the overall result of our case study. Table 2 includes resulted evaluation criteria with using CF with the neighborhood size 22 and using stepwise multiple regression models with the three types of MDTs. As described in the section 4.4, a lower value indicates better performance than a higher one, in MAE, VAE, MRE and VRE; while, a higher value is better in Pred25. Each evaluation criterion

was statistically significant at the 97% or more confidence level, with the t-testing or f-testing.

Our results indicate CF outperformed the others, i.e., it made the most accurate estimations with the lowest variances. All of evaluation criteria consistently indicate CF was the most effective. In our results, listwise deletion performed relatively better performance than the other MDTs, since MAE, MRE and VRE resulted as lower values. MAE, MRE and VRE of mean imputation and pairwise deletion indicate these were ineffective, although these were relatively better in terms of VAE or Pred25.
The detailed results on each method are presented in the following sections.

Table 2. Averaged evaluation criteria resulted with the 10 different experimental dataset

	MAE	VAE	MRE	VRE	Pred25
CF ($k = 22$)	0.21	2.20	0.82	3.45	36%
Regression (listwise deletion)	0.70	16.45	30.22	287581.18	10%
Regression (pairwise deletion)	52.75	97171.84	6344.27	18937623097.60	12%
Regression (mean imputation)	1.33	5.07	331.69	24208218.49	4%

5.2 CF Based Estimation

Fig. 2 presents a line graph of the MREs resulted by applying CF with each neighborhood size (from $k = 1$ up to $k = 50$). In the graph, the horizontal axis indicates neighborhood sizes, and the vertical axis indicates MREs corresponding to horizontal axis. This result shows the most accurate estimation was observed at neighborhood size of 22.

Fig. 3 intuitively presents the estimation accuracy of the most effective case of CF. This graph is illustrated as a double logarithmic chart for ease of understanding. The horizontal axis in the graph, indicates the estimated testing costs and vertical axis indicates the actual costs corresponding to horizontal axis; i.e., the points are plotted near by the line of y = x if the estimations were correct, while, distant point from the line indicates incorrect estimation and a more distant point from the line presents more incorrect estimation than the nearer one. This graph suggests there were more projects having lower testing costs than higher ones. And also shows CF accurately estimated the testing costs regardless of whether the actual costs were high or low.

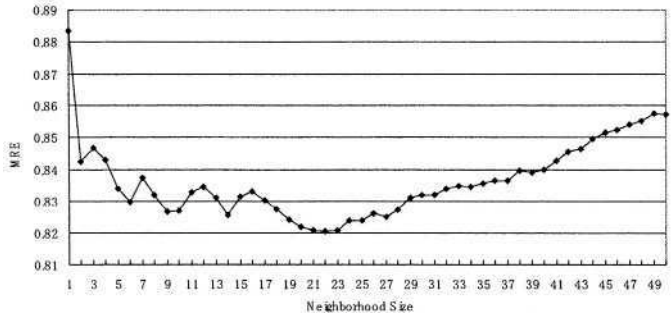


Fig. 2. Accuracy of the CF in each neighborhood size (from $k = 1$ up to $k = 50$)

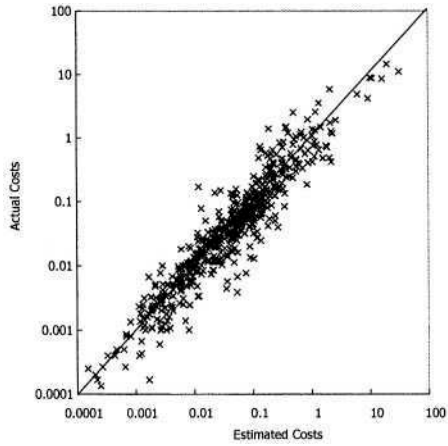


Fig. 3. Relationship between actual and estimated costs using CF (k = 22)

5.3 Stepwise Multiple Regression Model

Fig. 4, Fig. 5 and Fig. 6 present the estimation accuracies of the most effective case of each stepwise multiple regression model with listwise deletion, pairwise deletion and mean imputation. These graphs are illustrated like Fig. 3 described in the section 5.2. The graphs suggest there were more projects having lower testing costs than higher ones. And also show accuracies of the estimation were lower when the actual costs were lower. Every regression model estimated the lower actual costs incorrectly as bigger ones.

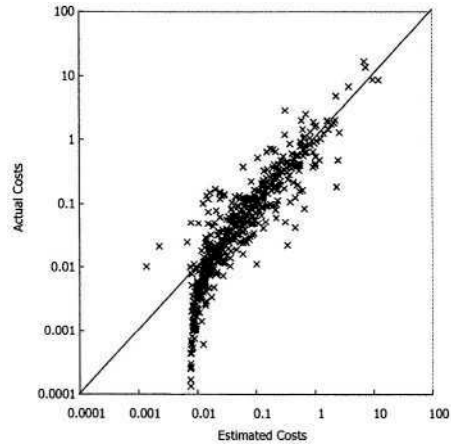


Fig. 4. Relationship between actual and estimated costs using stepwise multiple regression model with listwise deletion

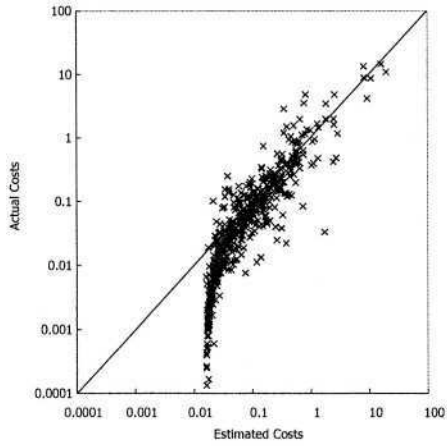


Fig. 5. Relationship between actual and estimated costs using stepwise multiple regression model with pairwise deletion

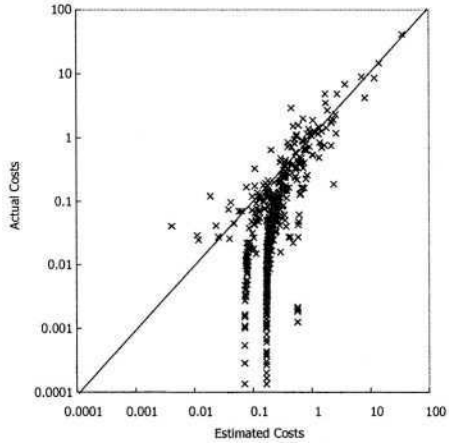


Fig. 6. Relationship between actual and estimated costs using stepwise multiple regression model with mean imputation

5.4 Discussion

Although our results indicate that the proposed method greatly outperformed stepwise regression models with the three types of MDTs, the MRE of CF (0.82) suggests the proposed method needs further improvements. One of the ways for improving is to develop various alternative CF algorithms for computing similarities and estimations,

which can be used instead of the equation (2) and (3). Especially, variations of the amplifier in the equation (3) have the possibility for improving accuracies of the estimations. Another way for improving the performances is to develop some methods to select an appropriate combination of predictor metrics that strongly affect the objective metrics, like the stepwise method described in the section 4.3.

One of the limitations of our study, we evaluated just only three types of MDTs for dealing with missing values: listwise deletion, pairwise deletion and mean imputation; however, there still exists more MDTs. For instance, Strike et al. [19] reported that *hot-deck imputation* provided the best performance in his simulation. We must compare the proposed method and the regression models using the other MDTs including hot-deck imputation in order to evaluate these performances.

Furthermore, we evaluated using just only one dataset; however there are many conditions (i.e., various amounts and distributions of the missing values) in the datasets collected by the actual industrial organizations. We must encourage the results of this case study on alternative datasets and conduct experimental simulation with various conditions of data, for improving reliability of the study.

6 Conclusions

This paper proposed a method for estimating software development effort based on collaborative filtering. The proposed method showed better performance than the conventional regression method when the data had substantial missing values.

In the future, we are going to develop CF algorithms for computing similarities and estimations, to improve estimation performances. Furthermore, we will conduct an experiment to compare the proposed method with other MDTs including hot-deck imputation method. we also will encourage the replication of this case study on alternative datasets and conduct experimental simulation with various conditions of data, for improving reliability of the study.

References

1. Albrecht, A., Gaffney, J.: Software Function, Source Lines of Code, and Development Effort Prediction. IEEE Trans. on Software Eng., vol.9, no.6, pp.83-92 (1979)
2. Boehm, B.W.: Software Engineering Economics. IEEE Trans. on Software Eng., vol.10, no.1, 4-21 (1984)
3. Breese, J. S., Heckerman, D., and Kadie, C.: Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proc. of the 14th Conf. on Uncertainty in Artificial Intelligence,* pp.43-52 (1998)
4. Briand, L., Basili, V., and Thomas, W.: A Pattern Recognition Approach for Software Engineering Data Analysis. IEEE Trans. on Software Eng., vol.18, no.11, pp.931-942 (1992)
5. Briand, L., El Eman, K., and Wieczorek, I.: Explaining the Cost of European Space and Military Projects. In Proc. Int'l Conf. Software Eng., vol.1, no.1, pp.61-88 (1996)

6. Conte, S.D., Dunsmore, H.E., and Shen, V.Y.: Software Engineering Metrics and Models. The Benjamin/Cummings Publishing Company, Inc., Menlo Park, California (1986)
7. Finnie, G., and Wittig, G.: A Comparison of Software Effort Estimation Techniques: Using Function Points with Neural Networks, Case-Based Reasoning and Regression Models. Journal of Systems and Software, vol.39, pp.281-289, 1997.
8. Goldberg, D., Nichols, D., Oki, B.M., and Terry, D.: Using Collaborative Filtering to Weave an Information Tapestry. *Comm. of the ACM,* vol.35, no.12, pp.61-70 (1992)
9. Gray, A., and MacDonnell, D.: A Comparison of Techniques for Developing Predictive Models of Software Metrics. Information and Software Technology, vol.3, pp.425-437 (1997)
10. Little, R., and Rubin, D.: Statistical Analysis with Missing Data. John Wiley & Sons, Inc. (1987)
11. Khoshgoftaar, T.M., Munson, J.C., Bhattacharya, B.B., and Richardson, G.D.: Predictive Modeling Techniques of Software Quality from Software Measures. IEEE Trans. on Software Eng., vol.18, no.1, 979-987 (1992)
12. Kromrey, J., and Hines, C.: Nonrandomly Missing Data in Multiple Regression: An Empirical Comparison of Common Missing-Data Treatments. Educational and Psychological Measurement, vo.54, no.3, pp.573-593 (1994)
13. Rahhal, S., and Madhavji, N.: An Effort Estimation Model for Implementing ISO 9001. In *Proc. of the 2nd IEEE Int'l Software Eng. Standards Symp.,* pp.278-286 (1995)
14. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J.: GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proc. ACM Conf. on Computer Supported Cooperative Work (CSCW'94),* Chapel Hill, North Carolina, United States, 175-186 (1994)
15. Salton, G., and McGill, M.: Introduction to Modern Information Retrieval, McGraw-Hill, New York (1983).
16. Sarwar, B.M., Karypis, G., Konstan, J.A., and Riedl, J.: Item-Based Collaborative Filtering Recommendation Algorithms, In *Proc. 10th International World Wide Web Conference (WWW 10),* Hong Kong, 285-295 (2001)
17. Shepperd, M., and Schofield, C.: Estimating Software Project Effort Using Analogies. IEEE Trans. on Software Eng., vol.23, no.12, pp.76-743 (1997)
18. Srinivasan, K., and Fisher, D.: Machine Learning Approaches to Estimating Software Development Effort. IEEE Trans. on Software Eng., vol.21, no.2, pp.126-137 (1995)
19. Strike, K., El Eman, K., and Madhavji, N.: Software Cost Estimation with Incomplete Data. IEEE Trans. on Software Eng., vol.27, no.10, pp.890-908 (2001)
20. Walston, C., and Felix, C.: A Method of Programming Measurement and Estimation. IBM Systems Journal, vol.1, pp.54-73, 1977.

Effective Software Project Management Education through Simulation Models: An Externally Replicated Experiment

D. Rodríguez[1], M. Satpathy[1], and D. Pfahl[2]

[1]Dept. of Computer Science, The University of Reading, Reading, RG6 6AY, UK
{d.rodriguezgarcia, m.satpathy}@rdg.ac.uk
[2] Fraunhofer Institute for Experimental Software Engineering (IESE)
Sauerwiesen 6, 67661, Kaiserslautern, Germany
pfahl@iese.fhg.de

Abstract. It is an undeniable fact that software project managers need reliable techniques and robust tool support to be able to exercise a fine control over the development process so that products can be delivered in time and within budget. Therefore, managers need to be trained so that they could learn and use new techniques and be aware of their possible impacts. In this context, effective learning is an issue. A small number of empirical studies have been carried out to study the impact of software engineering education. One such study is by Pfahl *et al* [11] in which they have performed a controlled experiment to evaluate the learning effectiveness of using a process simulation model for educating computer science students in software project management. The experimental group applied a Systems Dynamics simulation model while the control group used the COCOMO model as a predictive tool for project planning. The results indicated that students using the simulation model gain a better understanding about typical behaviour patterns of software development projects. Experiments need to be externally replicated to both verify and generalise original results. In this paper, we will discuss an externally replicated experiment in which we keep the design and the goal of the above experiment intact. We then analyse our results in relation to the original experiment and another externally replicated experiment, discussed in [12].

1 Introduction

The most important objective of software project management is to use the available resources effectively so that the tasks and subtasks of the development process are kept in schedule and within budget without sacrificing on product quality. Many interacting factors throughout the software life cycle can have impact over the cost and schedule of a project and the quality of the product. To monitor and control software development projects, management experience and knowledge on how to balance the various influential factors are required. To address these issues, process simulation techniques have been applied to the domain of Software Engineering (SE) during the last decade Abdel-Hamid and Madnick [1], etc. But although the potential of simulation models for the training of managers has long been recognised [6, 9],

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 287–301, 2004.

very few experimental studies involving process simulation as a means for software project management education have been performed.

The results of these experiments indicate that a natural one-way causal thinking could be detrimental to the success of software managers. They must rather adopt a multi-causal or systems thinking. Moreover, they must be aware of (unexpected) feedback to their management decisions. A dynamic model like the Systems Dynamics not only present a white-box view of the influencing factors affecting a certain process attribute of interest (say, cost or effort) but also it often provides some feedback of the possible outcome [1]. A static model like COCOMO [2] only presents a black-box view of the system. These findings highlight the need for new learning and education strategies. The first strategic step for teaching software project management must already be included in the curriculum of students. University education must teach computer science and software-engineering students not only technology related skills but also a basic understanding of typical management phenomena occurring in industrial (and academic) software projects. Pfahl *et al* [11] have performed a controlled experiment that evaluates the effectiveness of using a process simulation model for university education in software project management. In their study, the experimental group applied a Systems Dynamics simulation model while the control group used the COCOMO model as a predictive tool for project planning. The results of the experiment indicate that students using the simulation model gain a better understanding about typical behaviour patterns of software development projects. The result of an experimental study is not usually extrapolated to all possible software environments, especially so when we deal with human subjects. Many uncontrollable sources of variation exist from one environment to another; therefore, more studies need to be conducted in a variety of environments. In addition, replicated studies can help the researchers to combine knowledge directly or via some form of meta-analysis. Since intervening factors and threats to validity can almost never be completely ruled out of a study, complementary studies also allow more robust conclusions to be drawn when related studies can address one another's weak points [14]. Replication of a study means repeating a study based on the design and results of a previous study, whose goal is either to verify or broaden the applicability of the results of the original study. A replication can be internal or external. In an internal replication, the original researchers perform the replication, whereas in an external replication, different researchers conduct the replication. A scientific hypothesis gains increasing acceptance when external replications arrive at the same conclusion.

In this paper, we discuss an external replication of the same experiment (of Pfahl *et al* [11]) which was performed at the University of Reading in England. Another externally replicated study has also been conducted at the University of Oulu, Finland [12]. We discuss here our findings and perform a meta-analysis over the results of the original experiment and the results of the two replications. The organization of the paper is as follows. Section 2 presents the experimental details of the study. Section 3 summarizes the results of the data analysis and a brief meta-analysis over the results of the three experiments. Section 4 discusses the various threats to the validity of the study. Finally, Section 5 concludes the paper.

2 Description of the Experiment

Ours is a replication study; therefore we have tried to keep the same intent and environment as of the original experiment. However, the original experiment suggested some modifications as to the timing of the experiment, and in our experiment we have followed the suggestions. The main objective of developing and applying a simulation-based training module has been to facilitate effective learning about certain topics of software project management for computer science students. This was done by providing a scenario-driven interactive single-learner environment that can be accessed through the internet by using a standard web-browser. The training module used in the study is composed of course material on project planning and control. The core element of the training module is a set of interrelated project management models, represented by a simulation model that was created by using the System Dynamics (SD) simulation modelling method [5]. This model simulates typical behaviour of software development projects.

In order to investigate the effectiveness of computer-based training in the field of software project management using a SD simulation model, a controlled experiment applying a pre-test-post-test control group design was conducted. The subjects who were willing to participate in the experiment had to pass two tests, one before the training session (pre-test) and one after the training session (post-test). The effectiveness of the training was then evaluated by comparing within-subject post-test to pre-test scores, and by comparing the scores between subjects in the experimental group, i.e. those who used the SD model, and subjects in the control group, i.e. those who used a conventional project planning model instead of the SD model. In the study, the well-known COCOMO model [2] was used by the control group since this model is used in many industrial software organisations.

The various possibilities of conducting a training session is described as a three-layered structure. The first layer defines the learning goal, i.e. software project management with focus on project planning and control. The second layer defines the type of project planning model used in the training session, i.e. COCOMO model versus SD simulation model. Finally, the third layer defines the learning mode as another dimension to characterise the training session, i.e. inclusion or exclusion of a web-based interactive role-play. The combination of the distinctions made in layers two and three yield four different treatments. Our empirical investigations compare the effectiveness of two of them: T_A (Group A: SD model-based learning with web-based interactive role-play scenario) verses T_B (Group B: standard COCOMO-based learning without web-based interactive role-play). The following dimensions were used to characterise "effectiveness" of the training session:

1. Interest in software project management issues ("Interest").
2. Knowledge about typical behaviour patterns of software development projects ("Knowledge").
3. Understanding of "simple" project dynamics ("Understand simple").
4. Understanding of "complex" project dynamics ("Understand complex").

In the study, these four dimensions were represented respectively by dependent variables Y.1 to Y.4.

2.1 Experimental Hypotheses

The two hypotheses of the experiment were stated as follows:

1. There is a positive learning effect in both groups (A = experimental group, B = control group). Using the notations in Table 1, this can be formulated as:
   - $score_{post}(Y.i; A) > score_{pre}(Y.i; A)$, for i = 1...4
   - $score_{post}(Y.i; B) > score_{pre}(Y.i; B)$, for i = 1...4
2. The learning effect in group A is higher than in group B, either with regard to the performance improvement between pre-test and post-test (relative learning effect), or with regard to post-test performance (absolute learning effect). The absolute learning effect is of interest because it may indicate an upper bound of the possible correct answers depending on the type of training (A or B). This expectation can be formulated as follows:
   - $score_{diff}(Y.i; A) > score_{diff}(Y.i; B)$, *for* $i = 1, \ldots, 4$
   - $score_{post}(Y.i; A) > score_{post}(Y.i; B)$, *for* $i = 1, \ldots, 4$

Table 1. Terms and definitions of the hypotheses

Term	*Definition*
$score_{pre}$ (Y.i; X)	Pre-test scores for Y.i (i = 1,..., 4) of subjects in group X (X = A or B).
$score_{post}$ (Y.i; X)	Post-test scores for Y.i (i = 1,..., 4) of subjects in group X (X = A or B).
$score_{diff}$ (Y.i; X)	Difference scores for Y.i (i = 1,..., 4) of subjects in group X (X = A or B). $score_{diff}(Y.i; X) = score_{post}(Y.i; X) - score_{pre}(Y.i; X)$

Note that it is not expected that both relative and absolute learning effect will always occur simultaneously. This reflects on the fact that higher relative learning effects in group A compared to group B are less likely to occur when pre-test scores of group A are significantly higher than those of group B. Similarly, higher absolute learning effects in group A compared to group B are less likely to occur when pre-test scores of group A are significantly lower than those of group B. Standard significance testing was used to analyse expectations. The null hypotheses were stated as follows:

- $H_{0,1}$: There is no difference between pre-test scores and post-test scores within group A and group B, i.e.
  - $score_{pre}(Y.i; A) = score_{post}(Y.i; A)$ *and*
  - $score_{pre}(Y.i; B) = score_{post}(Y.i; B)$ *for all* $i = 1, \ldots, 4$.
- $H_{0,2a}$: There is no difference in relative learning effectiveness between group A and group B, i.e.
  - $score_{diff}(Y.i; A) = score_{diff}(Y.i; B)$ *for all* $i = 1, \ldots, 4$.
- $H_{0,2b}$: There is no difference in absolute learning effectiveness between group A and group B, i.e.
  - $score_{post}(Y.i; A) = score_{post}(Y.i; B)$ *for all* $i = 1, \ldots, 4$.

2.2 Subjects

Our replication study (henceforth the Reading experiment) was conducted during a university term with 11 second year undergraduate students doing the software engineering module (out of a total of 180). One of the authors is the instructor of the module and he invited the students to participate in the experiment. A total of 30 students responded but only 11 turned up on the day of the experiment. Finally, the treatment was divided randomly among the students, just depending on which computer they selected. The personal characteristics of the subjects have been summarised in Table 2. We have also included the personal characteristics of the subjects in the original experiment and those of the first replication. The original experiment was conducted at the University of Kaiserslautern in Germany (henceforth KL experiment) and its first replication was performed at the University of Oulu in Finland (Oulu experiment). The subjects of the KL experiment were graduate computer science students enrolled in the advanced software engineering class. In the Oulu experiment, the subjects were graduate and post-graduate.

Table 2. Personal characteristics of the subjects

	KL students	*Oulu*	*Reading*
Average age [years]	27.0	31.3	23.20
Share of women	11 %	50 %	9%
Share of subjects majoring in Computer Science	100 %	67 %	82%
Preferred learning style(s):			
• Reading (with exercise)	89 %	33 %	18%
• wb-based training	11 %	8 %	33%
• in-class lecture (with exercise)	22 %	25 %	72%
• working group (with peers)	33 %	42 %	81%
Opinion about most effective learning style(s):	- not asked -		
• reading (with exercise)		25 %	18%
• web-based training		17 %	33%
• in-class lecture (with exercise)		33 %	72%
• working group (with peers)		67 %	81%

2.3 Treatments

The training sessions for both the groups was composed of the following four scenario blocks; they were the same as the original experiment [10]:

- Block 1 - PM Introduction: General introduction into the main tasks of software project managers and the typical problems they have to solve with regard to project planning and control.
- Block 2 - PM Role Play: Illustration of common project planning problems on the basis of an interactive case example in which the trainee takes over the role of a fictitious project manager.
- Block 3 - PM Planning Models: Presentation of basic models that help a project manager with planning tasks, namely a process map, and a predictive model for effort, schedule and quality.
- Block 4 - PM Application Examples: Explanation on how to apply the planning models on the basis of examples that are presented in the form of little exercises.

Treatment of the Experimental Group. The experimental group passed all scenario blocks. The SD model was used as the predictive model in scenario blocks 3 and 4. In addition, the SD model was integrated into the interactive role-play offered by scenario block 2. The SD model used in the training session consists of five interrelated sub-models [10].

Scenario block 2 (PM Role Play) has been designed to help the trainee understand the complex implications of a set of empirically derived principles that typically dominate software projects conducted according to the waterfall process model. The set of principles used in the block scenario was distilled from the top 10 list of software metric relationships published by Boehm [3]. In order to make the trainee understand the implications of these principles (and their combinations), a role-play is conducted in which the trainee takes the role of a project manager who has been assigned to a new development project. Several constraints are set, i.e. the size of the product and its quality requirements, the number of software developers available, and the project deadline. The first thing to do for the project manager (in order to familiarise with the SD simulation model) is to check whether the project deadline is feasible under the resource and quality constraints given. Running a simulation does this check. From the simulation results, the project manager learns that the deadline is much too short. Now, the scenario provides a set of actions that the project manager can take, each action associated with one of the principles and linked to one of the model parameters. Soon the project manager learns that his department head does not accept all of the proposed actions (e.g. reducing the product size or complexity). Depending on the action the project manager has chosen, additional options can be taken. Eventually, the project manager finds a way to meet the planned deadline, e.g. by introducing code and design inspections (one of the principles discussed by Boehm [3]). The role-play is arranged in a way that the project manager can only succeed when combining actions that relate to at least two of the principles of Boehm. At the end of the role-play, a short discussion of the different possible solutions is provided, explaining the advantages and disadvantages of each.

Treatment of the Control Group. The control group passed only scenario blocks 1, 3, and 4. The predictive model used in scenario blocks 3 and 4 was the intermediate COCOMO model [2].

Differences between Initial Experiment and Replications. Since almost all of the participants of the KL experiment stated that they did not have enough time for working through the materials, more time was reserved for the treatment during our replication study. While the initial experiment was conducted on two days with one week of time in between, the Reading experiment was conducted on one single day. The Oulu experiment also adopted the same changes.

2.4 Experimental Design

For evaluating the effectiveness of a training session using SD model simulation, a pre-test-post-test control group design was applied. This design involves random assignment of subjects to an experimental group (A) and a control group (B). The subjects of both groups pass a pre-test and a post-test. The pre-test measures the performance of the two groups before the treatment, and the post-test measures the

performance of the two groups after the treatment. By studying the differences between the post-test and pre-test scores of the experimental group and the control group, conclusions can be drawn with respect to the effect of the treatment (i.e. the independent variable of the experiment) on the dependent variable(s) under study.

2.5 Experimental Variables

During the experiment, data for three types of variables are collected. The dependent variables (Y.1 ... Y.4) have been discussed earlier. The lone independent variable (X.1) is the type of treatment. Z1 (Personal background), Z2 (Time consumption/time need) and Z3 (Session evaluation) are the three variables that represent potentially disturbing factors. The conceptual model assumes that the independent variable and the disturbing factors affect the dependent variables [12].

Independent Variables. The independent variable X.1 can have two values: T_A, applied to the experimental group A, and T_B applied to the control group B. The difference between T_A and T_B is basically determined by two factors. The first factor is the training scenario according to which the course material is presented. The second factor is the planning model that is used to support software project management decision-making. With regard to the scenario, the main difference consists in the application of scenario block PM Role Play for treatment T_A. As a consequence of performing the scenario block PM Role Play, interaction of the trainee with the training module will be high whereas treatment T_B will only trigger low interaction of the trainee with the training module. With regard to the model that is used during the training session, treatment T_B exclusively relies on a black-box model providing point estimates, such as COCOMO. In contrast to this, use of SD model in T_A facilitates insights into behavioural aspects of software projects.

Dependent Variables. The dependent variables Y.1, Y.2, Y.3, and Y.4 are determined by analysing data collected through questionnaires that all subjects have to fill in, the first time during the pre-test, and the second time during the post-test. The value of each dependent variable will then be equal to the average score derived from the related questionnaire. The contents of the questionnaires are as follows:

- Y.1 (“Interest”): Questions about personal interest in learning more about software project management.
- Y.2 (“Knowledge”): Questions about typical performance patterns of software projects. These questions are based on the empirical findings and lessons learned summarised in Boehm’s top 10 list of software metric relations [3].
- Y.3 (“Understand simple”): Questions on project planning problems that require simple application of the provided PM models, addressing trade-off effects between no more than two model variables.
- Y.4 (“Understand complex”): Questions on project planning problems addressing trade-off effects between more than two variables, and questions on planning problems that may require re-planning due to alterations of project constraints (e.g. reduced manpower availability, shortened schedule, or changed requirements) during project performance.

Disturbing Factors. The disturbing factors remain the same as the original experiment. The contents of the respective questionnaires are as follows:

- Z.1: Questions about personal characteristics (age, gender), university education (number of terms, major, minor), practical software development experience, software project management, literature background and preferred learning style.
- Z.2: Questions on actual time consumption per scenario block, and on perceived time need.
- Z.3: Questions on personal judgement of the training session (subjective session evaluation).

Table 3. Time distribution for various stages during the experiments

	KL Experiment	*Oulu/ Reading*
Introduction to experiment	5'	5'
Background characteristics	5'	5'
Pre-test		
Interest	3'	5'
Knowledge about empirical patterns	5'	5'
Understanding of simple project dynamics	10'	10'
Understanding of complex project dynamics	12'	15'
Introduction to treatments	5'	5'
Random assignment of subjects to groups	5'	5'
Treatment	45'	80'
Post-test		
Interest	3'	5'
Knowledge about empirical patterns	5'	5'
Understanding of simple project dynamics	10'	10'
Understanding of complex project dynamics	12'	15'
Time need & subjective session evaluation	5'	10'
Total	130'	180'

2.6 Experimental Procedure

The duration of the phases in the KL and the Reading experiment are as they are in Table 3. The Oulu experiment also had similar plans as that of Reading. They are similar because both followed the changes suggested by the KL investigators. After a short introduction during which the purpose of the experiment and general organisational issues were explained, data on the background characteristics (variable Z.1) was collected. Then the pre-test was conducted and data on all dependent variables (Y.1 through Y.4) was collected, using questionnaires. Following the pre-test, a brief introduction into organisational issues related to the treatments was given. After that, the subjects were randomly assigned to either the experimental or control group. Then each group underwent its specific treatment. After having concluded their treatments, both groups passed the post-test using the same set of questionnaires as during the pre-test, thus providing data on the dependent variables for the second time. Finally, the subjects got the chance to evaluate the training session by filling in another questionnaire, providing data on variables Z.2 and Z.3. The time frames reserved for passing a certain step of the schedule was identical for the experimental and control groups. However, more time was reserved during the replication as

compared to the initial experiment. This was done in accordance with the recommendations of the original experiment [11]. Of the eleven students participating in the first replication, 6 were assigned randomly to the experimental group (A), and 5 to the control group (B).

2.7 Data Collection Procedure

The data collection procedure of our study remains same as the original study. We briefly discuss it here. The raw data for Y.1 to Y.4 were collected during pre-test and post-test with the help of questionnaires. Each answer in the questionnaire is mapped to the value range R = [0, 1] assuming equidistant distances between possible answers, i.e. "fully disagree" is encoded as "0", "disagree" as "0.25", "undecided" as "0.5", "agree" as "0.75", and "fully agree" as "1".

The raw data for disturbing factors were collected before pre-test (Z.1) and after post-test (Z.2 and Z.3). In order to determine the values of factor Z.1 ("Personal background") information on gender, age, number of terms studied, subjects studied (major and minor), personal experience with software development, and number of books read about software project management was collected. The values for factor Z.2 are normalised average scores reflecting the "time need" for reading and understanding of the scenario blocks 1, 3, and 4, for familiarisation with the supporting tools, and for filling in the post-test questionnaire. For group A, the variable Z.2' includes also scores related to scenario block 2. The values for factor Z.3 ("Session evaluation") are based on subjective measures reflecting the quality of the treatment.

2.8 Data Analysis Procedure

In a first step of the statistical analysis a t-test was used to investigate the effect of the independent variable X.1 on the dependent variables Y.1 to Y.4. For testing hypothesis $H_{0,1}$, a *one-way paired t-test* was used. For testing hypotheses $H_{0,2a}$ and $H_{0,2b}$, the *one-way t-test for independent samples* was used [13]. A prerequisite for applying the t-test is the assumption of normal distribution of the variables in the test samples. Checking for the normality assumption showed that no normal distribution of the variables in the test samples could be assumed. On the other hand, the outlier analysis showed that all data points lie within the range of ±2 standard deviations around the samples' means,.

Researchers should perform a power analysis [4] before conducting a study to ensure the experimental design will find a statistically significant effect if one exists. The power of a statistical test is dependent on three different components: significance level α, the size of the effect being investigated, and the number of subjects. Low power will have to be considered when interpreting non-significant results. Usually, the commonly accepted practice is to set $\alpha = 0.05$. Since sample sizes were rather small in the initial experiment and in our replication, and no sufficiently stable effect sizes from previous empirical studies were known, it was decided to set $\alpha = 0.1$. This was also the case with the Oulu experiment. *Effect size* is expressed as the difference

between the means of the two samples divided by the root mean square of the variances of the two samples [13]. For this exploratory study, effects where $\gamma \geq 0.5$ are considered to be of practical significance. This decision was made on the basis of the effect size indices proposed by Cohen [4].

3 Experimental Results

Data was collected from 11 subjects. The column "Pre-test scores" of Table 4 shows the calculated values for mean, median, and standard deviation of the raw data collected during the Reading experiment. The column "Post-test scores" shows the calculated values for mean, median, and standard deviation of the raw data collected during the post-test. The column "Difference scores" shows the calculated values for mean, median, and standard deviation of the differences between post-test and pre-test scores. Possible reasons for unexpected outcomes are discussed in a later Section.

Table 4. Scores of dependent variables

	Pre-test scores				*Post-test scores*				*Difference scores*			
Group A (6 subj.)	*Y.1*	*Y. 2*	*Y. 3*	*Y. 4*	*Y.1*	*Y. 2*	*Y. 3*	*Y. 4*	*Y.1*	*Y.2*	*Y. 3*	*Y. 4*
Mean($score_{pre}$(A))	0.63	0.40	0.33	0.22	0.68	0.80	0.81	0.61	0.05	0.40	0.48	0.39
Median($score_{pre}$ (A))	0.55	0.40	0.29	0.17	0.70	0.80	0.79	0.58	0.05	0.40	0.57	0.42
Stdev($score_{pre}$ (A))	0.22	0.18	0.20	0.23	0.22	0.13	0.12	0.14	0.10	0.18	0.27	0.33
Group B (5 subj.)	*Y.1*	*Y. 2*	*Y. 3*	*Y. 4*	*Y.1*	*Y. 2*	*Y. 3*	*Y. 4*	*Y.1*	*Y.2*	*Y. 3*	*Y. 4*
Mean($score_{pre}$ (B))	0.69	0.52	0.29	0.43	0.75	0.68	0.60	0.53	0.06	0.16	0.31	0.10
Median($score_{pre}$(B))	0.70	0.60	0.29	0.33	0.70	0.80	0.57	0.67	0.05	0.00	0.43	0.00
Stdev($score_{pre}$ (B))	0.17	0.23	0.14	0.22	0.16	0.18	0.12	0.22	0.07	0.26	0.23	0.30

Table 5 shows the calculated values for mean, median, and standard deviation of the raw data collected for the disturbing factors. In the initial experiment, there could be observed a difference between students in the experimental group (A) and the control group (B) regarding experience with software development (Z.1). This difference was neither observed in the Reading experiment nor in the Oulu experiment.

Table 5. Scores of Disturbing factors

Group A	*Z.1*	*Z.2*	$Z.2_{B2}$	*Z.3*	$Z.3_{B2}$
Mean $_{df}$	0.36	0.17	0.14	0.51	0.59
Median $_{df}$	0.33	0.00	0.00	0.50	0.58
StDev $_{df}$	0.06	0.27	0.22	0.17	0.10
Group B	*Z.1*	*Z.2*		*Z.3*	
Mean $_{df}$	0.3	0.05		0.65	
Median $_{df}$	0.33	0.00		0.63	
StDev $_{df}$	0.07	0.11		0.14	

3.1 Hypothesis $H_{0,1}$

Table 6 shows the results of testing hypothesis $H_{0,1}$ using a *one-way tailed paired t-test* is used to compare the means of the pre-test and post-test scores within each

group (A and B). Column one represents the dependent variable, column two the effect size, column three the degrees of freedom, column four the t-value of the study, column five the critical value for $\alpha = 0.10$ (the t-value has to exceed the critical value to be statistically significant), and column six provides the associated p value. By looking into the fourth and fifth columns of Table 6, we can check that group A achieved significant results for dependent variables Y.2, Y.3 and Y.4, and group B for dependent variables Y.1 and Y.3. Therefore the null hypothesis $H_{0,1}$ can be rejected for these cases at $\alpha = 0.10$. It is to note that for group A, the dependent variable Y.2 support the direction of the expected positive learning effect in both groups, however without showing an effect size of practical significance. In addition, for group B, values for dependent variables Y.2 and Y. 4 also support the direction of the expected positive learning effect, with and without practical significance respectively. Our analysis corroborates the result of the KL and Oulu experiments as regards to the variables Y.2 and Y.3 for group A, and for Y.3 in group B.

Table 6. Result for post-test vs. pre-test

Group A (6 Subjects) *Variable*		*Df*	*t-value*	*Crit t* $_{0.90}$	*p-value*
Y.1	0.48	5	1.17	1.48	0.15
Y.2	2.24	5	5.48	1.48	0.00
Y.3	1.79	5	4.39	1.48	0.00
Y.4	1.19	5	2.91	1.48	0.02
Group B (5 Subjects) *Variable*		*Df*	*t-value*	*Crit t* $_{0.90}$	*p-value*
Y.1	0.92	4	2.06	1.53	0.05
Y.2	0.61	4	1.37	1.53	0.12
Y.3	1.34	4	2.99	1.53	0.02
Y.4	0.33	4	0.74	1.53	0.25

3.2 Hypothesis $H_{0,2a}$

Table 7 shows the results of the testing hypothesis $H_{0,2a}$ using a *one-tailed t-test for independent samples.* For significance level $\alpha = 0.1$, the score difference between post-test and pre-test for the dependent variables Y.2 and Y.4 are significantly larger in group A as compared to group B, and thus hypothesis $H_{0,2a}$ can be rejected for these variables. It can also be noted that the values of variable Y.3 support the direction of the expected relative learning effect, showing a medium to large effect size. The value for variable Y.1 does not even support the direction of the expected relative learning effect. We achieve significant result for variable Y.4; however, for the KL and the Oulu experiments, the value of Y.4 does not support the direction of the hypotheses.

Table 7. Result for performance improvement

Group A versus B *Variable*		*df*	*t-value*	*Crit t* $_{0.99}$	*p-value*
Y.1	-0.11	9	-0.18	1.38	0.57
Y.2	1.10	9	1.81	1.38	0.05
Y.3	0.64	9	1.06	1.38	0.16
Y.4	0.91	9	1.51	1.38	0.08

3.3 Hypothesis $H_{0,2b}$

Table 8 shows the results of testing hypothesis $H_{0,2b}$ using a *one-tailed t-test for independent samples*. For significance level = 0.1, the post-test scores of variable Y.3 are significantly larger for the experimental group A as compared to the control group B, and thus hypothesis $H_{0,2b}$ can be rejected for this variable. It can also be noted that the values of variables Y.2 and Y.4 support the direction of the expected absolute learning effect, however, only with a small effect size. The values for variable Y.1 does do not even support the direction of the expected absolute learning effect. As regards to Y.3, we achieved a significant result. In both the KL and Oulu experiments, the values of Y.3 even did not support the direction of the hypothesis.

Table 8. Results for Post-test performance

Group A versus B *Variable*		*df*	*t-value*	*Crit.* $t_{0.90}$	*p-value*
Y.1	-0.35	9	-0.57	1.38	0.71
Y.2	0.21	9	1.30	1.38	0.11
Y.3	0.13	9	2.93	1.38	0.01
Y.4	0.28	9	0.73	1.38	0.24

3.4 Qualitative Results

In addition to filling in the pre-test and post-tests and the questionnaires about potential disturbing factors, the participants of the case studies had the chance of making comments or improvement suggestions, and could raise issues or problems that they encountered during the treatments. Comments and statements mainly supported the findings of the quantitative analyses. Positive comments about its usefulness as a whole were made in both groups. Negative comments or problem statements mainly addressed the difficulty of understanding the whole amount of information (both groups) and mainly the structure of the complex system dynamic model in the experimental group. In a lone case in group A, there was a concern with the lack of time for getting acquainted with the tools and for working through the treatments. This was an important objection in the KL experiment, and it was a relatively minor issue with the Oulu experiment.

3.5 Analysis, Summary, and Discussion

Table 9 shows the main results of all the 3 experiments as regards to the testing of null hypotheses $H_{0,2a}$ and $H_{0,2b}$, respectively. Meta-analysis techniques [8] are used for comparing and combining results from different studies. The benefit of meta-analytic procedures is that by combining the results of a number of studies, one can increase the power of the statistical analysis. This enables one to identify effects that could escape the scrutiny in a single study with much lower statistical power. Meta-analytic techniques are based either on p-values or effect sizes. To make a step in this direction and include both, p-values as well as effect sizes, in the discussion, the hypothesis testing results of each study were classified as follows:

- Statistical significance (sta. sig.): null hypothesis could be rejected at significance level $\alpha = 0.1$.
- Practical significance (pract. sig.): null hypothesis could not be rejected but effect size $\gamma \geq 0.5$.
- Positive effect (+): no practical significance could be observed but $\gamma > 0$. The number in parentheses indicates how many subjects would have been needed to achieve statistical significance with the given effect size.
- No effect or negative effect (-): t-value ≤ 0.

Table 9 shows that null hypothesis $H_{0,1}$ could only be rejected in all experiments for variable Y.3 (both for the experimental and the control groups). In addition, for the experimental group, $H_{0,1}$ could be rejected in all cases for Y.2 and in one case for Y.1. For the control group, $H_{0,1}$ could be rejected in two cases for Y.1 too.

Table 9. Summary of individual results of $H_{0,1}$

Variables	*Group A*			*Group B*		
	KL	*Oulu*	*Reading*	*KL*	*Oulu*	*Reading*
Y.1	stat. sig.	+	+	-	stat. sig.	stat. sig.
Y.2	stat. sig.	stat. sig.	stat. sig.	+	–	+.
Y.3	stat. sig.	stat. sig.	stat. sig.	stat. sig.	stat. sig.	stat. sig.
Y.4	+	–	stat. sig.	+	–	+

Table 10 shows that null hypothesis $H_{0,2a}$ could only be rejected in all cases for variables Y.2. A significant result was achieved in one case for variable Y.1. Regarding null hypothesis $H_{0,2b}$ statistical testing yielded statistically and practically significant results for variable Y.2. In the KL and the Oulu experiments, there is no indication that the experimental group performs better than the control group with regard to understanding of simple and complex project dynamics (variables Y.3 and Y.4). However, in the Reading experiment, a better performance for the experimental group has been obtained for these variables. The role-play scenario explicitly states project management principles that were not so clearly specified for the control group. On the other hand, this task imposed additional time pressure on the subjects in the experimental group, which might have resulted in low scores for questions related to dependent variables Y.3 and Y.4 in the KL and the Oulu studies. This was not observed in the Reading experiment.

There is a major difference between Reading experiment and the other two in relation to the experience of the subjects. Reading students were in the middle of a course on Software Engineering, and only a few weeks before they were introduced to principles of project management. We believe that such issues related to project management were fresh in their mind, and that might have been the reason why the results were better as regards to Y.4 and hypotheses $H_{0,1}$ and $H_{0,2a}$.

Table 10. Results of $H_{0,2}$

Variables	*KL Experiment*		*Oulu*		*Reading*	
	$H_{0,2a}$	$H_{0,2b}$	$H_{0,2a}$	$H_{0,2b}$	$H_{0,2a}$	$H_{0,2b}$
Y.1	stat. sig.	+(1000)	–	+(56)	–	+
Y.2	Pract. sig.	stat. sig.	stat. sig.	stat. sig.	stat. sig.	pract. sig.
Y.3	–	–	–	–	pract. sig.	stat. sig.
Y.4	–	–			Stat. sig	+

4 Threats to Validity

Construct Validity. It is the degree to which the variables used in the study accurately measure the concepts they purport to measure. The related issues remain same as the KL experiment [11]. We highlight the points here.

1. The mere application of a SD model might not adequately capture the specific advantages of SD models over conventional planning models
2. Interest in a topic and evaluation of a training session are difficult concepts that have to be captured with subjective measurement instruments.
3. It is difficult to avoid "unfair" comparison between SD models and COCOMO, because SD models offer features that per definition are not available for COCOMO. Since exclusively subjects of the experimental group perform scenario block 2, subjects of the control group might be disadvantaged.

Internal Validity. It is the degree to which conclusions can be drawn about the causal effect of the independent variable on the dependent variables. Potential threats include selection effects, non-random subject loss, instrumentation effect, and maturation effect. These issues also remain same as the original experiment:

1. A selection effect was avoided by random assignment of subjects.
2. Non drop-out of subjects has been avoided by the experimental design.
3. The fact that the treatments of group A and B were different in the number of scenario blocks involved and, as a consequence, in the time available to perform each scenario block, may have induced an instrumentation effect.

External Validity. It is the degree to which the results of the research can be generalised to the population under study and other research settings. The two possible threats are:

1. The subjects participating in the experiment were all students in computer science or related fields at an advanced level. Any generalisation of the results with regard to education of novice students, or even with regard to training of software professionals should be done with caution.
2. Adequate size and complexity of the applied materials might vary depending on previous knowledge about SD modelling and COCOMO.

5 Conclusions and Future Work

The empirical studies presented in this paper investigated the effect of using a System Dynamics (SD) simulation model to assist software project management education of computer science students. The treatment focused on problems of project planning and control. The performance of the students was analysed with regard to four dimensions, i.e., interest in the topic of software project management (Y.1), knowledge of typical project behaviour patterns (Y.2), understanding of simple project dynamics (Y.3), and understanding of complex project dynamics (Y.4). The

findings of the current replicated study corroborates the finding the first two experiments in the sense that using SD models increase the interest of the subject in software project management and also improve a students' knowledge of typical behaviour patterns. Hence, SD models represent a viable path for learning multi-causal thinking in software project management. This was supported by the subjective evaluation of the role-play scenario involving simulation with the SD model, which received very high scores.

Future work will include its replication for two reasons. A further replication should consider the examination of cause/effect relationships. And second, each empirical study exhibits specific threats to validity, which can only be ruled out by replication. We also intend to further analyse other dynamic methods such as Bayesian networks (BN) [7]. An experiment of comparing SD and BNin the context of software project management education is a part of our future work.

Acknowledgements. The research was supported by the Spanish Research Agency (CICYT- TIC1143-C03-01) and The University of Reading.

References

[1] T. K. Abdel-Hamid and S. E. Madnick, *Software Project Dynamics: an Integrated Approach.* Prentice Hall, 1991.
[2] B. W. Boehm, *Software Engineering Economics.* Prentice-Hall, 1981.
[3] B. W. Boehm, "Industrial software metrics top 10 list", in *IEEE Software,* 1987, pp. 84-85.
[4] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences. Second Edition.* Academic Press, 1988.
[5] J. W. Forrester, *Principles of Systems.* Norwalk, CT: Productivity Press, 1961.
[6] A. K. Graham, J. D. W. Morecroft, P. M. Senge and J. D. Sterman, "Model-supported case studies for management education," *European Journal of Operational Research,* vol. 59151-166, 1992.
[7] F. V. Jensen, *An Introduction to Bayesian Networks.* UCL Press, 1996.
[8] J. Miller, "Applying meta-analytical procedures to software engineering experiments," *Journal of Systems and Software,* vol. 54, no. 1, pp. 29-39, 2000.
[9] J. D. W. Morecroft, "System dynamics and microworlds for policymakers," *European Journal of Operational Research,* vol. 35301-320, 1988.
[10] D. Pfahl, M. Klemm and G. Ruhe, "A CBT module with integrated simulation component for software project management education and training," *Journal of Systems and Software,* vol. 59, no. 3, pp. 283-298, 2001.
[11] D. Pfahl, N. Koval and G. Ruhe, "An Experiment for Evaluating the Effectivenesss of Using a system Dynamics Simulation Model in Software Project Management Education," Metrics Symposium, London, UK, 2001, pp. 97-109.
[12] D. Pfahl, O. Laitenberger, J. Dorsch and G. Ruhe, "An Externally Replicated Experiment for Evaluating the Learning Effectiveness of Using Simulations in Software Project Management Education," *Empirical Software Engineering,* vol. 8, no. 4, pp. 367-395, 2003.
[13] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures.* CRC Press, 1997.
[14] F. Shull, V. Basili, J. Carver, M. J.C, G. H. Travassos, M. Mendonca and S. Fabbri, "Replicating Software Engineering Experiments: Addressing the Tacit Knowledge Problem," ISESE, Nara, Japan, 2001.

Software Engineering Research Strategy: Combining Experimental and Explorative Research (EER)

Markku Oivo, Pasi Kuvaja, Petri Pulli, and Jouni Similä

University of Oulu
Department of Information Processing Science
P.O. Box 3000, 90014 Oulun yliopisto, FINLAND
{Markku.Oivo, Pasi.Kuvaja, Petri.Pulli, Jouni.Simila}@Oulu.fi

Abstract. In this paper a new Experimental and Explorative Research (EER) research strategy is proposed. It combines experimental software engineering with exploratory research of new technologies. EER is based on several years experience of using and developing the approach in research of future mobile applications. In large international projects explorative application research includes quite often both industrial software developers and experienced researchers. This kind of an experimental research environment resolves the subject problem found in student experiments. It also does not have the difficulties found in experimental design and control of industrial projects that are constrained by strict commercial conditions. EER strategy provides benefits for both worlds: (1) experimental software engineering research benefits from almost industry level projects that can be used as experimentation environments, and (2) future mobile telecom application research benefits from better control and understanding of the characteristics of the applications and their development methods and processes.

1 Introduction

Software engineering discipline has struggled to become an established field of engineering and/or science. An experimental component is needed in software engineering for scientific validity (Basili 1996). However, there have been debates about the nature of the field and the research methodologies used. Traditionally software engineering has used very little experimentation (Tichy et al. 1995, Zelkowitz and Wallace 1998). According to these two extensive analyses of software engineering research publications, 50-60% of the published software engineering papers were not experimentally validated (Tichy 1998). This is a very high percentage compared to some other fields like optical engineering, where the percentage of invalidated papers is merely 15 %. On the other hand some prominent researchers advocate for less need of experimentation in software engineering (Hartmanis 1994).

Research and development of new technologies in fields like mobile applications is often exploratory and constructive. Proof of concept demos or prototypes are used to explore and push the limits of technology. The building of these prototypes is often an extensive project. However, the development is usually done using more or less ad-

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 302–317, 2004.

hoc process. The end-result is that, in a positive case, the project can only demonstrate the feasibility of the technology but not much more. There is no evidence of the effectiveness of the development methods for that particular technology development nor is there any convincing empirical evidence of the excellence of the technology itself. Therefore, there is a need to combine explorative research with empirical evidence, in specific in this research area.

In this article a new research strategy is proposed. In the strategy experimental software engineering is combined with exploratory research and development of new technologies especially in developing mobile applications for the future. The strategy is based on several years of experience in using and developing this approach in the research of future mobile applications. In large international research projects the research and development includes both industrial software developers and experienced researchers. This resolves the subject problem found in student experiments and alleviates the difficulties found in experiment design and control in industrial projects. The strategy provides benefits for both worlds: (1) experimental software engineering research benefits from almost industry level projects that can be used as experimentation environments, and (2) future mobile telecom application research benefits from better control and understanding of the characteristics of the applications and their development methods and processes.

In the following section a brief introduction of those aspects of experimental software engineering that are relevant for this paper is presented. In section 3 relevant research methodologies and paradigms for explorative research are discussed. In section 4 key issues of the research of future mobile applications are discussed. Mobile applications of the future form the application domain, where most of the experiences of the report were drawn. In section 5 experiences of combining explorative and experimental approaches in a research project are presented. In section 6 the findings and the proposed new research approach are concluded into a new explorative and experimental research (EER) paradigm. In the section also future research efforts are outlined. Due to the multidisciplinary nature of this paper the review of related research is embedded in each of the sections instead of a single review section.

2 Experimental Software Engineering

Experimental research in software engineering has gained popularity in recent years. Various experimental results have been published about experiments carried out both in vivo and in vitro (Basili 1996).

Table 1. Experimentation environment (Basili 1996).

In vivo	Experiment in the field under normal conditions
In vitro	Experiment in the laboratory under controlled conditions

The principle of an experiment is shown in Fig. 1. For example, if we study the effect of a new development method on the productivity we would define the

development method, tool support etc. as independent variables; productivity as dependent variable, treatment would be the use of the old and the new method, and the people that apply the method would be called subjects (Wohlin et al. 2000).

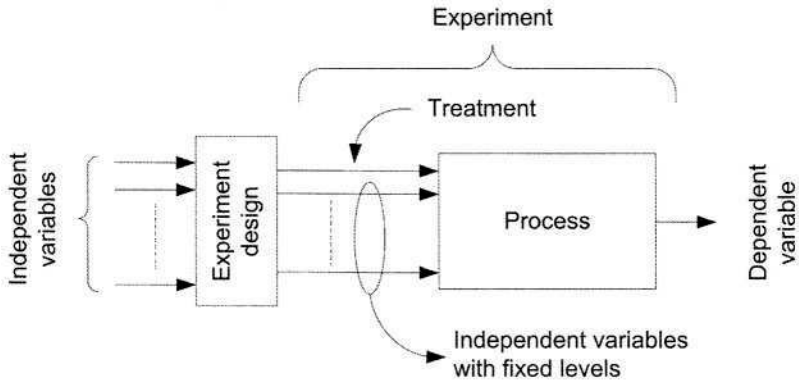


Fig. 1. Illustration of an experiment (Wohlin et al. 2000).

In an experiment we apply at least one treatment and control at least one independent variable. The focus of this paper is on experiments rather than other types of empirical research like surveys. Experiments can be categorized according to the number of teams replicating each project and the number of different projects to be analyzed in the experiment (Fig. 2).

Fig. 2. Experiments (Basili 1996)

		# Projects	
		One	*More than one*
# of Teams per Project	*One*	Single Project	Multi-Project Variation
	More than one	Replicated Project	Blocked Subject-Project

Observational studies do not include any treatments or controlled variables (Fig. 3).

Fig. 3. Observational studies (Basili 1996)

		Variable Scope	
		Defined a priori	*Not defined a priori*
# of Sites	*One*	Case Study	Case Qualitative Study
	More than one	Field Study	Field Qualitative Study

Traditionally experimental software engineering uses either students as subjects or organizes industrial experiments. Student experiments are problematic due to difficulties in scaling up the results into industrial software development. On the other hand, industrial experiments suffer from limited control of the settings of the experiment due to business constraints. Repeatability of the experiments is also problematic.

Validation method	Category	Description
Project monitoring	Observational	Collect development data
Case study	Observational	Monitor project in depth
Assertion	Observational	Use ad hoc validation techniques
Field study	Observational	Monitor multiple projects
Literature search	Historical	Examine previously published studies
Legacy	Historical	Examine data from completed projects
Lessons learned	Historical	Examine qualitative data from completed projects
Static analysis	Historical	Examine structure of developed product
Replicated	Controlled	Replicate one factor in laboratory setting
Synthetic	Controlled	Execute developed product in laboratory setting
Dynamic analysis	Controlled	Execute developed product for performance
Simulation	Controlled	Execute product with artificial data

Fig. 4. Experimental approaches (Zelkowitz and Wallace 1998).

Zelkowitz and Wallace (1998) define 12 experimental approaches (Fig. 4). The EER approach in this article covers mainly categories project monitoring, case study, field study (to certain extent), legacy, lessons learned, and replication.

3 Research Paradigms and Exploratory Research in Software Engineering, Information Systems and Other Domains

In this section a review of exploratory research is discussed in the context of software engineering and closely related information systems science. Some other scientific fields are also discussed in order to avoid re-inventing the wheel. The ideas of this section combined with experimental software engineering form the theoretical basis for the research strategy presented in this paper.

Exploratory research, which in our opinion is a required element in the research of future mobile applications, has not been recognized as mainstream research in either software engineering (SWE) or information systems (IS) research. Galliers (1992a, originally 1991) provides an excellent summary of alternative information systems research approaches listing the approaches as follows: laboratory experiments, field experiments, surveys, case studies, forecasting and futures research, simulation and game/role playing, subjective and argumentative, and action research. Järvinen (1999) provides a taxonomy consisting of: mathematical approaches, conceptual-analytical approaches, theory-testing approaches, theory-creating approaches, artifacts-building approaches and artifacts-evaluating approaches. Futures research certainly may be understood to be in the realm of exploratory research. Theory-building and artifacts-building/evaluating approaches likewise could be understood to be related to exploratory research. These approaches, however, do not grasp the gist of what is looked for in this article. Endres and Rombach (2003) in their excellent treatise of the empirical aspects of computing and the experimental approach to software engineering research do not refer to explorative aspects in research. They however note that although they have been emphasizing empirical and scientific methods in their book, they want to make it very clear that they do not want to lose the creative or

even artistic part of software and systems design. We see the same need in software engineering and information systems research.

Examinations of research undertaken during the past in IS field do not either reveal approaches that could be identified as exploratory. Hamilton and Ives (1992, originally 1982) analyzed strategies employed in 532 MIS articles published in 15 journals between 1970-79, Farhoomand (1992, originally 1987) in a similar manner performed a thematic analysis of research strategies of 536 articles published during the period 1977-85. While it is hard to say what is included under the term 'case study' in the two articles, any exploratory content seems to be minimal.

The need for exploratory research has however been noted for example by Banville and Landry (1992, originally 1989) who quote Herbert Simon in the following way: 'Science, like all creative activity, is exploration, gambling, and adventure. It does not lend itself very well to neat blueprints, detailed road maps, and central planning. Perhaps that's why it's fun.' From Hirschheim (1992, originally 1985) we can also pick Charles Peirce's notion that science should place as much emphasis on the processes of discovery as with how theories are justified.

The nature of future mobile applications necessitates therefore a look in other domains for a suitable research paradigm and strategy. We do not advocate the adoption of a paradigm based on exploratory research itself, rather the enhancement and combination of the empirical or experimental research with a strategy for finding or discovering the objects for research, which by definition do not exist yet.

Explorative research design is often proposed as the adopted research strategy when the problem is unclear, results are preliminary or tentative, and the research process is not defined or is in its early, formative stages. In many cases the references to the adopted research approach are from internal departmental working papers. Explorative research should not in our opinion be confused with theory-building approach or with case study approach (cf. e.g. Yin 1989). If a theory is sought and an explorative approach is chosen, the term theory-building suffices, likewise for case studies.

Explorative research has been quite widely used in social research. Govender (2003) used explorative research combined with descriptive research in studying adult distance learning citing Mouton and Marais (1990) as the theoretical source for exploratory research. According to Govender exploratory research aims to acquire new insights into a phenomenon rather than to collect and replicate data; to explicate central concepts; to determine priorities for further research and to develop new hypotheses about existing phenomenon. The research design of an exploratory study tends to be open and flexible.

Design research has also benefited from explorative research approach. Reymen (2001) used explorative research in her Ph.D. thesis and cited de Groot (1972) as the theoretical source. Reymen translated de Groot's approach as follows: "Explorative research is empirical research that is appropriate when the researcher is, on a relatively broad domain with little useful theory, confronted with an amount of observations or variables for which relatively few relevant facts are known. The researcher is, however, aiming at a certain type of relations, with corresponding ideas and relatively vague expectations. This aim determines which facts will be taken into account, what will be measured, and which kinds of relations will be studied. The goal of this kind of research is mainly not the ordering of facts or the creation of an overview of 'the existing', but it aims at establishing relations that are considered to be relevant for a certain theoretical or practical goal. The researcher starts from

certain expectations, from a more or less theoretical frame: He is trying to find relations in the material, but these are not defined by him in advance in the form of sharp hypotheses that can be tested; these hypotheses can thus also not yet be tested as such. Exact theory and/or hypothesis forming and testing must follow explorative research."

In marketing research whenever the focus of the research is the search for the dimensions of a question or the possible causes of facts it is referred to as "explorative". Explorative methods are mostly used to generate ideas and hypotheses and less to control assumptions. Explorative interviews stand out because the interviewer or presenter offers a wide range of possible answers thus concentrating more individually on the subject. (Skopos 2003). There are numerous examples of exploratory research also in various medicinal disciplines as well as natural sciences.

A little more rigorous approach is provided by Kleining and Witt in their two papers (Kleining and Witt, 2000 and 2001) mainly in the area of explorative psychology. The qualitative heuristic methodology applies four rules, which refer to the situation of the researcher, the topic of research, data collection and data analysis:

1. Openness of the research person
   - The researcher should be open to new concepts and change his/her preconceptions if the data are not in agreement with them
2. Openness of the research topic
   - The topic of research is preliminary and may change during the research process. It is only fully known after being successfully explored
3. Maximum variation of perspectives
   - Data should be collected under the paradigm of maximum structural variation of perspectives. There should be a multitude of different points of view, as different as possible: methods, respondents, data, time, situation, researchers etc.
4. Discovering similarities and integrating all data
   - The analysis directs itself toward discovery of similarities. It looks for correspondence similarities, accordance, analogies or homologies within these most varied sets of data and ends up discovering its pattern or structure. Completeness of analysis is required.

According to the authors the methodology has been used in social research and empirical humanities, e.g. in criminology, literature, popular music, theology, education, sick nursing and the study of national identities.

4 Exploratory Research on Mobile Applications

Research in future mobile telecom applications and services is often exploratory where prototypes and demonstrations of future applications are built in research projects, which are typically followed by demonstrations or field trial studies. These types of projects often include fairly large development projects but they often include more or less ad-hoc prototype development and they lack proper methods for understanding the development process and understanding the attributes of the new technology. In this section three research project examples will be outlined in order to

demonstrate the importance and role of exploratory part of our research strategy for the future technology development. They also demonstrate the historical evolution of the explorative ideas of research.

4.1 Virtual Prototyping Research

The roots of our exploratory research in mobile field date back to Virpi (Virtual Prototyping services for Electronics and Telecommunication Industries) project during 1996 to 1998 (Kerttula et al 1999). The research in the project was focused on shaping future product and service concepts in virtual reality design space, and advanced Virtual Reality based design environment was constructed. Besides physical appearance of products also product and environment functionality and behavioral modeling was studied. Virtual Prototyping environment allowed the project to create virtual prototypes of future telecommunication products and their associated services. Additionally, since the prototypes were virtual, the project was not bound to present-day technical limitations, but could take large conceptual and technological leaps into future product concepts. For example in the project it was possible to explore a pen-shaped cellular phone concept already 1998 although it has become possible to manufacture the electronics for such density only five years later.

4.2 Exploring Futuristic Media Phone Concept and Its Services Using Scenario-Based Design Approach

Since our research group was able to explore different product concepts in virtual design space, it became interesting to explore what a future cellular phone would be like and what might the new services be like. During the years from 1998 to 2000 an exploratory project called Cyphone (Cyphone 1998) was conducted, with a similar idea as car companies develop car design studies for their future products for automobile shows. In the Cyphone project it was assumed that future cell phones carry different multi-media formats and services and the focus was set on the concept of advanced media support. The project proposed a radical binocular shaped media phone concept and new services such as indoor and outdoor navigation and mobile tele-presence. In the Cyphone project a scenario-based design process was applied, where service ideas are first described with a storyboard, then expanded into a short multimedia movie with mock-ops, authoring and digital editing. The Cyphone movie was quite successful and has been presented hundreds of times to different audiences internationally.

4.3 Exploring Mobile Tele-presence Services

In the Paula project (from the year 1999 to 2001) (Paula 1999) mobile tele-presence services in the context of mobile virtual meetings were explored. In the project the research was focused on the design of interaction between the user and the service and explored what kind of user interfaces were best fitted for the new services. In the project the scenario-based design was applied into radically new, personal deviceless

user interfaces based on augmented reality. The interfaces were deviceless using an interface that was projected on user's hand.

5 Combining Exploratory and Experimental Research in Future Mobile Application Development Experiment

In this section a project example is presented, where exploratory and experimental research approaches were used, combined and feeding each other. The project, called Monica (www.monica.oulu.fi) was established in 1999 at the University of Oulu aiming to investigate topics on value-added service development for the 3G of mobile phones.

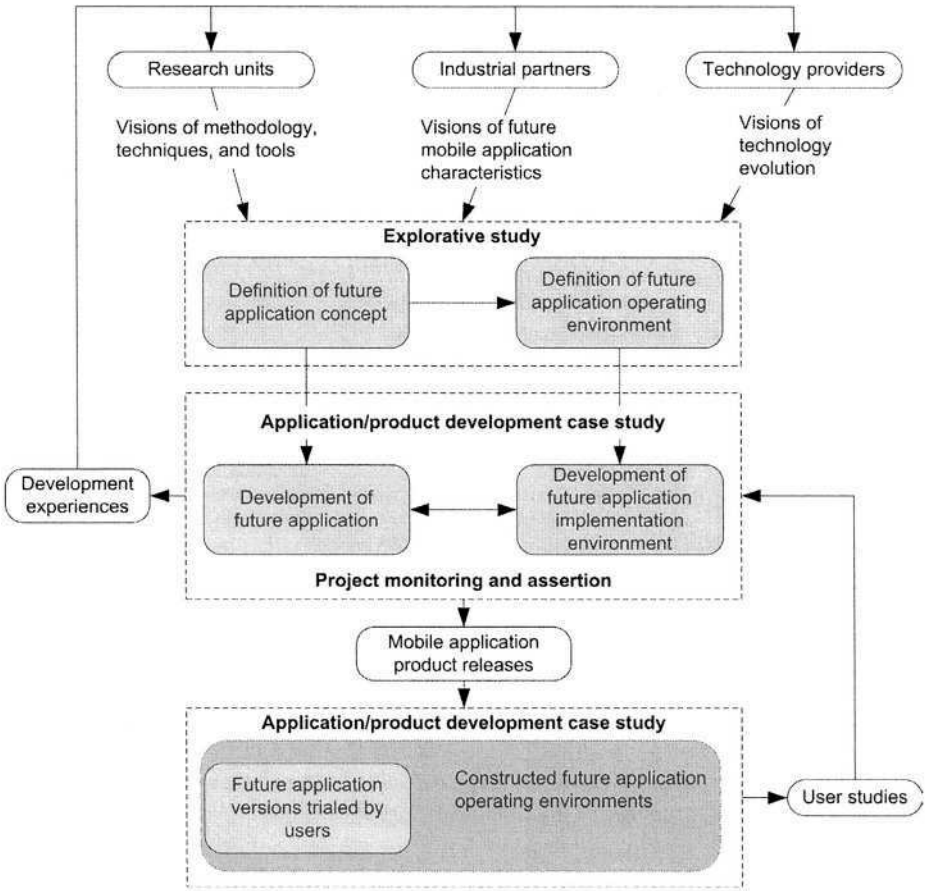


Fig. 5. MONICA research framework.

5.1 Research Context

The MONICA project was a challenging effort including industrial partners and research organizations aiming to experiment how the mobile applications might be developed in an efficient way in the near future. The time span to cover was 5 years ahead.

The entire research setting and roles of the participants are presented in the following figure (Figure 5). Industrial partners of the project represented both application development expertise (software house), technology providers (mobile telecom manufacturer), and application contents expertise (telecom provider). The visions of the features of the future applications were generated by the experts of mobile applications from the business point of view. The visions of the future technology were provided by the mobile telecom technology manufacturer. The visions of the future development approaches were defined by the software houses involved in the modern mobile telecom application business. The research organizations involved provided the insights of the exploratory and experimental research approaches and understanding of the new promising development approaches/technologies.

In Monica project both exploratory and experimental approaches were applied in parallel and communicating with each other. In parallel to the exploratory study the future application implementation was defined based on the visions from the technology providers. Based on that the future implementation environment was constructed by using current technological platforms and tools. In the following sections both exploratory and experimental development and research efforts will be presented separately in order to give insights of the efforts.

5.2 Exploratory Study on Product Concept Definition

The first step in the project was to define a vision for the product concept of the future application. The chosen case was a computerized version of *Arctic Bridge* (or *Tuppi* in Finnish), a traditional team-based card game that has its origins in Northern Finland. The game shares many similarities with bridge - its more widely known counterpart. The aim of this case was to construct a team game that would follow the idea of the original real world version. The rich interaction experiment was constructed in the form of 3D representation of the game, its players and the corresponding thematic environment. The game application was chosen as the application as it would push the limits of the mobile application especially to get as rich interaction as possible to get the game interesting.

The starting point of the work was to follow the traditional multimedia, game and film design processes from synopsis through media editing to programming. Although the system under development did not fit into the traditional multimedia or film context in terms of being purely presentational material, the basic process was considered to be close enough to start with. The overall research strategy was highly iterative and experimental. While the role of the application and technology development was exploratory. The main phases and the experimental context in the application development process are briefly described in the following (although the phases are here described in a certain order, this does not correspond to the realization of the work):

- *Synopsis* is the first written outline of the production and describes the most important aspects of the design. The approach and selected methods are decided at this stage. Rich interaction is visible as a design philosophy.
- *Background & Context Research* provides deep understanding of the background that informs better of the context and theme under development in the production.
- *Script Writing* is the main creative effort that produces the manuscript to be implemented.
- *Visualization and Concept Art* make the written descriptions visual.
- *Interaction Design* starts from the interactions illustrated by the manuscript and extends to the user interface. In a multi-player game, the interaction design is not just about human-computer interaction and physics models, but contains also issues such as interpersonal communication and group dynamics.
- *Level Design* includes designing the spaces and places for interaction, environmental cues and affordances for actions. This phase provides a more concrete illustration and plan of the virtual environment that forms the scene for the actions. The level design overlaps with the interaction design - the environment affords certain interactions and some interactions require specific features from the environment.
- *Materials, Models & Animation* is largely about modeling and animating the avatars.
- *Media Editing* contains fine-tuning the scenes, caricatures and facial expressions of the avatars.
- *Programming* is the part of the production that makes the story alive. Most of the functionality is set alive in this phase (e.g. animation sequences, physics model, control, etc.).
- *Integration* is the phase in which all the bits and pieces are put together, to work as a functional system. Combining graphics, audio, scripts, environment and models is an example of actions that need to be completed. In a way, this task is an ongoing activity starting from the first functional prototype until the final version is ready.

The industrial partners then evaluated the developed product concept and the application development could start.

5.3 Future Application Development – Project Monitoring and Case Study

The development experiment of the MONICA project consisted of a case study of implementation of a multi-client, multi-platform mobile card game. By analyzing similar systems developed by industrial companies (legacy research), it was evident that a special approach was needed in order to be able to support a large variety of client devices.

Most of the existing solutions were built around one platform and one bearer media. There were solutions for WAP over GSM link or Java applets over IP, or dedicated systems for connected Palm PDAs, and so on. This type of approach seemed to be the most robust and successful since it may leverage the strengths of one

single platform and eliminate the risks associated with combining different, incompatible technologies. A common characteristic of the analyzed solutions was the tight coupling between the business logic of the application and the presentation layer. These formed the specific features of the experiments.

The development experiment trailed applicability of agile development approaches for the development of future mobile applications. Research in the Monica project was performed in one-week cycles. During the weekly meetings experiences and feedback of the performed research and development were discussed and documented in meeting memos. In this way the research followed a quite normal agile software development process and could be combined for the demo and future application development cycles. Also the industrial partners participated in the weekly meetings occasionally and especially when their expertise was needed. This was project monitoring type experiment.

The experiment produced experiences of the future application concept development, future application implementation environment definition, and future application development approaches. The feedback was used as input to the experimental process as the experiment was performed in short cycles. One of the results of the experimentation was that a very productive method turned out to be the method, where there was a programmer making quick changes into the application product based on the feedback from a usability expert who watched the game being played. The programmer got immediate feedback and also quickly identified the problems. After changes the result of the improved application could be seen immediately. In this kind of situation, the changes to be made into the application will not become time consuming to conduct, and the results: "small releases" can be set for end-user evaluation rapidly. This was pair-programming between the usability expert and the programmer and provided an immediate alpha-test during the development cycles.

Integration of the ideas of XP (Beck 2000), and user-centered approach to software development was successful and provided good results. According to XP new versions of the product were delivered to the customers regularly (field study with real users at the Science Center[1]). The continuing dialog between the development team and the field study team was beneficial for both parties.

5.4 Observational Field Study on Mobile Application Product Usability

In the Monica project, the usability evaluation was a continuous activity throughout the design process. The developed releases of the future mobile application were implemented in the Science Centre Tietomaa for public use. The aim was to get the future application into use as early as possible, to find usability problems, to understand the reasons for the problems, and to give feedback and new ideas to programmers in order to assure that the development of new product releases will evolve into the right direction.

The main purpose of the field study was to ensure the usability of the user interface to guarantee satisfying playing experience. The main data collection method used was user-based observation complemented with individual interviews and questionnaires filled by the users. The user-based approach involved end-users interacting with the

[1] Science Centre Tietomaa, Oulu.

system and playing the card game. According to Parlangeli, Marchigiani & Bagnara (1999), in contrary to other methods, data coming from this kind of evaluation is directly derived from the subjects' experience. Many other techniques such as video recording of interaction, thinking aloud, and pre- and post-test interviews were used in the controlled environment in order to get measurements of relevant variables of the usability.

Also heuristic evaluation was used by including into the experiment card game experts, who inspected the system in order to find out possible problems in the user interface. This method, having the advantages of being cost-effective and comparatively quick and easy, was used many times during the development cycle.

Measures were set for the developed product including user satisfaction, predictability of the functions of the game, and fluent gaming experience. User observations, their immediate comments, and recorded feedback and also findings of heuristic evaluations, were written into the form of short stories, that was the way to present the research results in a way that supported the development of new releases of the card game product.

6 Exploratory Experimental Research Strategy

The new EER research strategy stems from the needs of empirical research in software engineering and explorative research in mobile applications research (Figure 6).

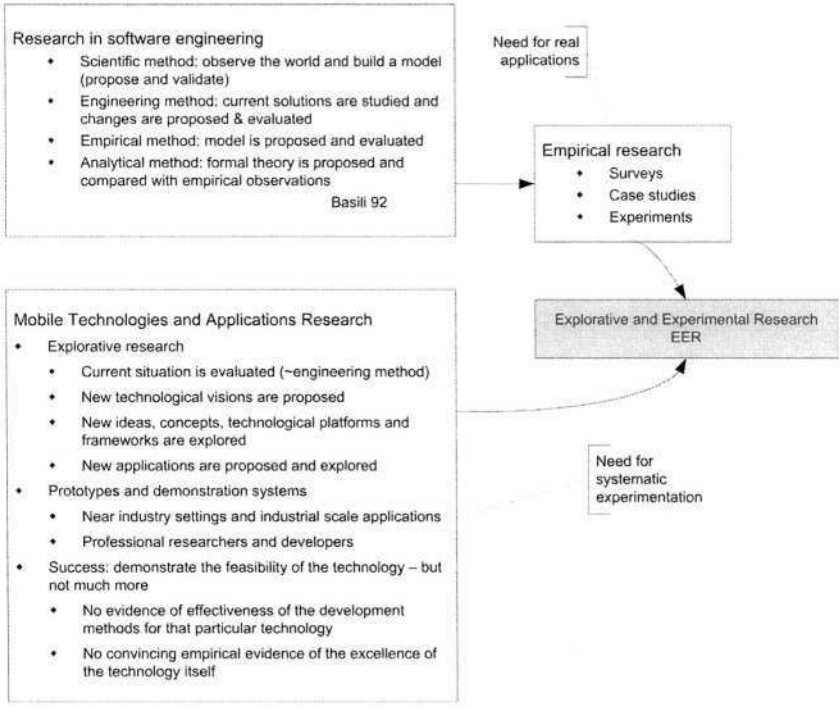


Fig. 6. Sources for Experimental and Explorative Research.

There is a constant need for realistic projects and applications when carrying out experimentation in software engineering. On the other hand there is a need (though often not recognized) for systematic experimentation in the research and development of future mobile technology and applications. By combining explorative and experimental approaches in the research and development of mobile applications we can provide help for both worlds. This is the core of the new explorative and experimental research (EER) strategy (Figure 7).

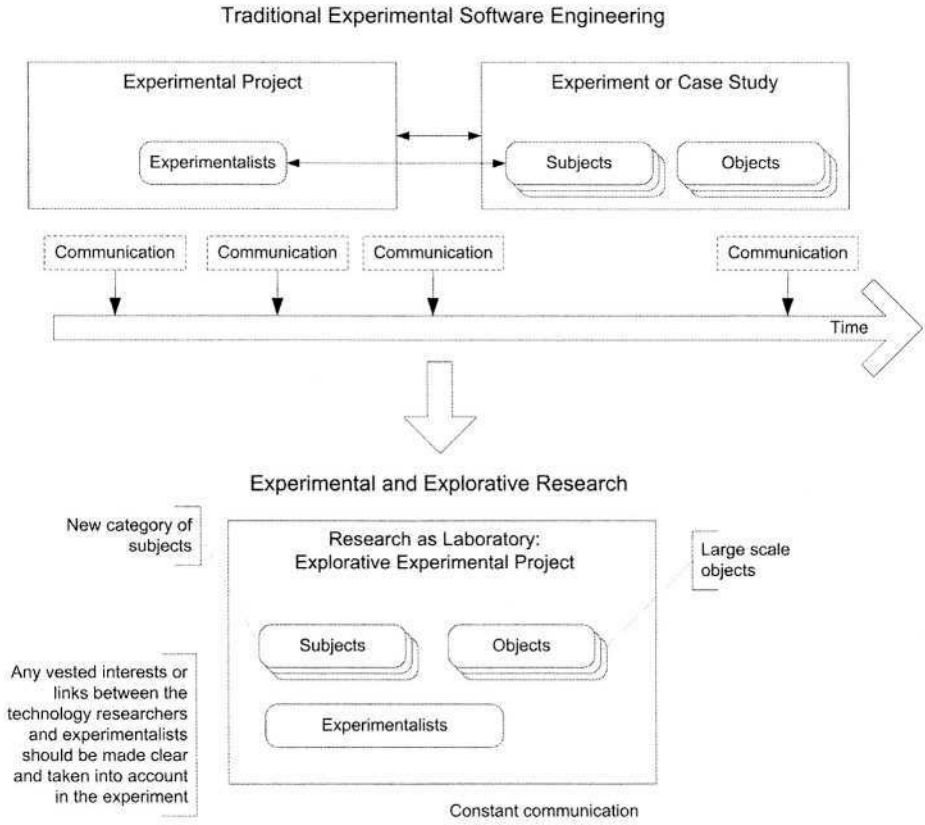


Fig. 7. EER principles and communication.

In traditional experimental software engineering (Figure 7, upper half) the experimentalists and the application developers (or the subjects) are separated. Their communication is pre-planned (points in time) and is, therefore, not natural. In EER (Figure 7, lower half) these two groups are integrated and based on that the communication is considerably improved. However, it is also important to take into account the risks of any vested interest when the groups are integrated so that the integrity of the experimentation is not jeopardized.

Explorative technology research is often iterative. Multiple versions of the prototypes are developed either based on previous versions or starting from scratch.

The nature of the research can benefit from replications of the experiment in different multiple iterations of the application development.

The experimental context is an industry like technology and application development. For data collection and analysis GQM approach is used (Basili & Rombach 1998). It provides proven methods for defining the object of study, the purpose, the focus and the point of view. The analysis follows the GQM method using both statistical techniques and human related techniques for qualitative analysis of the results in collaboration with the technology researchers and experimentalists. The collaboration of these two groups is essential to avoid false interpretations of the measurement results. Human interpretation of the results in addition to statistical analysis is essential (Latum et al. 1998).

7 Conclusions

A new research strategy that combines experimental software engineering and exploratory research has been presented. In an experimental exploratory research (EER) project a vision of the future applications and technology is first created. This vision may be visualized with various techniques including concept videos and virtual models. The vision is realized in a prototype in an experimental and exploratory research project. An experimental context is first designed to control, monitor, discover and learn while developing the prototypes in an exploratory development project. The exploratory prototype development is treated as an experiment and the developers are the subjects of the experiment. The objects of the experiment vary. They may be involved with development methods and processes or the underlying implementation technology.

EER is based on several research projects where the elements of the methodology have been used and developed. The main applications domain in these projects has been future mobile applications and technology.

The benefits of this strategy are twofold: (1) experimental software engineering research benefits from almost industry level projects that can be used as experimentation environments, and (2) future mobile telecom application research benefits from better control and understanding of the characteristics of the applications and their development methods and processes. The EER research makes a contribution to methodology research.

In practice both the explorative and experimental research included in a research project is performed in parallel, and their results are interactively "feeding" each other approaches. This is the way, how the results of the research project become combined as well as the conduct of the research approaches.

The research strategy proposed here will be used in new research projects concerning future mobile telecom technologies and embedded systems. The strategy itself will be also analysed more briefly in the light of scientific theories in order to confirm its explanation power.

Acknowledgements. We would like to acknowledge our colleagues in Monica, Paula and Cyphone projects, especially Jarkko Hyysalo and Tony Manninen, for their help in developing the new research approach. We would also like to thank our colleagues in the ISERN network for fruitful discussions and comments in our work.

References

1. Banville, B. and Landry, M. (1989): Can the Field of MIS be Disciplined, Communications of the ACM, 32, 1, January, pp. 48-60.
2. Banville, B. and Landry, M. (1992): Can the Field of MIS be Disciplined, in Galliers, R. (1992b) (ed.): Information Systems Research: Issues, Methods and Practical Guidelines, Blackwell Scientific Publications, Oxford, pp. 61-88.
3. Basili, V.R. (1996): The Role of Experimentation: Past, Present, Future, (keynote presentation), Proceedings of the International Conference on Software Engineering, IEEE CS Press, Los Alamitos, California, 1996.
4. Basili, V.R. and Rombach, D. (1998): The TAME project: Towards Improvement-Oriented Software Environments, IEEE Transactions on Software Engineering, vol. 14, no. 6, June 1988.
5. Beck, Kent. (2000). Extreme Programming Explained: Embrace Change. Upper Saddle River: Addison Wesley.
6. Cyphone (1998): Internet, Paula Project web page (accessed February 2002): http://paula.oulu.fi/index.php3?dir=Publications&click=Cyphone
7. Endres, A. and Rombach, D. (2003): A Handbook of Software and Systems Engineering – Empirical Observations, Laws and Theories, The Fraunhofer IESE Series on Software Engineering, Pearson Education Limited, Essex.
8. Farhoomand, A. F. (1987): Scientific Progress of Management Information Systems, Data Base, 18,4, Summer, pp. 48-56.
9. Farhoomand, A. F. (1992): Scientific Progress of Management Information Systems, in Galliers, R. (1992b) (ed.): Information Systems Research: Issues, Methods and Practical Guidelines, Blackwell Scientific Publications, Oxford, pp. 93-111.
10. Galliers, R. D. (1991): Choosing Information Systems Research Approaches, in Nissen, H-E., Klein, H.K. and Hirschheim, R.A. (1991) (eds.): Information Systems Research: Contemporary Approaches & Emergent Traditions, North-Holland, Amsterdam.
11. Galliers, R. D. (1992a): Choosing Information Systems Research Approaches, in Galliers, R. (1992b) (ed.): Information Systems Research: Issues, Methods and Practical Guidelines, Blackwell Scientific Publications, Oxford, pp. 144-162.
12. Galliers, R. (1992b) (ed.): Information Systems Research: Issues, Methods and Practical Guidelines, Blackwell Scientific Publications, Oxford.
13. Govender D. (2003): Creating an Environment Conducive to Adult Distance Learning, Available at http://www.col.org/pcf2/papers%5Cgovender.pdf, accessed 28.8.2003 at 14:09.
14. de Groot, A. (1972): Methodologie: Grondslagen van Onderzoek en Denken in de Gedragswetenschappen, Mauton, 's-Gravenhage, The Netherlands. (in Dutch) (7th ed.).
15. Hamilton, S. and Ives, B. (1982): MIS Research Strategies, Information and Management, 5, December, pp. 339-347.
16. Hamilton, S. and Ives, B. (1992): MIS Research Strategies, in Galliers, R. (ed.): Information Systems Research: Issues, Methods and Practical Guidelines, Blackwell Scientific Publications, Oxford, pp.132-143.
17. Hartmanis, J. (1994): Turing Award Lecture: On Computational Complexity and the Nature of Computer Science, Communications of the ACM, Oct. 1994, pp. 37-43.
18. Hirschheim, R. (1985): Information Systems Epistemology: An Historical Perspective, in Mumford, E., Hirschheim, R.A., Fitzgerald, G. and Wood-Harper, A.T. (1985) (ed.): Research Methods in Information Systems, Proceedings of the IFIP WG 8.2 Colloquim, 1-3 September, 1984, Manchester Business School, Elsevier, Amsterdam.
19. Hirschheim, R. (1992): Information Systems Epistemology: An Historical Perspective, in Galliers, R. (1992b) (ed.): Information Systems Research: Issues, Methods and Practical Guidelines, Blackwell Scientific Publications, Oxford, pp. 28-60.
20. Järvinen, P. (1999): On Research Methods, Opinpaja Oy, Tampere.

21. Kerttula, M., Battarbee, K., Kuutti, K., Palo, J., Pulli, P., Pärnänen, P., Salmela, M., Säde, S., Tokkonen, T., and Tuikka, T. (1999): New Product Development based on Virtual Reality Prototyping. MET-julkaisuja 13/1999. Metalliteollisuuden Kustannus Oy, Helsinki, Finland. 97 p.
22. Kleining, G. and Witt, H. (2000): Introspection as a Research Tool for an Exploratory Psychology, Workshop "Qualitative Forschung in der Psychologie", Tübingen, October 20-22, Available at http://www.uni-tuebingen.de/qualitative-psychologie/t-ws0/19-Kleining-Witt.htm, accessed 28.8.2003 at 16:44.
23. Kleining, G. and Witt, H. (2001): Discovery as a Basic Methodology of Qualitative and Quantitative Research. Forum Qualitative Sozialforschung / Forum: Qualitative Social Research [On-line Journal], 2(1). February. Available at: http://qualitative-research.net/fqs/fqs-eng.htm, accessed 28.8.2003 at 16:57.
24. Latum, F., Solingen, R., Oivo, M., Hoisl, B., Rombach, H.D, and Ruhe, G. (1998): Adopting GQM-Based Measurement in an Industrial Environment, IEEE Software, January 1998, pp. 78-86.
25. Mouton, J. and Marais, H.C. (1990): Basic Concepts in the Methodology of Social Sciences, Pretoria: Human Sciences Research Council.
26. Parlangeli O., Marchigiani E. & Bagnara S. (1999). Multimedia systems in distance education: effects of usability on learning. In Interacting with Computers 12 (1999) 37-49.
27. Paula (1999): http://paula.oulu.fi/index.php3?dir=Publications&click=Research.
28. Rambold,N., Storr, (1993): Computer Integrated Manufacturing, McGraw-Hill
29. Reymen I. (2001): Improving Design Processes through Structured Reflection: A Domain-independent Approach, SAI Report 2001/1, October, Eindhoven, Netherlands. Available at http://www.sai.tue.nl/research/SAIReports/SAI2001_1.pdf, accessed 28.8.2003 at 14:53.
30. Skopos (2003): Available at http://www.skopos-mr.co.uk/market_research/methods/specialists/exploratory_research/exploratory_research.html, accessed 28.8.2003 at 16:34.
31. Tichy, W.T. (1998): Should Computer Scientists Experiment More?, Computer, vol. 31, no. 5, pp. 32-40, May 1998.
32. Tichy W.F., Lucowicz, P., Prechelt, L., and Heinz, E.A. (1995): Experimental Evaluation in Computer Science: A Quantitative Study, Systems and Software, January 1995, pp. 1-18.
33. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., and Wesslén, A. (2000): Experimentation in Software Engineering - An Introduction, Kluwer Academic Press.
34. Yin, R.D. (1989): Case study research. Design and methods, Sage Publications, London.
35. Zelkowitz, M.V. and Wallace, D.R. (1998): Experimental Models for Validating Technology, Computer, vol. 31, no. 5, pp. 23-31, May 1998.

Automatic Measurement at Nokia Mobile Phones: A Case of SDL Based Software Development

Minna Pikkarainen[1], Matias Vierimaa[1], Hannu Tanner[1], and Raija Suikki[2]

[1] VTT Technical Research Centre of Finland, VTT Electronics, P.O.Box 1100, FIN-90571 Oulu, Finland
{firstname.lastname@vtt.fi}
[2] Nokia Mobile Phones, P.O.Box 50, FIN-90571 Oulu, Finland
{raija.suikki@nokia.com}

Abstract. Software measurement forms a sound basis for monitoring software development process and software product quality. Implementing software measurement is, however, difficult and creates several challenges and possible pitfalls. These challenges include the establishment of measurement practices, effective metrics collection and the use of collected metrics to improve development practices or product. Automated software measurement can have a significant role when adopting software metrics for systematic use. This paper is based on the results of a measurement automation project at Nokia Mobile Phones during the years of 1999-2000. This paper describes the experiences from defining metrics for controlling new SDL based development practices. The process through which the automation was implemented is described in detail and implications of the findings are discussed.

1 Introduction

The embedded software industry grows very fast and is R&D intensive. Products with embedded software have become more sophisticated and especially the software inside the products has increased in size and become more complex [3],[18]. SW development plays a major role in product development [3], and typically forms 50-80% of the overall effort.

While strict time-to-market requirements and business needs demand shorter development cycles, the software industry needs to invest heavily in the development of new methods and techniques, and in the adoption of existing best practices [14]. New development methods and technologies are often challenging to introduce to existing development process since they require learning and training, and sometimes even new ways of thinking [7]. Therefore, the benefits of these methods become visible only after some time has passed after introduction [11]. Moreover, in critical software development projects, the project staff has only a limited time to introduce new methods and even less time to effectively analyse the impacts of these new methods.

This paper is based on the results of a measurement automation project at Nokia Mobile Phones during the years of 1999-2000. The purpose of this paper is to describe the experiences from and the process of using automatic software measurement

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 318–332, 2004.

to monitor SDL (Specification Description Language) based SW development within three software development projects. It was found that while measurement automation can be difficult in a volatile software development environment, it is, however, possible and gives benefits to software developers. Detailed discussion of the measurement automation processes and findings in the SDL based environment provide much-needed experiences that can be further used by other practitioners starting and implementing their measurement automation activities.

This paper is composed as follows: the next section presents the background of this work, including a brief description of SDL based SW development, software measurement and automation, and a description of Nokia Mobile Phones and their starting point for the SDL based measurement. Third section provides a description of the actual implementation of the measurement automation process. In the fourth section the key findings and lessons learned from the case projects are discussed. The last section concludes the paper with final remarks and outlines future actions.

2 Background

When an old tool is replaced or a new tool is added to a SW development environment the impact of changes has to be compared to the previous situation [21]. New methods and technologies affect the project manager's ability to predict schedules and to control the product development. Monitoring of impacts is therefore particularly important when implementing process changes [17]. Projects that take new SW development tools and technologies into use need to pay special attention to the controllability and predictability of SW development. In this retrospect, this section describes the concepts and purpose of SDL based SW development, software measurement and automation. In addition, the background for our case is outlined.

2.1 SDL Development

The fast-growing complexity of embedded systems necessitates new design methods and tools in order to solve the problems of design, analysis, integration and validation of complex systems [9]. SDL is a widely used language and it is considered especially suitable for the development of complex telecommunication systems.

SDL is used mainly in real-time, distributed and communicating systems to increase productivity, reduce errors and improve software maintainability [16]. An SDL development process is usually divided into three phases: requirements specification, high level design, and detailed design [16]. Typically, the procedure from requirements analysis to product implementation and testing involves the following steps [8]:

- Collect the initial requirements in a text document.
- Study system analysis results, Message Sequence Charts (MSC) or/and use-cases depicting typical scenarios. The resultant classes are implemented in SDL as block diagrams and data-type definitions.

- Complete the SDL diagrams and specifications to a level where they can be simulated and checked for consistency with the system requirement analysis.
- Use verification and validation to determine whether required properties are correctly and completely implemented. When SDL design has proved consistent with the requirements, code for the application can be generated.
- Make a test suite. Tests can be generated from the SDL specification.
- Generate code to create an executable test suite that can be run in a test system.
- Run the executable tests and test the application in the target environment.

The process of defining SW requirements and functional specifications using SDL is very similar compared to other SW development methods and languages. Only the notation for documenting the analysis results is different. Instead of data flow diagrams, an SDL based specification contains message sequence charts (MSC). The purpose of these charts is to provide sufficient understanding of the specification functionality, collect items that are missing from the specification to get an idea of the maturity level of the specification and to collect information of the protocol entities as well as to gain an understanding of the needed protocol entity partitioning [8].

2.2 Software Measurement and Automation

Measurement practices introduced to volatile software development must be as transparent as possible to the software developers. In order to be effective, metrics collection, analysis and the presentation of the results should be automated (e.g., [15]; [1]; [4]). While there is a great deal of literature on implementing software metrics programs, there is less agreement, however, on what exactly contributes to the success of implementing metrics [4]. Moreover, in spite of this literature being available, companies are facing serious problems initiating even the simplest metrics programs [4], [6].

Ideally, measurement activities should be driven by clear goals in order to focus and limit the costs [9]. Collected measurement data and the results of the measurements should be based on project and organisational goals [10]. The measurement process, in our case, was based on the GQM (Goal, Question, Metric) approach as defined by Latum and Lavazza [12];[13].

The GQM approach is commonly used in software measurement. The approach defines measurement goal which is then described by set of questions. Metrics are then defined to answer the questions. GQM uses structured interviews to define questions and metrics. Collected data is later analysed and presented in feedback sessions to find out whether project has met its measurement goals.

In automatic software measurement, the measurement process consists of manual and automatic process steps [3]; [10]. Manually implemented process phases are 1) Planning the measurement automation, 2) Defining measurement goals (for the first time) and 3) Defining metrics definition (for the first time). It is possible to use tool support in the goal and metrics definition phases [13], but the actual work is highly

dependent on human decision [10]. The members of a measurement project should define:

- Data sources from where the measurement data will be collected
- Date (and time), or the trigger condition, when the data is collected
- Calculation templates for analysing the data
- Data storage where the result information will be saved.

Tasks that are repeated often should be made automatic. The tasks include, for example, the collection of the data from predefined data sources and calculation of the analysis data. However, the calculation formulas and the presentation format of the result graphs have to be predefined within the used tool.

The measurement automation process [10] starts by planning the automation, which includes defining the metrics, data sources, schedule of collection, and data analysis in detail. During the implementation of the environment for measurement automation the technical solutions for data retrieval are built and the first data analysis and presentation are carried out. Automated parts of the process are the repeated steps of collecting data, calculating the metrics, and the creation of analysis graphs. In the final phase the metrics' results and experiences are packaged.

2.3 The Case of Nokia Mobile Phones

Nokia Mobile Phones (henceforth NMP) is the world's largest mobile phone manufacturer. Nokia's comprehensive product portfolio covers all cellular protocols. As the functionality of mobile phones moves from voice-centric to multimedia, imaging, entertainment and business applications, the complexity of the software increases.

SDL and new unit testing methods were taken into use at NMP in three product development projects during the year 1999. This created needs to measure quality of design and implementation, and improve prediction and control of product development with new implementation language (i.e., SDL). A GQM based measurement project was launched as co-operation between the Technical Research Centre of Finland (henceforth VTT) and NMP.

Based on the experiences from previous measurement projects, measurement activities were seen as an important part of monitoring the status of NMP product projects. However, due to product time-to-market restrictions, the project staff had only a limited time to use for measurement activities. Thus, there was a need for a measurement project, which would specifically tackle the problems caused by the lack of time to collect the desired metrics. The purpose of the measurement project, which is the main focus of this paper, was to automate the previous measurements in the product development projects. This was expected to enhance quality and predictability of SDL based product development projects.

As a result of the previous measurements the developers at NMP had many useful measurement experiences and documents including defined and analysed metric re-

sults. Due to the need for measurement automation, it was not efficient enough to define all metrics using the GQM approach. Instead, when defining metrics, the possibility of automatic collection, analysis and presentation played a significant role.

3 Goal-Oriented Automatic Measurement in SDL Based SW Development

The purpose of this section is to describe the implementation of the automatic measurement process. The measurement automation process at NMP is shown in Fig. 1:

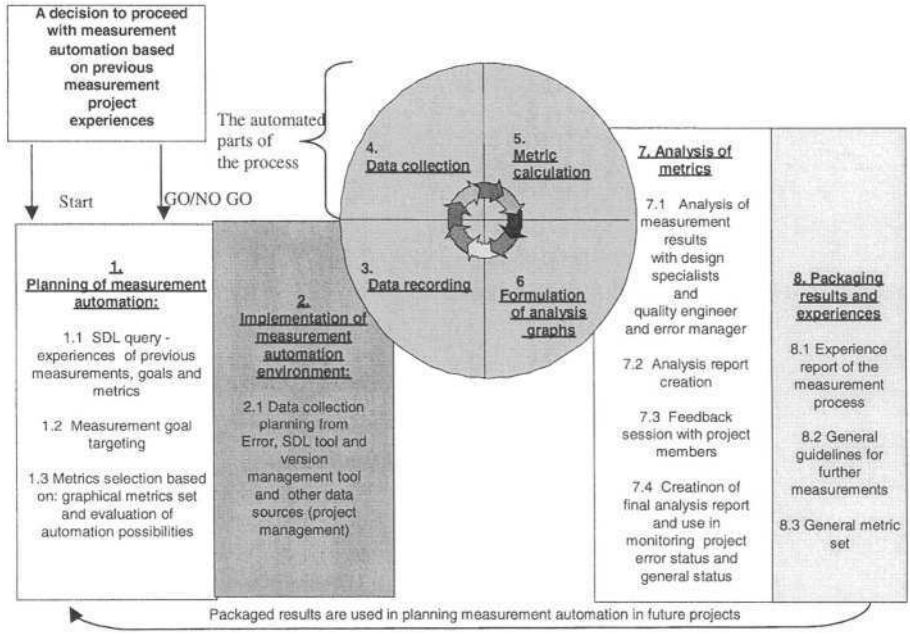


Fig. 1. Measurement automation process at Nokia Mobile Phones

Fig. 1 process is based on measurement automation process [10]. The process illustrated in Figure 1 will be used as a framework throughout the paper when the results of measurement automation are presented, analysed and discussed.

3.1 Planning of Measurement Automation

The work was started by planning the automation (Step 1, Fig. 1). Instead of the GQM interviews [12], measurement experts decided to conduct a survey to initiate the SDL based measurement (Step 1.1, Fig. 1). The questionnaire was directed to project members, project managers, error managers and quality engineers. The pur-

pose was to obtain the information to define the measurement goals and metrics. The questionnaire included questions about the respondents' expectations on SDL measurement and the most suitable focus for the measurement. Experiences from the previous GQM project were used as the basis for this action. Based on the results, the measurement goals were defined (Step 1.2, Fig. 1) together with measurement experts from VTT and the stakeholders from NMP.

Defining the goals is important and creates the basis for all measurement activities [17]; [10]. The project managers especially had difficulties in the following areas: 1) predicting the SDL module testing effort/time in the development projects and 2) evaluating SDL module quality in order to start integration. The main goals for the measurements were defined as follows:

1. To improve the predictability of SDL based SW development
2. To evaluate the quality of SDL based SW modules

The metrics were selected (Step 1.3, Fig. 1) in a meeting together with the VTT measurement experts, NMP project managers, error managers, quality engineers and the main designer. Metrics selection was based on a set of graphical presentations of metrics, which was created based on previous project experiences, results from SDL query, literature study, and evaluation of metrics automation possibilities.

After the selection meeting, the selected metrics were evaluated and subsequently updated. These metrics were then included in the NMP case project's quality plan. Examples of the goals and metrics are shown in Table 1:

Table 1. Examples of the goals and the metrics defined for the case project

Goal	Metrics
Goal 1: Improve the predictability of the SDL based SW development in SDL projects	Metric.1 Fault status by SDL module Metric.2 Fault status by time Metric.3 Effort used in each testing phase Metric.4 Effort used in test development Metric.5 Testing time / errors found Metric.6 SDL module complexity / number of faults
Goal 2: Evaluate the Quality of SDL based SW development in SDL projects	Metric.1 Number of faults by SDL module and priority Metric.2 Number of faults by priority Metric.3 Number of faults by SDL module and error classification

Typically, the project managers will use informal assessments of critical factors during SW development to assess whether the end product is likely to meet the requirements [2]. This can be difficult when new design methods and tools are used. In our case, the predictability of the SDL based SW development was measured by the fault, complexity and effort metrics (Table 1). One purpose of the metrics was to help to estimate the remaining testing effort based on the quality, complexity, and effort used. The metrics defined were found to help the design phase, and also to reduce effort later in the coding, testing and maintenance phases. Literature has also indi-

cated that a better design can lead to less complex code and make software easier to maintain [20].

3.2 Implementation of Measurement Automation Environment

Collecting metrics data involves the expenditure of resources such as time, labour and money [20]. Automation saves effort, is more efficient and produces more accurate results, i.e. if the data collected is reliable [13]. The implementation of the environment for measurement automation (Step 2, Fig. 1) was initiated after the selection of the measurement goals and metrics. Some automation activities were already planned as a part of the previous projects. Implementation of the automatic measurement environment was started by an analysis of metrics that could be automated. This analysis was performed based on the location and reliability of the data, and the ease of collecting and calculating the metrics. The environment was implemented by the VTT measurement experts using MetriFlame [19].

3.3 Data Recording, Collection, Calculation, and Presentation

Planning the data collection (Step 2.1, Fig. 1) was started by defining the data sources for the selected metrics. The main data source was the error database. This was due the fact that this database contained most of the data needed. The principal idea of automatic measurement was to facilitate the metric data collection and calculation activities using the MetriFlame. The following data sources were used:

- Fault data was collected automatically from the Error and Inspection databases.
- Status information was collected from the configuration management database.
- Complexity information was calculated using the SDL design tool data.
- Effort information was collected from project management tools.

All collected data was converted to a MetriFlame-specific format using tailored data converters. The converter for the SDL design tool was most challenging to implement. This was due to the necessity of calculating the complexity of the SDL modules, which was not readily available. After the conversion the calculation formulas and the presentation style for the resulting metrics could be created. The data collection from the different data sources is shown in Fig. 2.

Measurement automation does not, however, end with the automatic collection of metric data. It is continued with the calculation and the formation of the resulting analysis graphs (Steps 3-6 in Fig. 1). In our case, the metrics were gathered between May and December 2000 in three case projects. The time between measurements in each project was determined based on the quantity of available data and the time-cycle was tailored to meet the specific needs of each case project.

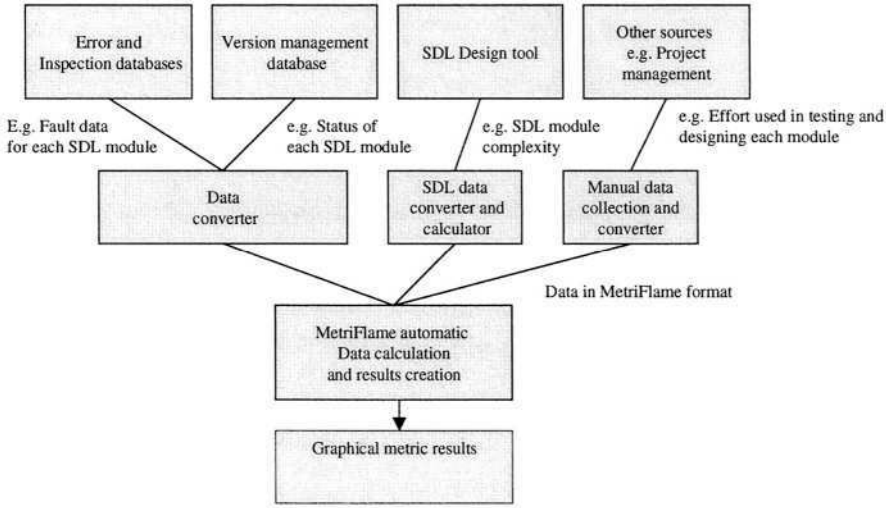


Fig. 2. Data collection from the different data sources

The first metrics collections were conducted and mainly carried out by the VTT measurement experts. However, during the later cycles, also the NMP quality engineers involved in the case projects used MetriFlame to automatically collect and analyse the data.

3.4 Analysis of the Metrics

Metrics analysis in feedback sessions forms a vital part of the measurement process. In the case projects, metrics were analysed every four weeks.

The results were first analysed in a session with the VTT measurement experts and the main designer, the quality engineer and the error manager from the NMP case project. During the first analysis session:

1. All collected metrics were analysed,
2. Problems and improvement ideas that arose from the results were documented,
3. Most valuable metrics were selected for more detailed analysis,
4. Notes and defects in the results were collected and documented.

After the first session, a second analysis session was organised. During the second analysis the main results, improvement ideas and problems found during the first session were presented, discussed and analysed together with the NMP project staff. Project staff had valuable knowledge about the progress of the case projects and could further interpret the main results. The feedback received during these meetings was taken into account when planning for the next measurements. The results were used in project management and SDL quality assurance activities.

3.5 Packaging the Results and Experiences

Packaging the results and experiences is an important part of a metric program since it enables the reuse of metrics and analysis results in future projects. As a result of packaging, a description of the metrics, a guideline for automating measurements, and an experience report were drawn up.

Information about the metrics and their analysis procedures can be reused in organisations [3]. In reality this requires, however, that the metrics and related background and analysis information are carefully documented and packaged during the measurement projects. In our case, the main purpose of the metric set document was to facilitate and speed up the measurement planning procedures. This was done by defining a set of possible measurement goals, metrics and metric analysis presentation styles for the case projects.

The guideline for automatic measurement included detailed descriptions on how to get started and how to implement automatic measurement in the organisation. Experience report included the key findings discovered during the SDL based measurement project. These reports were directed to the project quality engineers and error managers, aiming at facilitating the work of the quality manager and the person responsible for measurement in future projects.

4 Results: Experiences and Lessons Learned

The findings presented in this section are based on the experiences and documentation drawn up. The data was collected from the following sources: Goal definition and metrics selection workshops, measurement results feedback sessions and experience reports drawn up by the SDL measurement automation project. The key findings are shown in Table 2 and discussed later with more detail.

When starting to plan the metrics, project staff is often interested to have many kinds of metrics. Project staff may be busy and purpose of the measurements can easily be forgotten. Therefore, the active participation of managers to keep focus is required.

Metrics selection is one of the most challenging and important phases of measurement projects. Metrics must be useful also to project, not only to management. The metrics must also describe the defined measurement goals and be automatically collectable. According to our experiences, a graphical presentation is a good way to clarify and give idea of future results. In our case, we used the results from the previous measurements and literature. The technical restrictions that affect automatic measurement should be analysed before the selection. Several requirements for the automatically collectable metrics exist, for example, the data sources must support automatic data retrieval. Early analysis of automation possibilities will reduce the amount of work later. Metrics selection requires some time from the project managers, the measurement experts and the quality managers. This is, however, worthwhile, as it guarantees that the project will collect only useful metrics, for which collection, calculation and setting the presentation format can be automated.

Table 2. Key findings for each measurement phase

Planning measurement automation
- A targeted survey facilitates the collection of ideas for measurement goals and metrics and is easy to distribute to the relevant stakeholders. The questionnaire may be reused for each metrics definition cycle. - Prioritisation of metrics (what to collect) can be difficult since the project members may initially want to collect many kinds of metrics. This can be facilitated if: - Constraints for metrics automation are analysed before selecting the metrics. - Graphical examples of the metrics results are created. - A plan for measurements should be included to the project plan and quality plan.
Implementing the environment for measurement automation
- Measurement automation requires a lot of time, results should be repeatable. - The use of measurement tools that support automation (MetriFlame or similar) will help and speed up the implementation. - Changes in the software development environment may cause changes to the measurement automation. Predictable future changes should be taken into account.
Data recording
- Data correctness can be improved by defining correct input formats for data, training project staff to record data correctly and reminding project staff regularly about data correctness.
Data collection
- Useful data sources for SDL-specific measurement data are inspection databases, SDL design charts, error management databases, project management data and configuration management tools (i.e., SDL module versions, etc.). - The use of complementary data sources increases the value of the analysis. - Define clear goals and metrics in order to avoid problems during data conversion and calculation from different tools.
Formulation of analysis graphs and analysis of metrics
- In order to save effort, review results first with a small group of experts. After the first session, arrange feedback session with focused results with project staff. - Make the results available (on-line) for project managers and quality managers at minimum and preferably to all who have participated to measurement process. - Using the results without project staff analysis is very risky because the necessary background information behind the metrics may remain unknown.
Packaging results and experiences
- Guidelines and experience reports can often be directly used in forthcoming projetcs. - The process steps, roles and automation environment may need, however, to be modified to match the needs of the new project.

A good way to plan and report a measurement process is to include it as an integral part of project and quality plans. This combines measurement activities as part of the project's normal working routines and project staff does not regard it as extra work.

In practice, the preparation work for automatic measurement is one of the most time-consuming tasks when starting a measurement project. Metric data sources, calculation and schedules, and storage for the metrics' results should all be defined and implemented before actual measurement collection and analysis can start. We found out that the use of specific tool support for automatic measurement (MetriFlame in our case) makes the measurement data collection, calculation and presentation of the analysis graphs steps straightforward. In MetriFlame, we already had the mechanism for automatically calculating the metrics, creating analysis graphs, and scheduling the tasks. We only had to implement the data collection converters, which convert the data from the different data sources into a format that MetriFlame can read. However, the development of mechanisms for collecting data automatically from different sources requires time and effort from measurement experts because technical constrains of the tools that act as data sources must be understood. Thus, developing the measurement automation is a task, which should be carried out like a development project including the definition of tasks and responsibilities. However, one should remember that the measurement automation environment must only be created once, must be flexible and usable beyond one development project. When this has been done, the project staff can use the automatic tools to collect, calculate and create graphs.

It should be noted that changes in the measurement automation environment are always possible. If the metrics or tools where the data is collected are updated or changed, the changes must also be done in the measurement automation environment. These changes can be laborious or even impossible to implement. Measurement automation is most efficient in a stable working environment, but possible future changes should be predicted where feasible and the measurement tools should be made flexible enough.

Mistakes made during data recording are a very common problem. Erroneous data can easily be inserted into a database and it may lead to false measurement results. This kind of defects can be reduced in two ways:

1. By restricting the formats for recording data during the planning phase
2. By training the project members for entering the data, and by reminding project staff about the data correctness during the development project.

Collected metric data is dependent on the measurement goals and selected metrics. In our case, the data was collected from several sources like the SDL design tool, error and inspection databases and also some data was collected manually. During SDL based measurement, we found out that it is possible to automatically collect measurement data related to SDL based SW development process from various data sources. The SW development tools, processes and predefined metrics that could be used in measuring an SDL based SW development process are shown in Table 3:

Table 3. Examples of the SDL based measurement tools and metrics

Tool	Example metrics	Notes
Inspection database	- Inspected diagrams vs. all diagrams. For each SDL module: - Number of defects - Defect severity and type	Inspection tools contain useful information of inspected SDL and MSC diagrams, including the number and types of defects found. This data gave us a possibility to analyse the quality of the SDL modules in SW development.
SDL design	For each SDL module: - SDL design chart complexity, size, performance	We found design information useful in evaluating SDL module complexity. The complexity metrics were integrated with data describing faults and effort.
Error database	For each SDL module: - Number of faults found during testing - Type and severity of faults found during testing	Error databases were found out to be very useful from the automatic measurement viewpoint. Useful information about the tested SDL modules was obtained, such as the number and types of faults reported.
Project management	For each SDL module: - Planned effort vs. actual effort	Effort information can be collected from a project management tool. One condition for collecting metrics was that effort data must be entered separately for each SDL module.
Configuration management	For each SDL module: - Number of created versions	Configuration management tools contain useful background information for measurement data analysis. The number of versions gave us useful information for the effort analysis.

Combined information from different sources is much more useful than information from single source alone. It also makes integration of results from different sources possible. We also found out that measurement goals and measurement automation possibilities are affected by the tools and the processes, during which the data is collected. In our case, the main goal was to improve the predictability and ensure the quality of SDL based SW development.

Feedback from the project staff has a critical impact on the success of measurement. Due to large amount of metrics and graphs produced automatically, the most important and critical results were selected for the project staff feedback session. First metrics analysis took two hours from the small group of measurement experts and project/quality managers. This preliminary work decreased the time required for the feedback sessions where the project staff was present. the feedback sessions of the most important and critical results took only 1.5 hours from the project members each month. These feedback sessions coupled to project meetings, which also decreased the time used in arranging the meetings.

Up-to-date metric results should be available for project managers and error managers in a pre-defined location. Automatically collected, regularly analysed and updated metric results are a useful way of monitoring project status and defects and predicting the remaining effort and defects. It is, however, important to remember that using the metric results without project staff feedback is very risky. Without feedback from the project, the managers may draw conclusions based on fault results.

Packaging the results and experiences is useful because it summarises the work done on automatic measurement and enables reuse of the results. The most important metrics were used also in later SW development projects at NMP. The guidelines for automatic measurement and the experience report have also been used when starting new SW development projects. However, the measurement process steps and roles were modified to correspond with the needs of the new projects.

5 Conclusions

While metrics programs are challenging to implement, literature has shown that automated software measurement has a significant role when adopting software metrics into systematic use. Many reports show how measurement has been successfully used in improving quality and productivity. Automating measurement is recognised as an important factor especially in a situation where strict time-to-market requirements leave little time for the project staff to introduce new methods and analyse their impact. This paper has discussed the processes and key findings from systematic automation of SDL based measurement in three case projects at Nokia Mobile Phones.

As a result of the automatic measurement project the product projects gained a set of metrics they found very useful. These could be collected automatically from the metrics databases. The result graphs were regularly analysed in the project meetings and used to facilitate project management.

The key findings discussed in this paper were:

- Metrics selection is one of the most important task in measurement automation. However, the prioritisation of metrics (i.e., what to collect) can be difficult since the project members may initially want to collect all possible metrics. Metrics selection can be facilitated by analysing constraints for metrics automation before selecting the metrics and creating graphical examples of the metrics.
- Changes in the software development environment may cause changes to the measurement automation environment, these changes can be laborious to implement. Thus, measurement automation is most efficient in a stable working environment.
- The useful data sources for SDL-specific measurement data are inspection databases, SDL design charts, error management databases, project management data and configuration management tools (i.e., SDL module versions, etc.). Using different data sources increases the reliability of the metrics analysis.

- The metrics and the results of their analysis should be made available on-line at minimum for project managers and quality engineers. It is suggested, however, to make the data available also to individual project members. Using the metrics' results without analysis is very risky because the necessary background information behind the metrics may remain unknown.

It should be reminded that these findings are based on only three measurement case projects. However, the results were obtained over a period of two years, which mitigates the limited number of projects. It should also be noted that these findings are based on analysis carried out during the start-up phase of measurement automation. The processes and findings can be different if measurement automation was more established.

During this measurement automation project the importance of metrics became evident. Planning the metrics and implementing the metrics collection may be time-consuming and hard work, but the results are rewarding. After the measurement automation project described here, several measurement activities have started in Nokia Mobile Phones projects. As one result, MetriFlame was not chosen as an automatic measurement tool at NMP. MetriFlame wasn't compatible enough with other tools and it was therefore used as a reference when improving measurement automation. Some of the final results - the most important metrics, the guidelines and experience reports - have been used in the initiation of new SW development projects. Also, the process steps, roles and automation environments have been tailored for use in many Nokia Mobile Phones projects.

Acknowledgements. The authors would like to thank all people at NMP and VTT who reviewed the paper. Special thanks to Dr. Pekka Abrahamsson for his valuable comments.

References

[1] Basili, V. R. and Rombach D.(1988). "The Tame Project: Towards Improvement-Oriented Software Environments." IEEE Transactions on software engineering 14(6).

[2] Fenton, N., Krause P. and Neil M. (2002). "Software Measurement:Uncertainly and Causal Modelling." IEEE Software 19: 116-122.

[3] Fugetta, A., Lavazza, L. Morasca, S. Cinti, G. Oldano and Orazi E. (1998). "Applying GQM in an industrial software factory." ACM Transactions on Software Engineering and Methodology 7(4): 411-448.

[4] Gopal, A., Krishnan, M. S. Mukhopadhyay T. Goldenson D. R. (2002). "Measurement Programs in Software Development: Determinants of Success." IEEE Transactions on software engineering 28(9): 863-875

[5] Hall, T. and Fenton N. (1997). "Implementing Effective Software Metrics Programs". IEEE Software 14: 55-65.

[6] Herbsleb, J. D. and Grinter R. E. (1998). Conceptual simplicity meets organizational complexity: case study of a corporate metrics program. Proceedings of the 1998 International Conference on Software Engineering.

[7] Humphrey, W. S. (1989). Managing the Software Process. Reading, Mass., Addison-Wesley.

[8] International Engineering Consortium (2002). "Specification and Description Language (SDL).", http:/www.iec.org/online/tutorials/sdl/topic02.html (available 18.9.2002).

[9] Jong, G. (2002). UML-Based Methodology for Real Time and Embedded systems design. Automation and Test in Europe Conference, Paris.

[10] Komi-Sirviö, S.,Parviainen P. and Ronkainen J.(2001). Measurement Automation: methodological background and practical solutions - a multiple case study. Software Metrics conference, London, McGraw-Hill Publishing Company.

[11] Laitinen, M. and Fayad M. E. (1998). "Surviving a process performance crash." Communications of the ACM 41(2): 83-86.

[12] Latum, F., Solingen R. , Oivo, M. Rombach D. Ruhe G. (1998). "Adopting GQM Based Measurement in an industrial environment." IEEE Software 15: 78-86.

[13] Lavazza, L. (2000). "Providing Automated Support for GQM measurement process." IEEE Software 17(3): 56-60.

[14] Niemelä, E., Kuikka, S. Vikuna, K. Lampola, M. Ahonen, J. Forssel, M. Korhonen, R. Seppänen V. and Ventä O. (2000). Teolliset komponenttiohjelmistot -Kehittämistarpeet ja toimenpide- ehdotukset. Helsinki, Tekes.

[15] Pfleeger, S. L. (1993). "Lessons learned in building a corporate metrics program." IEEE Software 10: 67-74.

[16] SDL Forum Society (2002). "What is SDL?", http://www.sdl-forum.org/SDL/index.htm (available 19.8.2002).

[17] Solingen, R. and Berghout E. (2001). Integrating goal-oriented measurement in industrial software engineering: industrial experiences with additions to the Goal/Question/Metrics (GQM) method. Metrics 2001, London, McGraw-Hill publishing company.

[18] Vierimaa M., Kaikkonen T., Oivo M., Moberg M. (1998) Experiences of practical process improvement, Embedded Systems Programming Europe. Vol. 2 (1998) No: 13, 10 - 20.

[19] VTT, Technical Research Centre of Finland (2002). "MetriFlame www-pages." http://www.vtt.fi/ele/research/soh/products/metriflame/index.html (available 18.9.2002)

[20] Zage, W. and Zage D. (1993). "Evaluating Design metrics on large scale software." IEEE Software 10: 75-81.

[21] Zahran, S. (1998). Software process improvement: practical guidelines for business success. Reading, Mass., Addison-Wesley Pub. Co.

Using a Reference Application with Design Patterns to Produce Industrial Software

Marek Vokáč[1] and Oluf Jensen[2]

[1] Simula Research Laboratory
marekv@simula.no
[2] EDB Business Consulting
oluf.jensen@c2i.net

Abstract. System architectures are described in abstract terms, often using Design Patterns. Actual reuse based on such descriptions requires that each development project derive a concrete architecture from the chosen Patterns, and then implement it in code.
This paper describes a case study of an industrial development project that adopted a *reference application* as a starting point, in order to avoid the need to design a complete architecture. Reference applications are usually made by platform or component vendors. Such applications are used to provide executable examples of best practices, for a particular platform, component set, or other technology. In this case study, the Pet Store application for the J2EE platform was chosen. Pet Store is documented by Design Patterns and is also available as source code. The development consisted of replacing the application logic, while keeping the architecture intact. The project was thus transformed from an *ab initio* development project into a major reuse/modification project. This development project was part of a software process improvement effort looking at processes *for* and *with* reuse.
Our results indicate that this approach works well, provided that the functional and non-functional requirements of the project match those of the reference application. The development time was shortened by approximately one third relative to the original estimate, and a well-designed application was produced despite lack of experience with the platform and n-layer architectures. Surprisingly, the production deployment to a different application server was still problematic, even though the original reference application was available as a sample for that server.

1 Introduction

Software Design Patterns [8, 12, 29] document solutions to well-known and defined problems, and thus provide an opportunity for architectural reuse. However, the structures and behaviour specified by a Pattern are at an abstract level and have to be translated into a concrete architecture, and then implemented in code, in each development project.

There exist other forms of architectural guidelines, such as reference architectures. These vary from very high-level and methodology-oriented approaches that describe organisational processes and general criteria for choosing between technologies, such

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 333–347, 2004.

as [6], to the more concrete, technological recommendations described in [13], where possible components of a mobile application are described using prose and UML diagrams. Still, the burden of correctly designing and implementing the concrete architecture lies with the developers of each particular project.

By contrast, small pieces of sample code tend to illustrate a low-level solution to an isolated problem, and are more useful as examples than for direct reuse. While they often assume a particular architectural model, they do not explicitly specify or provide it.

This paper investigates a different approach: development by reuse and adaptation of a publicly available *reference application.* The starting point is a complete, working application that implements acknowledged best practices, with a concrete architecture documented by reference to well-known Design Patterns.

In our case study, the development was based on the *Pet Store* reference application [32] available for the J2EE platform. The development was performed by replacing the user interface, functionality and database structure, while leaving the architecture and application structure intact.

The decision to attempt large-scale reuse was partly motivated by the wish to quickly create a high quality Web application, in spite of a lack of experience with the J2EE platform and multi-layered architectures. It was also motivated by an ongoing software process improvement project, in which the company cooperated with several research institutions. This project defined a refinement process investigating a development process *for* and *with* reuse.

This paper contains the following sections: Section 2 reviews reuse concepts and locates the present contribution in context. Section 3 presents the research methods, and Section 4 the development project that was studied. Results are described in Section 5, and Section 6 concludes and describes possible future work.

2 Background and Concepts

A major challenge in developing a modern multi-layer, web-oriented application is to design an architecture that supports the required functionality on the specified platform, and to translate that architecture into running code. In this section, we look at some popular forms of software reuse, and define the salient concepts used in the present paper. These concepts form the basis for the present case study and the project that was investigated.

2.1 Forms of Reuse

The tradition of software reuse dates back to 1968 [21] and earlier. Over time, several distinct modes of reuse have been described, as by Thomas, Delis *et al.* [33]. Their classification divided reuse into three kinds: *verbatim* reuse, where a component is not modified; reuse with *slight* modification; and reuse with *extensive* modification. In addition, we must take into account the kinds of artefact that are being reused.

At the architectural level, the artefacts available for reuse are abstract. Design Patterns [31], books on Best Practices [3, 17, 22], UML models [13] and more general

reference architectures [9] must all be translated into concrete architectures suitable to the project at hand.

At a more concrete level, we find libraries and *frameworks* such as STRUTS [4]. Frameworks are used by plugging new components into them, to leverage existing functions, and to a certain extent, they dictate or encourage a particular architectural direction.

A concept that has recently received some attention is the *product family,* as described in [19, 20]. When individual applications cover overlapping sets of requirements, there is a potential for a product family with shared modules. This often results in frameworks, libraries or other constructs meant to foster reuse between the applications.

In his seminal paper, Brooks [7] argued that the *essence* of software lies in its complexity, conformity, changeability and invisibility; most of the improvements in the field of programming have addressed *accidental* factors, such as high-level languages and system response times (and thereby development cycle speeds). In the same way, plugging components into a framework or reusing sample code snippets may not solve the difficult problem of designing an architecture that conforms to a complex combination of functional, non-functional and platform requirements, while providing reasonable flexibility for future development. The task becomes especially difficult when time constraints tempt the project developers to formulate simple solutions to immediate problems.

2.2 Reference Applications

Our work is based on the concept of a *reference application.* Such applications are made available by platform [23, 24, 32] or component [5, 15, 27] vendors for public use. They share a number of defining features: they are complete, running applications; full source code can be downloaded and dissected; they implement current best practices in the area they are meant to illustrate; and they are very well documented.

We are particularly interested in the fact that a reference application is executable, and provides full source code in addition to architectural descriptions. In doing so, it bridges the gap between abstract prescriptions for "good" architecture, and the mass of details that must be considered in any final implementation. This is one of the more difficult parts of the design process, and a reference application provides a fully implemented answer–within the requirements set by its authors.

By basing the development on Sun's Pet Store reference application, the project became somewhat similar to a product family project. However, the "original" application was not an in-house product but instead a publicly available reference application. The development work belongs to the "reuse with extensive modification" kind at the *code level,* but "reuse with slight modification" at the application and architectural levels.

2.3 Patterns versus Code

The relationship between the abstract and fairly simple description provided by a Design Pattern and the complex structure that may arise from its implementation can be illustrated by looking at two central patterns in the Pet Store reference application.

One of the most central Patterns is the Model-View-Controller [31], which seemingly consists of three components. However, as implemented in Pet Store, the Model part consists of Enterprise JavaBean objects that access the database and process data, the role of Controller is performed by a set of objects that handle incoming requests, and the View is generated from screen templates combined with dynamic data.

In this way, what is conceptually a fairly simple pattern gives rise to a large number of objects (4+ for Model, 7 for View, 7 for Controller) for a single Use Case. The architecture provides considerable flexibility for screen and interaction design, and separates the business and data access logic from the presentation. However, correctly deriving and implementing such a design *ab initio* is not trivial.

At the opposite end from the presentation layer, we find the database. This component is often specified by the customer, because choosing the company DBMS tends to be a strategic, high-level decision.

There are several versions of the SQL standard, and each vendor has specific additions, extensions and limitations. It is therefore often necessary to make changes to applications when switching from one database to another. For instance, the syntax and rules for OUTER JOIN varies, and choosing the correct cursor type may have critical impact on performance. If SQL statements are spread throughout the code, this becomes a more difficult task.

The Data Access Object pattern specifies that one should make an object that encapsulates access to a particular resource, while hiding the actual implementation of the storage mechanism. This is a way of partially achieving "persistence transparency" as defined in [16]. In the general case, this can be a large task, but implementing access objects for particular classes is more tedious than difficult.

3 Research Methods

Our research design is a single case study with one study unit – the development project [34]. The study was initiated after the development project was mostly completed, and was therefore conducted retrospectively. The research was performed as part of the Software Process Improvement through Knowledge and Experience (SPIKE) project, a research/industry collaboration partly funded by the Norwegian research Council.

Data was collected from project documents and logs, including the source code and UML designs. A full-day post-mortem seminar where all the developers participated was conducted. In addition, individual developers and representatives of the customer were contacted as needed to collect and verify information.

One of the authors (O. J.) served as development project manager and contributed first-hand experience.

4 The Studied Development Project

The Norwegian public sector actively attempts to provide 24-hour service for its citizens by using the Internet. The development of publicly accessible portals makes it possible for anyone to obtain certain services, which would otherwise only be avail-

able during normal working hours, at any time. Examples include Inland Revenue services (including tax returns), work and employment services, Social Benefit services and Municipal services.

The purpose of this project was to develop one such service, which makes it possible for the public to apply for a driver's license over the Internet, and which implements a secure and automated application process:

- The information received from the applicant is checked for correctness
- The application is checked against the rules and regulations governing driver's license applications.
- Once the application for a driver's license is accepted, it becomes available to a public servant employed by the Road Authority, who is then responsible for processing the application.
- The applicant is automatically given a coded reference number by e-mail. This makes it possible for her to check the status of her application at any time.
- Once the application has been processed and accepted, the applicant may then use her reference number to select a date for her driver's license test.

The inception phase of the project produced a Requirements report containing specifications on several levels: Background and Purpose of the application; Terminology: methods and tools; a Business Process Model description; a Use Case Model description; a High level Business Class skeleton; sketches of User Dialog and Forms (HTML), and Database Models. This report also ensured that the customer's basic requirements were well documented. A short description of two central Use Cases follows:

USE CASE Registration of license application
The following information is registered:

- Personal information: First name, Last name, social security number, home address and e-mail address
- Applying for: First time applicant, additional vehicle classes
- What vehicle class
- Health information

The submitted information is checked against public registers and applicable regulations before it is accepted and stored as a driver's license application.

USE CASE Application status
Any applicant can look up her own application to view its status. The applicant must use the reference number she received via e-mail when the application was accepted.

4.1 The Pet Store Reference Application

The Pet Store reference application is presented in the book "Designing Enterprise Applications with the J2EE™ Platform" [30]. It conforms to our general definition of a reference application:

> "Its goal has been to introduce enterprise developers to the concepts and technology used in designing applications for the J2EE platform, and to give practical examples of a typical enterprise application."

Furthermore, Pet Store is an example of a particular kind of J2EE application:

> "The Pet Store application is a typical e-commerce site. The customer selects items from a catalogue, places them in a shopping cart, and, when ready, purchases the shopping cart contents. Prior to a purchase, the sample application displays the order: the selected items, quantity and price for each item, and the total cost. The customer can revise or update the order. To complete the purchase, the customer provides a shipping address and a credit card number."

Pet Store implements a number of hierarchically ordered Design Patterns. For instance, the MVC architectural pattern is implemented using Front Controller and Composite View, and database access in the Model part is performed by objects conforming to the Data Access Object pattern. This illustrates which tiers the application uses, and how to distribute the functionality across the tiers.

Sun's documentation [32] consists of a high-level architectural overview, followed by a description of the relevant patterns, with references to actual classes and objects in the Pet Store application.

This provides both a pattern-based description and a working implementation with all the details filled in. The application is executable (provided one sets up a database server), and can be traced/debugged to find the actual order of execution.

The Pet Store reference application has been the subject of some debate in various Web forums, especially following its use in a benchmark to compare the J2EE and .NET platforms [2, 10, 35]. It is also referred to in articles such as [28], on analysing the usage of exceptions in large Java systems, and it is used in university curricula, e.g. [25]. It is therefore relatively well known among Java/J2EE practitioners.

4.2 Development Methods

In addition to simply developing a new application for a customer, the company also wished to improve its software development process. To achieve this, the software process improvement project was established in collaboration with research institutions. One of the goals was to find and establish a refinement process, processes and methods *for* and *with* reuse.

The implementation of processes *for* reuse implies that any future development project would have access to a "toolbox" of reusable elements, such as model elements, components or design patterns. The implementation of processes *with* reuse implies that development projects have knowledge of, and make use of this toolbox of reusable elements. In an example of successful reuse at Matra Cap Systemes [14], the process was jump-started by focusing on projects that could produce short-term gains while at the same time laying the foundation for more long-term reuse. The same approach was adopted here, by using the Driver's License project as the starting point for an improvement process. A crucial process was that of choosing a reuse strategy for the development project.

Key factors in the evaluation of reuse strategies were that the developers had relatively little experience with the J2EE platform and multi-level architectures. They therefore wished to find a form of reuse that did not require the design and full implementation of a complex architecture from abstract descriptions.

Given these considerations, and with reference to the various forms of reuse outlined in Section 2, the project chose to look closely at Sun's Pet Store reference application. This revealed that Pet Store's functionality had a sufficient level of similarity to the new application, as detailed in the next two sections.

4.3 Functional Requirements Matching

The refinement process suggests looking for matching functional requirements. To the development team, the functional similarities between the two applications were clear, as illustrated in the following table:

Table 1. Functional mapping between the Pet Store and Driver's License applications

Driver's License	Pet Store
The applicant selects the service and type of license	The customer shops by selecting items from a catalogue
The applicant applies for a license	The customer places an order
The applicant fills in personal details	The customer fills in personal details
Personal details are verified against public registers; rules and regulations are checked against a rule database	Credit card data are verified externally; order consistency and completeness are verified by ERP or other systems
The applicant receives a confirmation e-mail with a case ID	The customer receives a confirmation e-mail with an order ID
Other (legacy) systems may process the same information	An administration manager reviews stock and enterprise financial data

The database structure used in Pet Store did not match the legacy database that the Driver's License application was to use. It was expected that the Data Access Object pattern used in Pet Store would shield the rest of the application from the necessary modifications.

4.4 Non-functional Requirements Matching

Simply matching functional requirements is not sufficient to ensure successful reuse; non-functional requirements also have to be considered. Figure 1 shows a logical architecture that fits both the Driver's license and Pet Store applications:

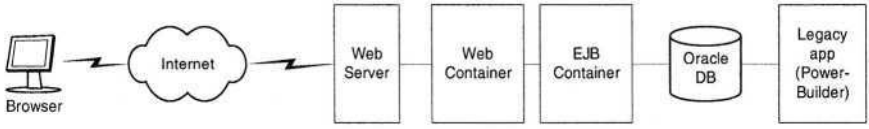


Fig. 1. Logical architecture of the Driver's License application

The product had to satisfy a number of environmental and other non-functional constraints. Pet Store matched these requirements quite well, with some exceptions:

Table 2. Non-functional mapping between the Pet Store and Driver's License applications

Driver's License	Pet Store
Platform: J2EE v. 1.2	J2EE v. 1.2
3- or n-layer architecture	3-layer architecture
Browser: IE5 and equivalent	Web pages are generated from tagged templates, adaptable to different browsers
Data storage in legacy database	Data is stored through Data Access Objects, adaptable to requirements; *possible mismatch*
Deployment platform uncertain	Pet Store is a reference/sample application included by all major platform vendors
Project size: ≤ 10 forms, ≤ 20 database tables	Pet Store is a relatively simple e-commerce application of comparable size
Rigorous validation of personal details needed	Incoming data (orders, addresses) assumed to be valid; *mismatch*

4.5 Application Server Compatibility

The Pet Store application has been used as a benchmark for testing application server conformance. For instance, IBM includes it on the IBM WebSphere Application Server V4.0 for Windows NT product CD. Similarly, Oracle provides a Pet Store implementation for their Oracle9iAS Containers for J2EE [26].

The Pet Store application is billed as a reference for both architecture and implementation by Sun, the vendor most involved in defining and implementing the J2EE platform. It seemed reasonable to have a high degree of confidence in the compatibility of the application with all the major server platforms.

It was therefore expected that basing the Driver's License application on Pet Store would make it simple to port it to different application servers. This was an important requirement, since the customer for the system had not made the final choice of application server at the time the project began. A server change was consequently almost unavoidable.

4.6 Other Factors

Basing development on Pet Store was expected to give the development team a number of advantages not available with other forms of reuse:

Development and implementation would be carried out by replacing all modules within each tier with project-specific code. The Pet Store architecture would be retained while replacing the actual functionality, avoiding the need for an architectural design, prototyping, etc.

Because the application would be executable from day one, regressions were expected to be easy to identify. The developers, project management and the customer

would be able to see a running version that implemented the full production architecture at frequent intervals.

Finally, the total time used in development was expected to be less than for a normal development process, since development effort could focus directly on implementation, bypassing architecture and design. The project management expected to be able to show a working subset of the application to the customer after only two iterations, at least one iteration less than would otherwise have been the case. An iteration normally lasted about 3 weeks.

5 Results

Our case study found results that can be related to four different areas of the development project: organization; implementation; data structure and security; and deployment.

5.1 Project Organization

At the outset, the project team decided on a software development process that resembled the Unified Process approach [18]. The U.P. promotes iterative development, and describes iterations within the following phases: Inception, Elaboration, Construction, and Transition.

With a working application as a basis, iterations could be run in parallel in two phases that are usually performed in sequence: the Elaboration phase and the Construction phase. The Elaboration phase could focus more on identifying high-risk functionality and non-functional requirements, while the Construction phase focused on the implementation of a stable executable subset of the final product. At the same time, a Design and Implementation model was produced through reverse engineering at regular intervals. Otherwise, design and architectural activities were largely considered *unnecessary,* since the development effort was "reuse with only slight modifications" at the architectural level.

On average, six developers were involved. The development team was initially organized according to the defined tiers: two people were responsible for the client tier, two were working on the EJB tier, one person was working on the WEB tier, and one person was responsible for the database tier. Earlier experience suggests that this approach has a number of benefits [11].

However, as development work progressed, it turned out that it would be better to reorganize work according to what *design patterns* a team member was working on. In other words, when a person was working on the Front controller pattern (which starts in the WEB tier), then he would also be responsible for the controller-related objects for that function in the EJB tier.

This provided team members with cleaner interfaces between each other's functionality and design patterns, and clear functional responsibilities. The project management also had a better control of responsibility for separate functionality. A consequence was better management and implementation of change orders from the customer.

However, there were exceptions to this rule. The people working on the client tier were, in the early stages, very occupied with the design of the application forms, and parameterization to the WEB tier.

5.2 Implementation

The implementation discipline had to produce a working subset of the final product at regular intervals, maintain the implementation model, and solve data validation and security issues. At the outset, the project had a working Pet Store enterprise application. This was now to be transformed into a Driver's License application enterprise system.

The prime use case was defined as Registration of license application (Section 4). The first functional subset to be produced had to receive and process a driver's licence application, through all tiers, and store it successfully in the database. This implied connecting all the different tiers so that application form data could be properly processed and stored in the database.

Connecting a new set of client tier web pages to the WEB tier proved simple. Once the web pages were designed, it was largely a parameterization task to secure a connection to the various handlers on the WEB tier. This is largely due to the screen definitions held in xml files, inherent in the Pet Store design.

Similarly, the connection between the WEB tier and the EJB tier proved to be simple, as the event handling mechanism is largely independent of the events it handles.

5.3 Database Structure and Data Security

The data validation and security issues clearly belonged to the database tier. Solving these issues in this tier caused no conflicts with the functional responsibilities of the rest of the development team.

The structure of the legacy database was completely different from that of the Pet Store database. This required changes in the bean-managed persistent entity beans, and their Data Access Object pattern. The interaction between the Controller administrating calls to the database, and the entity beans performing the calls, also had to be reworked.

The validation and security issue was caused by a fundamental mismatch between an assumption underlying Pet Store and the Driver's License application. Since the Pet Store is a shop, it assumes that its customers will provide valid data and simply accepts the data proffered. However, in our case there are extensive rules and regulations to take into account. Also, since the end result–a driver's license–is a valuable legal document, there is a danger of malicious misuse and fraud. It is therefore necessary to validate and protect the data in what was previously a closed, in-house system at the Road Authority.

The security issue was resolved by storing driver's license application information in temporary tables until the system approved the application. Triggers and stored procedures ensured that accepted and approved application data was moved to the proper tables. This process was not part of the Java application logic at all. Application data that was not accepted was discarded. Both the database structure and security issues required several iterations before they were satisfactorily closed.

In the first iteration, no such temporary tables were implemented. In addition, only page one of the application form was stored in the database. Then, step by step during the following iterations, data from all application form pages were stored securely in the database. This had no negative impact on the progress of the rest of the project.

5.4 Deployment

Deployment on the production platform was the focus of the transition phase. During the preceding elaboration and construction phases, all implementation and deployment were performed using the Oracle OC4J application server. This application server is very easy to use, enabling the development team to produce subsets of the final product at short intervals.

Six months into the construction phase, the customer decided to implement its new public portal on the IBM Websphere 4.01 application server. The transition to this new application server proved to be difficult. Both the original Pet Store application and the Driver's License application failed to deploy to this server.

This came as a surprise to the development team. If the Pet Store enterprise application acts as a reference application, then it should be able to run on all application servers that are certified to run J2EE. Conclusions derived from researching the issue can be summarized as follows:

- Different application servers have different deployment descriptor files, and these are organized differently.
- OC4J had a simpler and less restrictive RMI check, while Websphere required a strong RMI check.
- The classloaders of Websphere caused problems, and may be related to the strong RMI check.
- The Websphere single server edition behaved differently than the advanced edition.
- Websphere generally proved very difficult to work with [1]. The compile, deploy, and run cycle was slow and required a lot of machine resources. Debugging was difficult, and often gave little valuable information.

6 Conclusions and Future Work

Our case study investigated a commercial project to develop a Web-based public service, for applying for driver's licenses. Having completed the requirements documentation, the project had a choice between several possible strategies for the actual development. One possibility was to design and implement the application from the ground up, using available tools in the form of Design Patterns, documented Best Practices and sample code.

Alternatively, it could be designed around a commercial or public domain framework or library. Ultimately, the team chose a different approach, by basing development on a reference application.

The developers had relatively little experience with the J2EE platform and multi-level architectures. They therefore wished to find a form of reuse that did not require the design and full implementation of a complex architecture from abstract descriptions, such as Design Patterns or architecture guidelines. The Pet Store reference application had a sufficient level of functional and non-functional similarity to the new application that it was chosen as the basis for the new application.

The development team, together with central members from the participating research institutions, performed a post mortem analysis on the project experience. The following sections summarize the results.

6.1 Positive Experiences

Through Pet Store, the project possessed a basic architecture that employed the current best practices on J2EE design patterns and technology. Since the application was already implemented, little effort needed to be focused on architectural issues, and most effort was expended on implementation activities.

The learning curve for the team members was very good. This was definitely a problem-based learning approach, while at the same time the textbook answer was readily at hand.

The development tasks were considered professionally challenging, both with regard to functions of the final product, and to the implementation in a new and immature technology. The team motivation was therefore high, even though this was no longer a "pure" development project.

Collaboration within the group was very good. Because of the chosen approach, there was always an executable application, and it was therefore easier to coordinate the efforts of the team members. The problem of people working for an extended period on an isolated problem and thereby drifting away from the group was avoided.

The total time and effort spent on the development matched expectations quite well. There is no doubt that this approach worked well for the application, and resulted in a combination of flexible architecture and short time to market that would otherwise have been hard to achieve.

6.2 Negative Experiences

Deployment on a production application server was expected to be easy, since Pet Store–the reference application used as a basis–is supplied as a sample application by the platform vendors. However, this turned out not to be the case, and even with assistance from the vendor, it was difficult to deploy both Pet Store and the new application on the production server.

The database design in Pet Store is stand-alone, and the application architecture does *not* contain provisions for interfaces and adaptation to existing database schemas (as opposed to purely technical adaptation to DBMS servers, which *is* covered by the Data Access Object pattern used). The database structure issue was solved by rewriting most of the data access objects, so here the benefit of using Pet Store as a basis was small.

The mismatch between the assumption of user trustworthiness between Pet Store and the Driver's license application is an example of a non-functional requirement

that has architectural implications. When evaluating a development approach, it is important to check both functional and non-functional requirements closely. Special attention should be paid to non-functional requirements, even if it can be more difficult than checking functional requirements.

6.3 Conclusions

Basing new development on the code of an existing, well-documented application can be a viable method of reuse. Provided that functional and non-functional requirements match, the actual domain of the reference application and the new application need not be the same. Database structure and security are two areas where particular care must be taken, and where the potential for reuse may be limited. The Pet Store application by Sun is sufficiently well structured and documented to be usable in this role.

6.4 Future Work

In most e-commerce applications, there is a front end that presents what products are available, and accepts user information. Other tiers are responsible for processing and storing this information, and producing a response that is then presented to the user by the front end. Data is stored in a database and passed on to other systems for further processing.

In the Pet Store reference application, this functionality is described at two different levels: at an analytical and design level, through a set of design patterns, and at an implementation level through the implementation of Pet Store.

The software improvement project defined a refinement process that would produce a toolbox containing an increasing number of refined model elements, frameworks, components and design patterns.

The elaboration phase of the development processes should also contain a refinement discipline, a discipline *with* reuse. The heart of this discipline is a search for similarities at many levels: functional and non-functional requirements, architecture and code.

Conversely, refinement *for* reuse should aim at producing artefacts at many levels. One way would be to attempt to refine a Pet Store type application into a set of design patterns (a pattern language for e-commerce applications?), and their equivalent components.

Then, when a similar e-commerce application is to be developed, the development process would mostly be a process of connecting a set of well-defined design patterns and their components into a working application.

Some of the preconditions needed to succeed using the reference-application approach were stated in the sections on functional and non-functional requirement matching (4.3–4.6). Further work is needed to determine a more general specification of the necessary and sufficient conditions.

Acknowledgments. The authors are most grateful to Prof. Dag Sjøberg, Prof. Ray Welland and Andrew McDonald for their extensive input and comments. This project was in part funded by the Norwegian Research Council as part of the SPIKE project.

We would also like to mention the entire project development team at EDB Business Consulting, without whom this project would not have been a success.

References

1. Aeinehchi, N., *Do NOT use WebSphere unless you are BLUE.* 2002. http://www.theserverside.com/reviews/thread.jsp?thread_id=13639
2. Almaer, D., *Making a Real World PetStore, TSS Newsletter #31.* 2002, The Server Side. http://www.theserverside.com/resources/article.jsp?l=PetStore
3. Alur, D., J. Crupi, and D. Malks, *Core J2EE Patterns.* 2001: Sun Microsystems, Inc. 460.
4. Apache Jakarta Project, *STRUTS home page.* 2003. http://jakarta.apache.org/struts/
5. ATG, *ATG Business Commerce Reference Application Guide.* 2003, ATG. http://www.atg.com/repositories/ContentCatalogRepository_en/manuals/ATG6.0.0/pdf/ATG6BusCommRefGuide.pdf
6. Bernus, P. and L. Nemes, *A framework to define a generic enterprise reference architecture and methodology.* Computer Integrated Manufacturing Systems, 1996. 9(3): p. 179-191.
7. Brooks, F.P.J., *No silver bullet: Essence and accidents of Software Engineering.* IEEE Computer, 1987.
8. Buschmann, F., et al., *Pattern-Oriented Software Architecture.* 1996, Chichester: Wiley.
9. Ciancarini, P., et al., *Coordinating multiagent applications on the WWW: A reference architecture.* Ieee Transactions on Software Engineering, 1998. 24(5): p. 362-375.
10. Ditzel, C., *Charles's Corner: Java Technology Pointers.* 2003, Sun. http://java.sun.com/jugs/pointers.html
11. Frederick, C. *Extreme Programming: Growing a Team Horizontally.* in *XP/Agile Universe 2003.* 2003: Springer-Verlag Heidelberg.
12. Gamma, E., et al., *Design Patterns: Elements of reusable object-oriented software.* 1995: Addison-Wesley, Reading, MA, 1995.
13. Hallsteinsen, S. and E. Swane, *Handling the diversity of networked devices by means of a product family approach,* in *Software Product-Family Engineering.* 2002. p. 264-281.
14. Henry, E. and B. Faller, *Large-scale industrial reuse to reduce cost and cycle time.* Software, IEEE, 1995. 12(5): p. 47-53.
15. Infragistics, *Expense Application - reference application.* 2003, Infragistics. http://www.infragistics.com/products/thinreference.asp
16. ISO, *ISO/IEC 10746: Information technology -- Open Distributed Processing -- Reference model.* 1998. http://www.iso.org/iso/en/CombinedQueryResult.CombinedQueryResult?queryString=10746
17. Kassem, N., *Designing Enterprise Applications with the J2EE Platform.* 2000: Addison-Wesley Pub Co. 368.
18. Larman, C., *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process.* 2nd ed. 2001: Prentice Hall. 656.
19. van der Linden, F., *Software product families in Europe: the Esaps & Cafe projects.* IEEE Software, 2002. 19(4): p. 41-49.
20. van der Linden, F. and J.K. Muller. *Composing product families from reusable components.* in *Systems Engineering of Computer Based Systems, 1995., Proceedings of the 1995 International Symposium and Workshop on.* 1995.
21. McIlroy, D. *Mass produced software components.* in *NATO Conference.* 1968. Garmisch, Germany.
22. Microsoft, *Application Architecture for .NET: Designing Applications & Services.* 2003: Microsoft Press. 200.

23. Microsoft, *Duwamish 7.0.* 2003, Microsoft. http://msdn.microsoft.com/netframework/downloads/samples/?pull=/library/en-us/dnbda/html/bdasampduwam7.asp
24. Microsoft, *Microsoft .NET Pet Shop 2.0.* 2003, Microsoft. http://msdn.microsoft.com/netframework/downloads/samples/?pull=/library/en-us/dnbda/html/bdasamppet.asp
25. Ngu, A., *CS5369A Enterprise Application Integration.* 2003, Department of Computer Science, Southwest Texas State University. http://www.cs.swt.edu/~hn12/teaching/cs5369/2003Spring/admin/intro.html
26. Oracle, *Oracle9iAS Containers for J2EE User's Guide Release 2 (9.0.2).* 2003, Oracle Inc. http://otn.oracle.com/tech/java/oc4j/doc_library/902/A95880_01/html/toc.htm
27. Rational Inc, *PearlCircle Online Auction for J2EE.* 2003, Rational Inc. http://www.rational.com/rda/wn_2002.jsp?SMSESSION=NO#pearlcircle
28. Reimer, D. and H. Srinivasan. *Analyzing Exception Usage in Large Java Applications.* in *ECOOP workshop on Exception Handling.* 2003. Dramstadt.
29. Rising, L., *The Patterns Handbook.* 1998: Cambridge University Press.
30. Singh, I., et al., *Designing Enterprise Applications with the J2EE Platform.* 2002: Addison-Wesley. 352.
31. Sun, *J2EE Patterns Catalog.* 2002, Sun Microsystems, Inc. http://java.sun.com/blueprints/pattern/j2ee_patterns/index.html
32. Sun, *Java Pet Store Demo 1.1.2.* 2003, Sun. http://java.sun.com/blueprints/code/jps11/docs/index.html
33. Thomas, W.T., A. Delis, and V.R. Basili, *An analysis of errors in a reuse-oriented development environment.* 1995, University of Maryland, Institute of Advanced Computer Studies.
34. Yin, R.K., *Case Study Research, Design and Methods.* 3rd ed. 2003, Thousand Oaks, CA.: Sage Publications.
35. Öberg, R., *Review of "The Petstore Revisited: J2EE vs .NET Application Server Performance Benchmark".* 2003, Rickard Öberg. http://www.google.com/search?q=cache:8OPCFEFDFd0J:www.dreambean.com/petstore.html+petstore+java+experience&hl=en&ie=UTF-8

Using RUP for Process-Oriented Organisations*

João M. Fernandes[1] and Francisco J. Duarte[2]

[1] Dep. Informática, Universidade do Minho, Braga, Portugal
[2] Blaupunkt Auto-Rádio Portugal, Braga, Portugal

Abstract. In this paper, based on a proposal of a reference framework for process-oriented software development organisations, we discuss some issues related to the usage of the RUP's disciplines as instantiations of those processes. The proposals made in this paper were consolidated based on experiences from real projects. The respective lessons learnt from two of those projects are reported.

1 Introduction

Organisations are moving from a hierarchical structure, sub-divided by departments, to a model where multi-disciplinary teams run horizontal business processes that cross all the organisation. Thus, to develop software applications for this type of organisations two topics must be taken in consideration: (1) the software development process must be appropriate and controlled; and (2) the organisational platform where the processes run must be modeled and considered. The structure of the software development organisation has a major impact on the final quality of the software system being created.

This paper uses a reference framework for process-oriented software development organisations, presented in [1], that instantiates processes as RUP's disciplines. This framework was used in 2 projects and we show the lessons learned from them.

The structure of the paper is the following. In section 2 a generic reference framework for process-oriented organisations that focus its activities towards the needs of the clients is introduced. In section 3, based on that generic framework and in the RUP's disciplines, we detail it in order to describe the specific case of organisations that develop software. The Business Process Modeling is explained in section 4. Finally, on section 5, 2 case studies are presented, including the results analysis and lessons learned.

2 Generic Framework for a Process-Oriented Organisation

Business processes can be seen as a set of tasks executed to achieve a given business goal. Within the business process, human resources, raw material, and internal procedures are combined towards a common objective. An enterprise can be viewed as a group of competence centres, which share the same goal, whenever a process is instantiated [2].

In this type of organisations, the members are no longer dependent on an hierarchical chain of command. Each member must have expertise in one area where he performs

* Work partially supported by Fundação para a Ciência e a Tecnologia (FCT) and Fundo Europeu de Desenvolvimento Regional (FEDER) under project "METHODES: Methodologies and Tools for Developing Complex Real-Time Embedded Systems" (POSI/37334/CHS/2001).

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 348–362, 2004.

his functions. For that, he is helped by a coach to improve his capacities/skills and by a process owner to co-ordinate his activities with the other team members.

Generically, an organisation exists to supply a set of products or services to its clients. To achieve this, the organisations need to execute a set of internal activities. Actually, organisations do not exist isolated, but are part of markets, where other organisations can be their competitors, suppliers or clients. The creation of wealthy, within private organisations, or the supplying of products or services with social impact, within non-profit organisations, follow the same underlying principle: the fulfillment of the clients' needs. Thus, these needs and the expectations of the clients are vital for defining the internal structure of an organisation.

A process inside an organisation can be viewed as a set of activities that has as inputs a set of services and/or materials. It also has as outputs a set of services and materials. This view must be oriented towards the necessities of the clients and the creation of added-value. This implies that the clients' requirements must always be considered, both during the development and the performance of the system.

In an organisation, there are other processes rather than those that provide added-value to the clients. Within an organisation, the management by processes requires a structure that differs from the typical functional hierarchy. It is mandatory to synchronise the processes among them and to fulfil the strategic objectives of the organisation. Thus, for a process-oriented organisation, a structure with the following components should exist [1]:

Process Management Top Team: This team includes the top managers and all processes owners.

Process Godfather: For each process one godfather, which must be a top manager, should be designated.

Process Owner: For each process, an owner is needed. He must have know-how on managing processes and persons, but also competency in the areas associated with the process.

Multi-Disciplinary Team: These teams must be created for each added-value process. Considering the organisation size and its strategic objectives, multi-disciplinary teams may also be established for management and support processes. This decision must be taken by the process owner and godfather.

Execution Teams and Team Leaders: These teams and their leaders represent the instances of a given process. Therefore, during the execution of a process, some teams will use it with a specific focus. For example, for a given production process, one team may be responsible for producing parts for industrial clients, while other team may produce them for individual clients. This partitioning must be managed by the process' multi-disciplinary team, but the team leader is supposed to suggest its composition. Each leader must obey the definitions and use the indicators of the process which he is responsible for.

To align a process-based organisation with its strategic objectives, it is crucial that the goals are based on the organisation's mission and vision, and also on its principles and values. Based on those strategic objectives and in the business plan, the priority when deciding the key processes within the organisation can be perceived.

After completing these stages, highly connected to the top management, we propose the following pragmatic procedure to introduce the processes in an organisation: (1) Define the processes: identify the collection of processes in an organisation and decide on the relative importance of each one; (2) Define the sub-processes and tasks: each process is decomposed into sub-processes, until we reach a level of detail where only primitive activities remain; (3) Identify the interfaces among processes; (4) Assign the owner and godfathers to the processes and the members to the multidisciplinary teams; (5) Define the goals of the processes; (6) Define the key indicators; (7) Define the processes' team leaders and executing teams; (8) Measure and monitor the processes; (9) Execute corrective actions; (10) Review and continuously improve the processes.

The identification of the key processes within an organisation must be accomplished with the help of criteria. Some of those criteria are: (1) Identify if the process traverses several functions; if this is not the case, we might have an activity or a task; (2) Verify if the process is measurable and is integrated in the business plan; otherwise, the process must be terminated, since it is not aligned with the organisation's strategy; (3) Verify if there is a focus towards the clients and their requirements; when this does not happen, the process may be inefficient; (4) Determine the importance of the process for the final quality of the product; (5) Determine the importance of the process for the fulfilment of the organisation's mission; (6) Verify if the process is multi-disciplinary; if that is not the case, we may have an activity or a task; (7) Find out the importance of the process to the success of the organisation.

Based on these criteria, we can select which processes of a given organisation will have a process-oriented management and the activities and tasks they will be composed of. Some processes, activities, and tasks will be eliminated since they do not add any value to the clients, neither to the organisation. These eliminated (or redefined) processes, activities, and tasks and their respective consequences in terms of reorganisation and impact in human resources are the essence of re-engineering [2].

Besides this general framework, inside each process category (management, support, and added-value), there are several processes that define in more detail the types of activities that exist inside organisations (fig. 1.a). The business processes are designed with the aim of bringing value to the organisation. This value must be quantified to serve as a basis to trace the process state and possible improvements. Some general indicators for each process were proposed [3]: productivity, added value, cycle time, queue size, quality indices.

In the management category (top of fig. 1.a), 3 business processes are proposed:

- Global Strategy (GS): Within this top-level management process, potential clients are evaluated, the innovations are planned, and the general policy for the enterprise, such as its mission, its vision, its principles, its values, and its long-term objectives, is defined.
- Policy Deployment (PD): The organisation's policy and short-term objectives, usually defined in an annual basis, are unfolded and delivered to the competence centres and processes.
- Business Plan (BP): The plan and the budget of the activities and their distribution among processes and competence centres are defined. Normally, these activities are

planned with an annual scope; the activities for controlling the execution of the plan are also defined.

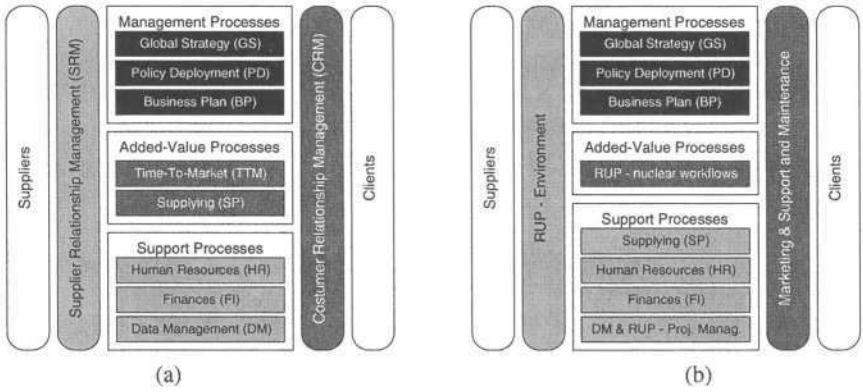


Fig. 1. (a) General composition of the processes within a generic organisation; (b) General framework for the processes of organisations that develop software.

In the added-value category (center of fig. 1.a), there are 3 business processes:

- Time-To-Market (TTM): This represents the development process associated to a new product and the environment needed to produce it.
- Supplying (SP): This process occurs normally after the TTM for the same product, and starts with a request from the client, or with a decision of production. It ends with the deliver of a product to the client with the fulfilment of the required schedules and quality levels.
- Customer Relationship Management (CRM)[1]: This process includes all activities that allow clients to be satisfied with products or services supplied to them.

In the support category (lower part of fig. 1 .a), 4 business processes are suggested:

- Supplier Relationship Management (SRM)[2]: This includes activities such as the selection of the materials' suppliers, the execution of audits to verify the quality, or the establishment of agreements for just-in-time delivers.
- Human Resources (HR): This process includes all issues associated with the well-being and qualifications of the organisation's collaborators.
- Finances (FI): All the activities related to fiscal issues belong to this process.
- Data Management (DM): This process includes activities associated to data quality and the indicators of the organisation, the security of data, the priority of computer related projects and their budget control, data-warehousing and data-mining.

[1] Although the CRM process is mainly seen as an interface process, it was also included in the added-value category, because client-oriented organisations must address the customer needs.

[2] The SRM process is mainly classified as an interface process, but it was also included in the support category, because it generates inputs for added-value processes.

The activities related to the continuous improvement, to the quality and environment management including certification tasks, are all examples of activities that tend to be confused with processes. These activities are part of all processes, such as those of fig. 1 .a, and should not have a separate management.

3 Framework for Organisations That Develop Software

Since the framework presented in fig. 1 .a describes a generic organisation, it can also serve as a basis for modelling an organisation that develops software. Being a more specific type of organisation, we can add more details and propose more specific processes to the processes presented in the generic framework.

We propose the instantiations of the business processes within the organisations that develop software (fig. 1).

Therefore, the processes named Global Strategy (GS), Policy Deployment (PD), and Business Plan (BP) are equivalent to those of any other organisation.

Since software is an intangible product, it is obvious that no raw materials are needed to produce it. For organisations that develop software, the SRM process is instantiated by the RUP's Environment discipline, since it furnishes the working environment (e.g. development tools), which will be used by the teams, and the development guidelines to be followed within the organisation.

The Supplying (SP) process consists essentially in creating copies of an application. In contrast with more traditional industries, where it represents probably the most important process, in software, due again to its intangible nature, this is not an added value process. Usually, the kind of materials used to support it, DVD and printed manuals, imply that external suppliers are hired.

The Time-To-Market (TTM) process is mapped into the set of RUP's core disciplines: Business Modeling, Requirements, Analysis and Design, Implementation, Test, and Deployment. This set of activities, or sub-processes, run in parallel for the same development project [4]. In our opinion, this is the most critical process for an organisation that develops software.

The Human Resources (HR) process for software factories is the same as for other types of organisations. We must however point out that software development requires highly-specialized people, being their hiring a critical issue for the success of the organisation. It is impossible to produce quality software without skilled people.

The Finances (FI) process is the typical fulfillment of the fiscal obligations, which is common to all types of organisations.

The Customer Relationship Management (CRM) process is decomposed into the sub-processes: Marketing, and Maintenance and Support. This ensures that, when a software application is delivered to the final clients, its life-cycle does not end, but instead continues with this process, incorporating changes and corrections and providing training to the users, until the application is no longer used by the client. The Marketing assumes, in this case, a form similar to that observed in other types of organisations.

The Data Management (DM) process, in what concerns the data related to the RUP's core disciplines, is mapped into the RUP's Project Management discipline. In this discipline, some activities lead to the production of indicators of the project status. Its

existence is the foundation to take decisions based on facts, related to the progress of the project and also to adjust and improve the software development process.

To reach the highest CMM levels [5], the continuous improvement of the development processes must be part of each process, instead of being a single autonomous process.

RUP's core disciplines, that in fig. 1.b implement the added-value process, are subdivided in activities, which can be viewed as sub-processes. The description of those sub-processes is made with UML activity diagrams, complemented optionally with other type of diagrams, such as interaction diagrams and business object diagrams. This representation is also valid for all other processes of a generic organisation, as illustrated in fig. 1.a. Whenever an organisation that develops software executes a development process, the TTM process will be executed. Since we are proposing this process to be implemented by the six RUP's core disciplines, it implies that the Business Modelling discipline can also be executed. This discipline strongly recommends the usage of activity diagrams (and other complementing diagrams) for modelling purposes. Therefore, the target organisations, as proposed in fig. 1 .a, will be modelled with a collection of these diagrams. In parallel, within the organisation that develops software, since the Business Modelling discipline is a sub-process of the TTM process, it can also be modelled by activity diagrams.

4 Business Process Modeling

In this section we concentrate on 'Business Modelling', because of the six RUP's core disciplines (Business Modelling, Requirements, Analysis and Design, Implementation, Test, and Deliver), it is the one most directly related to the client' needs. During the development of software, all the stakeholders must have a common understanding of the business processes that exist in the target organisation. This reality is not circumscribed to the obvious organisational information systems, but can also include real-time embedded systems if they affect the business processes.

Within RUP, the business processes model is obtained in the Business Modelling discipline. The main activities in this discipline are centred around the identification, refinement, and realisation of the business processes and in the definition of the roles of people associated to the business. Each role in this RUP's discipline has under its responsibility the execution of several activities that will have as deliverables several artifacts (tab. 1). For example, the activity 'Refine Business Processes' includes the activities 'Structure the business use case model', 'Review the business use case model', and 'Detail business use cases'.

Among all activities and their respective artifacts, only some of them are mandatory. This flexibility permits the configuration of RUP, so that it can be adapted to a specific project executed in a specific organisation. Thus, taking into account the options made when choosing the artifacts, the following set allows modelling business processes [4].

Business Vision: This artifact captures the goals of a particular business modelling activity. It should be stated 'Why' and 'What' are to be modelled. It also serves as an oracle to all future high level decisions.

Business Glossary: In this artifact all business terms and expressions are kept.

Business Rules: The business rules correspond to policy statements and conditions that should be fulfilled, from the business perspective. They are similar to systems' requirements, but they focus on the business core, expressing rules related to business, but also its architecture and style. Its modeling must be rigorous, being one possibility the usage of the Object Constraint Language (OCL) as specified in UML [6]. Alternatively, using a natural language in a structured form is also admissible, since even though there are clear advantages in using a more formal approach, the need to allow its understanding by all the stakeholders is usually paramount.

Business Use Case Model: The main goal of this artifact is to show how the business is being used by all stakeholders. This is achieved by modeling the business processes and their interactions with external parties, based on use case diagrams (with stereotypes for business use cases and business actors) [7]. The business processes are associated to a discipline that needs to be modeled, and that specify how added-value is created for the business actors. This modeling can be supported by activity diagrams, possibly extended with the representation of organisational units interfering in the business process and with the distribution of the activities by those organisational units.

Table 1. Roles, activities and artifacts for business modeling in RUP.

Roles	Activities	Artifacts
Analyst of the Business Process	Verify target organisation	Business rules
	Establish and adjust objectives	Business use case model
	Capture the vocabulary of the business	Business Glossary
	Find business actors and use cases	Business object model
	Maintain the business rules	Business vision
	Structure the business use case model	Supplementary business specification
	Define the business architecture	Target organisation verification
		Business architecture
Reviewer of the Business Model	Review the business use case model	
	Review the business object model	
Designer of the Business	Detail business use cases	Organisational units
	Find business workers and entities	
	Define the automation requirements	
	Detail business entities	
	Detail business workers	

5 Case Studies

The proposals made in this paper are being introduced in the second author's organisation, in what concerns the organisation's structure and the adoption of a RUP-based process and UML. At the moment, two software projects were conducted following the approach suggested here with promising results, and we next report on some of the lessons learned from these projects.

The first project (Travel Management) allows to validate the importance of using adequate methods for capturing requirements and for modelling them using a notation that clients can easily read. It also helps to understand the challenge of adopting standard software. In the second project (Premium Wage), the selected development process is more extensively used and we evaluate the capacity of the process to cope with complex organisations. The final application requires the complete re-engineering of some business processes, the extension of functionalities into new business processes, and finally the design and implementation of a new business process and its supporting application.

5.1 Travel Management

The first project was the development of a travel management system. The need and motivation for implementing a computer-based system to manage the travels, mainly their planning and the payment of the respective expenses, is based on the fact that previously this process occurred quite often, but had no automatic support. These business processes, although frequently executed, do not belong to the set of core processes.

This project had a requirement from the target organisation of being implemented in the SAP R/3 ERP platform. This imposition derives from the strategic decisions made by the top managers of the target organisation:

- Minimization of the number of distinct computing platforms within the same organisation.
- Reduction of the technical knowledge needed to accomplish maintenance activities.
- Reduction of the number and types of tools that the final users need to know.
- Improvement of the specialization level, and consequently the performance, of the IT personnel and final users.
- Economic advantages obtained by negotiating a big quantity of software licenses.
- Support advantages. The organisation is not a small client of several suppliers, but is instead a big client of only one supplier. This advantage must be weighted against the risk of dependency.

This project was chosen by the organisation that developed the software to introduce the UML graphical notation, with the following motivations:

- To improve the quality of the documentation from previous projects (incomplete textual descriptions in scope and detail).
- To improve the inter-communication among all the stakeholders.
- To improve the rigor of the documented information.
- To augment the quality of the final product, being this understood as the degree of fulfillment of the client's requirements.
- To ease the support for the final application.

Typically, ASAP [8] is adopted as the development process in SAP projects, but in this project we decided to adopt RUP. Although ASAP is adequate for SAP-based solutions, it has some weak aspects that are not desirable in modern processes, namely the fact that the current (As Is) and the future (To Be) situations are mostly described in a free textual form, and also the difficulty in managing the requirements change.

Results Analysis. We next analyze the way RUP and UML were used in this project. The main positive points that were identified on the execution of this project are the following:

- It is mandatory to assess the risks and the respective correction and contention actions. This activity reduces the occurrence of the potential problems and also minimizes their negative impacts when they do occur.
- The effort to design and implement the architecture of the final solution was small, since it was already available and documented.
- The introduction of RUP, in an organisation with a maturity level of its software development process not higher that CMM-3 [8], requires developers to understand that developing software is not only writing code, but includes also other activities that have an important impact on the quality of the final product.
- The quality of the documentation improved in quantity, detail, and rigor, being also an example for the rest of the organisation and allowing a better maintenance and support.
- The stakeholders meetings improved in productivity. Quicker results (requirements definition) were achieved in less time.
- The client's requirements were satisfactorily implemented.

As negative points, we identified the following ones:

- It is difficult to map the requirements on the functionalities already available in the standard ERP software.
- It is necessary to have a deep knowledge about the standard ERP software's process reference model.
- Some members of the development team try to resist to the introduction of a new process, such as RUP.
- The lack of a study about the economic impact of an application (i.e. its return on investment) induces some clients to just evaluate the costs, when the application is delivered. This fact is even more evident whenever the target architecture is standard software.
- If the quality assessments are informally conducted and without the participation of all stakeholders, the project status may be imprecisely perceived.
- RUP business modeling was not conveniently executed. The AsIs description was made in text (based on a already available guideline) and the ToBe situation was described only with use cases and their realisations in the final system (not in the business).

Lessons Learned. The main lessons learned with this project were:

- The adoption of standard software solutions (like ERP) implies an extra activity of conflict negotiation between the client's needs and the functionalities offered by the solution. Although it is possible and not difficult to implement changes to a ERP, usually this is not a good idea since it implies that in the next upgrade of the product all these customised solutions have to be evaluated to be transported, discarded, or

changed. The conflict negotiation can only be conducted by software development team members, because typically the clients do not have a deep knowledge about the ERP's functionalities and customisation.
- RUP introduction should be made in a project were the high level artifacts (e.g. business use case) have an easy and understandable transposition to lower level artifacts (e.g. source code). Without this attribute, RUP can be misunderstood either by clients and development team, and be wrongly seen as an paper generator.
- RUP introduction should not be made by hierarchical (from top to bottom) imposition, but through presentations explaining its potentials. This helps the development team to have a proactive behaviour during the usage of RUP.
- It is impossible to introduce RUP without a training effort both in UML and RUP.
- The target organisation has a deep influence on the software development organisation. Problems like the replacement of key members and the inexistence of complete definitions for requirements, which are under clients' responsibility, may have a negative impact on the software development organisation.
- Quality verifications (like the milestones for phase transition inside RUP) should be formally executed and documented, and being carried out with the presence of all stakeholders.
- The Business Modelling discipline is vital for the success of the final application and should be executed in all projects that involve business applications.

5.2 Premium Wage

The second project was the development of a software system to calculate the payment of extra money to employees, based on their productivity. This project was classified as critical, since it may have important social and behavioral impacts on the organisation, if the amount is badly calculated or if it is impossible to explain how it was obtained. The adoption of this system is to improve the organisation's overall productivity.

Besides its criticality, the business process is also complex due to its dependency from other processes. In the present case, the payment of a premium depends on three main factors: individual absenteeism, quality of the products produced in his line, and individual performance. The first two sub-processes were extended in order to support new functionalities required. The third, a complete reengineering was carried on. Finally, to the premium wage values calculation a completely new process was designed, modeled, and implemented.

In the project the selected development process is extensively used and we evaluate the capacity of the process to cope with complex organisations. The final application requires the complete re-engineering of some business processes, the extension of functionalities into new business processes, and finally the design and implementation of a new business process and its supporting application.

Business Modeling. The technology-planning horizon for big companies is now a synthesis of software engineering and process engineering [9]. Additionally, the success of a project depends heavily on the correct perception of the business process to be modeled. Taking into account these two aspects, the RUP's Business Modeling discipline

assumes a critical role in the software development process and therefore a special attention is paid to it in this section.

This discipline can generate the artifacts presentd in table 2, and has as objectives the following:

- To understand the structure and dynamics of the organisation where the system will be executed.
- To comprehend the current problems of the target organisation and to identify potential improvements.
- To assure that clients, final users, and developers have a common understanding about the target organisation.
- To capture/deduct the requirements of the system necessary to support the target organisation.

The procedure taken for configuring this discipline in this case study was was also adopted for describing the other RUP's disciplines.

One way of parameterising RUP is by choosing which artifacts to use and their level of detail. On table 2, we show which artifacts on the Business Modeling discipline were used in the project and the arguments considered to decide on its usage. This choice was validated by the quality assessments. Both the subset of used artifacts and also its degree of detail can not be anticipated with rigor, but must be selected based on experience and knowledge of the development team in relating the characteristics of each project with the functionalities offered by the artifacts.

Table 2. Artifacts in business modeling.

Artifact	Used	Motivation
Business Rules	yes	Rigor in describing a new business process
Business Use Case Model	yes	First description of the functionalities and the organisation's business actors
Business Glossary	no	The business terms are common to the target and software development organisations (they are sub-organisations of the same organisation)
Business Object Model	yes	Realization of the Business Use Cases
Business Vision	no	The business vision is common to the target and software development organisation
Supplementary Business Specification	no	Business Use Case Model and the Business Object Model are sufficient
Target Organisation Verification	no	The target organisation is perfectly known by the developers. The current are modelled by the Business Use Case Model - Current Situation
Business Architecture Document	no	The details present in the Business Use Case Model and in the Business Object Model are sufficient
Organisational Units	yes	Mapping the business process functionalities on the target organisation's structure

The criteria to fulfill this choice are related with:

- Characteristics of the project itself (e.g. criticality of the modelled business processes, type of target organisation).
- Characteristics of the organisation that develops software (e.g. team size, level of knowledge about internal rules).
- Temporal restrictions. Since resources are limited in engineering projects, it is always necessary a balance between the quantity and detail of the produced artifacts and the deadlines for implementing the project.

The produced artifacts result from a set of activities that occur inside those disciplines. In this case study, we identified the need for the artifacts to represent two distinct situations in terms of business: one part of the project represents reengineering activities of some business processes, while the other part represents the introduction of a new business process.

In several diagrams (e.g. Business Use Case Model), the standard UML is augmented with the stereotypes defined by RUP, thus allowing the creation of RUP-like artifacts. Other techniques not included in RUP, such as the classification of requirements FURPS+ [10] or the planning of maintenance teams [11] are also used.

Other Relevant Artifacts. Besides the artefacts suggested by RUP, we believe that it is also important to document other issues associated with the development of a software application:

- List of verification to assess the project status in the Quality Verifications that evaluate the transitions between the process phases (inception, elaboration, construction, transition).
- Planning and analysis of the required capacity to support and maintain the system when it is in the productive phase.

For this case study, we estimated the following values:

STEP 1. Estimate the requests volume (RV) with the table 3: Requests Volume (r/d)= $1.46 \times 0.56 = 0.82$. We estimate that each request is served by one single person.

Table 3. Estimation of Requests Volume.

Organisational unit	Requests / day	Resolution Time (days)
Productivity by chain	1	1/8
Quality VQ	1/3	1/16
Presences in Post	1/22	1/8
Individual Performance	1/22	1/8
Premium Wage Calculation	1/22	1/8
TOTAL	1.46	0.56

STEP 2. Estimate the adequate level of preparation (LP): For the application under consideration, it was agreed to use the value of 80%.

STEP 3. Define the base number of persons: Using the suggestions in [11], with $RV = 0.82$ and $LP = 80\%$ we obtain a value less or equal to 2 persons.

STEP 4. Add extra capacity. For this case study, we consider that 2 persons is enough, since the organisation has already experience in dealing with similar systems.

STEP 5. Establish a strategy to diminish the number of requests in the queue. For the present case study, the following issues were considered relevant: (i) the requests from the organisational units 'Productivity per Line' and 'Calculate Prize Money' have higher priority; (ii) requests from other organisational units have a normal priority; (iii) high-priority requests are always served in first place.

Queues with separate priorities for each subsystem can also be created, but this alternative was not followed, since we intended to have a global estimation for the support and maintenance of the application responsible for the complete business process.

Results Analysis. We now analyse the positive and negative points associated with the parameterisation of RUP to implement the application under development. The main positive issues were:

- RUP, but especially the artefacts of the Business Modeling discipline, had a good performance in modelling the business processes, since they served as a basis for the artefacts at lower levels.
- The UML artefacts, with the stereotypes suggested by RUP, were a very efficient communication medium among all the stakeholders.
- The artefacts generated by RUP, namely the source code, have an easy utilisation for future systems that need them, allowing thus the reuse of code.
- The development process standardisation in the organisation that develops software was initiated to adopt the usage of RUP.
- RUP proved that it can adapt to the needs of organisations that develop software and specific projects.
- The set of RUP's disciplines allows to cope with the complexity and ensures that no important aspect related to the software development is forget, diminishing the risk of failure.

We also identified the following main negative aspects:

- The usage of RUP and UML must be accompanied with training for the developers.
- The need to create new roles to execute the software development process, such as the business process designer or the tools expert, forces internal reengineering.

Lessons Learned. The lessons learned with this second project were the following:

- It is very important that all stakeholders be aware of the tasks related to the use of a new software product. In this way all stakeholders, as suggested by RUP, should know and validate all the requirements.

- The transformation of use cases into class diagrams was accomplished via the following methodology [7]: (1) Transform system use cases into classes of 3 categories: Interface, Control, Data; (2) From the full set of classes generated, only the meaningful (depending of the description and use case realisation) are kept; (3) Aggregate the similar remaining classes (e.g. database-related classes); (4) Create associations between the groups of classes. From the remaining class groups, a mapping into a 3-tier architecture was inferred. The class groups, Interface, Control, and Data, were distributed into the three tiers, Presentation, Business, and Data, respectively. This method provided a mechanisation that lead to a time improvement in reaching the final architecture for the system.
- In matrix organisations, some problems, external to the clients and to the software development teams, arise just because of the existence of departments. When the departments are among the stakeholders for a project, they block the software development, because it will model business processes that can threaten the existence of tasks inside departments or even the departments.
- The RUP usage, even in the first projects, is not itself a delay factor to reach the final solution.

6 Conclusions and Future Work

The main conclusions for our work, based on the two case studies are:

- For an organisation represented by processes, RUP proved useful for modelling it and to mapping its business processes into software applications to support them.
- RUP can be used for systems already in execution, developed without a process-oriented perspective, allowing their integration into a new system, by reengineering the business processes, through the business use case diagrams for current and future situations.
- A good understanding and documentation of business processes is essential to build quality software systems and RUP recommends and supplies support for this discipline to be properly conducted.
- RUP configuration for a given project is essential for its success. This is a task mainly based on experience from past projects.
- The quality of personnel is the most important issue for the final quality of the software product.
- The Business Modelling discipline in RUP can be the basis for the business processes reengineering. The comparison of current and future situation on Business Modelling and the implementation of future situation into software systems are the basis for that. Also, as future situation tends to be implemented into software system, besides the business reengineering an improvement of processes performance can be achieved via the usage of software systems where manually processes existed.
- The structure and support during the project phase are key factors for the correct software development process and for the success in its utilisation.
- The main added value for a software development organisation is the correct business modelling of target organisations. The remaining tasks (such as programming the applications or testing) tend to be more easily automated and executed with supporting tools.

- The maintenance and support planning for a software system should be planned during its development process.
- The software development organisation must provide guarantees to its clients. Its own structure, the software development process used, and the quality of its collaborators are fundamental factors to achieve it.

As future work, we foresee the following. Instantiated specifications of software development organisation model (e.g. for CMM-2 or higher levels) can be provided. This way, the transition of existing organisations into reengineered ones could be performed more easily using to a current Business Modelling situation and a future situation described by the already available templates. The RUP parameterisation for a specific project, running in a specific software development organisation, and for a specific target organisation is based mostly on empirical decisions. An auxiliary process to parameterise it would provide benefits to all stakeholders as well permit its use by organisations that do not have expertise to configure it. Finally, the existence of a simulation environment to run current and future situation business processes, to test them, to evaluate them through business performance indicators, and to preview the impacts of reengineering would provide a valuable framework for business modelling decisions on future situations.

References

1. João M. Fernandes and Francisco J. Duarte. A Reference Model for Process-Oriented Software Development Organizations. In C. Gonzalez Perez, B. Henderson-Sellers, and D. Rawsthorne, editors, *OOPSLA 2003 Workshop on Process Engineering for Object-Oriented and Component-Based Development,* pages 31–42. COTAR, October 2003.
2. Michael Hammer. *Beyond Reengineering: How the Process-Centered Organization Is Changing Our Work and Our Lives.* Harper Collins, 1996.
3. C. Marshall. *Enterprise Modeling with UML: Designing Successful Software through Business Analysis.* Object Technology. Addison-Wesley, 2000.
4. Rational Software Corporation. Rational Unified Process: Product Overview, August 2003. http://www.rational.com/products/rup.
5. M. Paulk, B. Curtis, M. Chrissis, and C. Weber. Capability Maturity Model for Software, Version 1.1, 1996. Software Engineering Institute, Carnegie Mellon University.
6. Object Management Group. OMG Unified Modeling Language Specification version 1.4, 2001. version 2001.03.00.
7. João M. Fernandes and Ricardo J. Machado. From Use Cases to Objects: An Industrial Information Systems Case Study Analysis. In *7th International Conference on Object-Oriented Information Systems (OOIS '01),* pages 319–28. Springer-Verlag, August 2001.
8. SAP. Training Manuals ABAP/4, 2000. SAP Portugal.
9. Howard Smith and Peter Fingar. *Business Process Management: the Third Wave.* Meghan-Kiffer Press, 2002.
10. R. Grady. *Practical Software Metrics for Project Management and Process Improvement.* Prentice Hall, 1992.
11. R. Ramaswamy. How to Staff Business-Critical Maintenance Projects. *IEEE Software,* 17(3):90–4, 2000.

Web-Based System Development: Status in the Norwegian IT Organizations

Jianyun Zhou and Tor Stålhane

Department of Computer and Information Science, Norwegian University of Science and Technology, 7491 Trondheim, Norway
{jianyun, stalhane}@idi.ntnu.no

Abstract. Software process improvement seeks for better methods and techniques to develop quality products with reduced time. A prerequisite for this is to understand the current status and problems. In this paper we present a survey that gives an overall picture of the status in the development of Web-based systems. We investigated how the recognized best practice in the software community is employed in WebSys development, with respect to time-to-market and quality requirements - reliability and robustness. Data have been collected through questionnaires and interviews. Exploratory data analysis is used to discover patterns underlying data through successive iterations. Ten main findings are presented in three groups: features of the WebSys development projects, results related to time-to-market and use of engineering methods for reliability and robustness. Based on these findings, some key research areas are identified.

1 Introduction

In recent years, the World Wide Web (WWW) has become a very popular platform for developing Internet applications, due to its powerful communication paradigm and open architectural standards. This leads to a crucial question: can software engineering principles, concepts, methods, and best practice be applied to Web development? Most researchers agree that Web-based systems (hereafter referred as WebSys) are different from most other types of software systems, and there is an urgent need for the disciplined and systematic approaches to the successful development, deployment and maintenance of high quality Web-based systems [19].

In the WebSys development process, there is a large number of contradicting quality requirements [20]. For example, there may be performance constraints, security constraints, and usability constraints. However, an important industry requirement is shorter time-to-market. The importance of having a short product development time has been argued by many, e.g. [7, 21, 28]. Being early to market increases access to the market introduction window, which is usually considered as a winning strategy in today's fast-pace industry [7]. Coming too late to the market place may lead to incredible economic losses. An example is given in [3], where a company witnessed that being only one week too late to market resulted in the total annihilation of the

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 363–377, 2004.

potential market, and several million dollars of software investments were wasted. For WebSys, time-to-market pressure is even much higher than for any other type of software, which is described as immediacy in [12]. That is, time-to-market for a complete Web-based system can be a matter of a few days or weeks [22].

Another important quality requirement for WebSys is reliability, or more specific, robustness. Robustness is defined as "The degree to which a system or component can function correctly in the presence of invalid input or stressful environmental conditions" in [10]. Robustness is considered to be a critical factor for WebSys. Firstly, Web-based systems are accessed via the HTTP protocol, which has made such systems almost available for everyone. It is difficult, if not impossible, to control the population and input profile of end users and to limit the type of browsers. Web-based systems must have tolerance to errors and abnormal situations from the client side. Secondly, Web-based systems are often not separately developed, usually integrating existing subsystems (components or legacy systems), which are not specifically produced for Web-based system. Web-based systems must have tolerance to errors and abnormal situations caused by internal component failure.

These two requirements are difficult to satisfy at the same time. In an effort to be "first to market", reliability and robustness may be scarified; while much emphasis on quality often leads to "late to market" and later makes it difficult to acquire market share. As time-to-market has become a dominating factor for industrial success, developers often spend less effort on quality and follow ad hoc [19] approach for developing WebSys.

Many have argued that if the ad hoc approach persists, it will lead to a "tangled Web" [23]. In order to avoid this, Web engineering, as a new branch of software engineering, has emerged. Many new methods and techniques have been proposed. However, the situation so far has not been improved much. We argue that one important reason is the absence of solid research in this field. Both researchers and practitioners lack understanding and insights of the actual status and problems. Compared to software engineering, where empirical software engineering is gaining more and more interests, there is little empirical data to report the status in WebSys development and analyze the impact of the proposed methods and techniques.

In this paper we report the results from an exploratory survey conducted among eleven IT organizations in Norway. The purpose of this survey is to explore and form an overall picture of the status and challenges in current WebSys development, with respect to how time-to-market and reliability, especially robustness requirements are addressed and engineered, and how the best practice from the software community are employed. The goal is to understand whether there are weakness and problems in the development process, which are common to different organizations. If any is found, the focus should be on finding solutions to such areas. In Section 2, the scope of the survey is described and the derived research questions are defined. Section 3 gives details about the methods applied in the investigation. In Section 4 the results of the survey are presented, and ten facts about WebSys development are disclosed and analyzed. Section 5 summarizes the findings and outlines areas for future research.

2 Research Questions

We define the scope of this survey according to a goal definition framework GQM [4, 26]: *Characterize* how IT organizations work in the development process *for the purpose of* exploring and revealing the actual status: what is good and what is bad, *with respect to* time-to-market and reliability (robustness) requirements *from the point view of* the developers *in the context of* Web-based systems. The more focused research questions derived are:

- **Q1:** What are the features of the WebSys development projects in terms of application types, implementation technology, schedule and resources?
- **Q2:** How is time distributed in the development process? How are the recognized ways of decreasing development time such as iterative development, process planning and reuse used?
- **Q3:** How are the engineering methods for reliability and robustness used in WebSys development? Has robustness been considered as an important quality factor by developers?

3 Methods

3.1 Data Collection

A survey was conducted to obtain answers to the research questions described in Section 2. The process went through literature review, questionnaire preparation and refinement, execution, raw data collection and validation and data analysis and presentation.

Participants in this survey are employees (either project managers or developers) from eleven IT organizations in Norway, who have developed WebSys for at least five years. The selection of participating organizations was based mainly on convenience sampling [6], as most of them were collaborative partners of the institute. In addition, to improve precision of the results, stratified sampling was used to make it possible to distribute the selected companies along a set of properties such as business scope and geographical location. These sampling techniques were prioritized due to the ease of manipulation. For the purpose of exploratory research, they are useful, while not very reliable, to present evidence and gain insights about the interesting topics, since the sample was assumed to be representative of the Norwegian IT industry in the domain of WebSys development.

Data collection was conducted by questionnaires, supplemented with interviews. Firstly, a preliminary version of the questionnaire was tested in interviews with two selected companies. The questionnaire was modified and improved on the basis of input and feedback from these two interviews. The final version of the questionnaire was divided into three sections: features of the development projects, methods for shorter development time, and methods for reliability and robustness.

The final questionnaire was sent to all the participants by electronic mail. The respondents have been encouraged to write additional comments. The waiting time for answers from the companies varied from several days to several weeks. We did not take any accelerating measures in the process. Upon each reception of the response, the answers were edited to detect possible errors and omissions. As a result, some companies were asked to review their answers, at least on some questions, and some companies were invited for focused interview [6] to provide more inputs.

3.2 Data Analysis and Presentation

For analyzing the quantitative data collected through the questionnaires, the main technique employed was exploratory data analysis (EDA). Exploratory data analysis is both a data analysis perspective and a set of techniques [27] with the emphasis on visual representations and graphical techniques. In the exploratory data analysis, the data guides the choice of analysis rather than that the planned analysis is enforced on the data. This is comparable to the situation that our research should be problem-oriented rather than tool-oriented [6]. Focusing on data makes us flexible and able to respond to the underlying patterns of the data and to search for clues and evidence by successive iterations in the analysis process.

To present data and results, we use several kinds of visual charts such as bar chart, pie chart and line chart. The unstructured information obtained in comments and interviews is used as qualitative data to explain results and reveal indications.

4 Results and Findings

In this section, the results of the survey are presented. Where the results show the facts and evidences about WebSys development, specific results are denoted as **Fn**. We found ten facts. Section 4.1 describes the features of the current WebSys development projects. Section 4.2 presents the results related to development time, and section 4.3 presents the use of reliability and robustness engineering methods in WebSys development process.

4.1 Features of the WebSys Development Projects

To identify solutions for process improvement, it is essential to know the features of the development projects, which affect the framework underlying the process. This is the focus of the first part of the questionnaire. In the following we present three important findings concerning project features, such as application types, implementation technology, time schedule and personnel resources.

F1: Investigated organizations involve the development of different kinds of WebSys, from the simplest to the most complicated; most often developed systems are service-oriented applications.

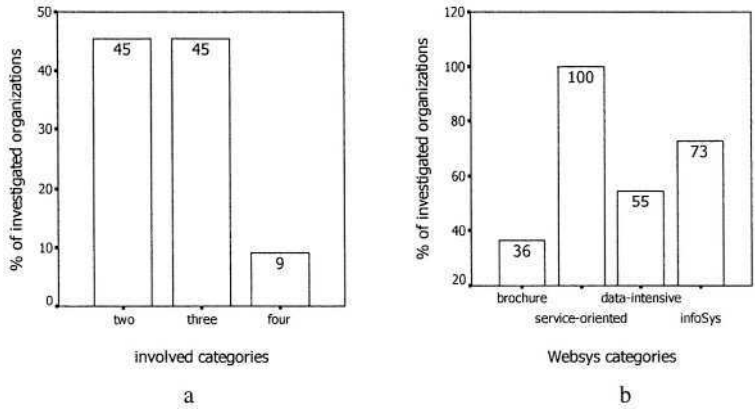


Fig. 1. a: Involved WebSys categories; b: Categories of WebSys developed in organizations

WebSys are not limited to one type. They range from simple web pages to complex applications. Different systems can be grouped into brochure, service-oriented applications, data-intensive applications, and information system applications according to their data and control complexity [1]. The result in Figure 1a shows that none of the organizations develop a single type of systems; most of them are involved in two or three types. Among the four categories of applications, service-oriented application is developed by all the investigated organizations, as showed in Figure 1b. Such systems, most often mentioned in the questionnaire comments, are e-commerce applications, entertainment applications and games. This fact indicates that the development process and process improvement adopted by organizations developing WebSys should be flexible and possible to be adapted to different target systems. Another important aspect significantly affected by the application types is the software architecture. The choice of architectural design and modeling techniques has important impact on the total development time and quality of the final product.

F2: A myriad of Web-related technology is mixed in WebSys development. Object Web and Web services represent the main future interests.

The low-level implementation technology for Web-based systems has evolved quickly in the past few years. According to the functionality, a myriad of technology falls into four categories: static web (e.g. HTML), simple-response web (e.g. CGI, PERL, PHP), Object Web (e.g. JavaBeans, CORBA, applet), and Web services. The result in Figure 2a shows that the technology is highly mixed in WebSys development. When the respondents were asked for their viewpoints on the future dominating technology, sound for Object Web and Web Services is high, respectively 100% and 91%, as illustrated in Figure 2b.

F3: WebSys project tends to be short and small.

In the survey, the respondents were asked for their experience of the longest project, the shortest project, the largest project, the smallest project, and their estimation of average project time and project size. The table in Figure 3 gives the descriptive

statistics from the questionnaires. The average longest time is 18.8 months, shortest is 1.6 months, average is 3.5 months, while the average largest size is 11.6 persons, smallest is 1.5 and average is 3.

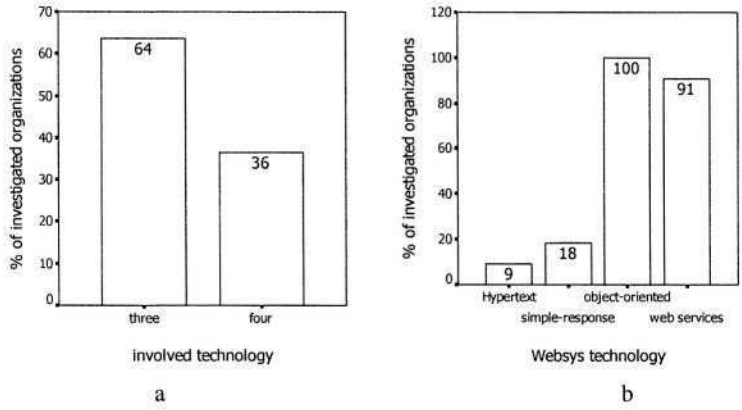


Fig. 2. a: Involved WebSys technology; b: Future interests in WebSys technology

	N	Minimum	Maximum	Mean	Std. Deviation
longest project	11	6,00	48,00	18,8182	14,26757
shortest project	11	,03	6,00	1,6191	1,94880
average time	11	,50	12,00	3,5455	3,33320
biggest project	11	3,00	52,00	11,6364	13,62551
smallest project	11	1,00	3,00	1,5455	,68755
average size	11	2,00	5,00	3,0000	1,09545
Valid N (listwise)	11				

Fig. 3. Descriptive statistics for project time and project size

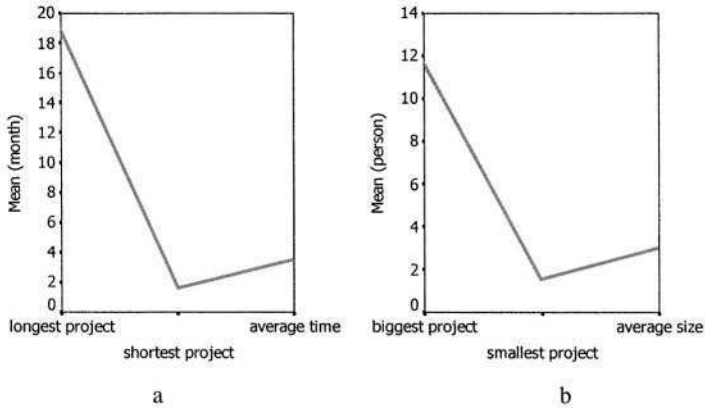


Fig. 4. a: Mean project time; b: Mean project size

If the results are presented in line charts, the underlying data pattern becomes much clearer. Figure 4a reveals that the average time of WebSys project is significantly biased towards the shortest projects. The same pattern exists for project size, as showed in Figure 4b.

4.2 Results Related to Development Time

There are many ways of decreasing time in a software development project. Among the strategies suggested by software engineering best practice, we investigated how iterative development, process planning and reuse are used in the organizations. First of all, let's have a look at time distribution in the different development phases.

F4: There is agreement on that the development phase (implementation and testing) is the most time-consuming phase and the requirements phase is the least time-consuming phase.

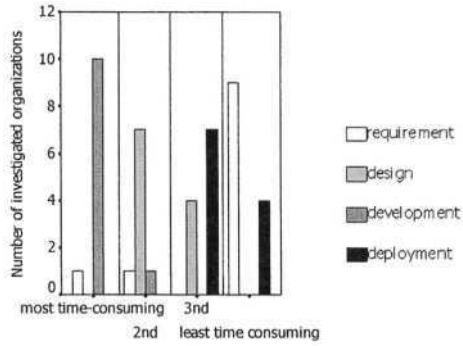


Fig. 5. Time distribution in the WebSys projects

The respondents have been asked to provide their experience concerning time distribution among the major phases in WebSys development. These phases have been selected to comply with the phases provided in [15]. The clustered bar chart in Figure 5 shows that there is large agreement among participants on that the development phase is the most time-consuming (10 out of 11) and the requirements phase is the least time-consuming (9 out of 11).

This result indicates the development phase is, more or less, a bottleneck in the whole process. One obvious solution is to find more efficient methods and tools to support activities in this phase, which in turn can result in a shorter development cycle. Another solution can be derived from the best practice in the software community, which recognizes that additional investment in requirements and design phases will result in time saving in the overall development time [24].

F5: Iterative development is popular in the investigated organizations, while the development process is still ad hoc to a large extent.

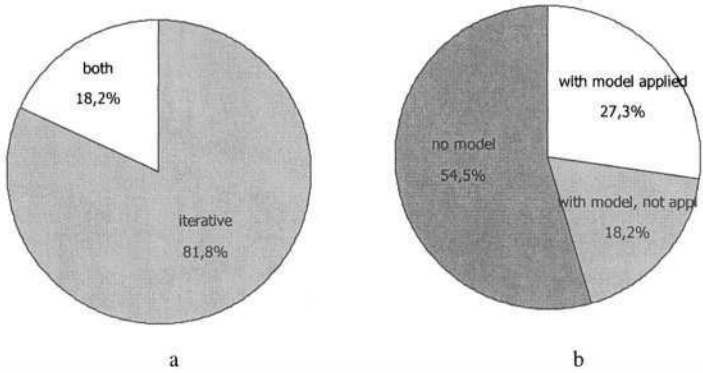


Fig. 6. a: Iterative versus monolithic development; b: With process model versus without process model

In the survey, the respondents were asked whether they use monolithic or iterative development approach. Figure 6a shows that iterative approach is used to quite a large extent. This can be considered as a natural response to the F3 to handle compressed time pressure, as there is growing recognition that an iterative and incremental approach to software development is often more suitable and less risky than monolithic waterfall approach [9].

In Figure 6b, 54.5% of respondents stated that their organizations do not have a process model at all; 18.2% stated that they have a process model but seldom apply it in practice; while only 27.3% of respondents stated that they develop WebSys with a defined process model. In other words, despite the emphasis in the literature on the importance of the process model, it is little applied in practice. The same result is found in [8] and it argued that developers often have an inner resistance against defined process and the change of process. It is also common that defined processes are disregarded or deviated from [5]. To solve these problems, process management is required. The key factors for successful process management are identified in [2]. They are change management, synchronization, baselining the current way of working, documentation, process user involvement, and management commitment.

F6: Process planning issues such as scheduling, work estimation and milestones are well addressed in the investigated organizations.

Respondents were asked to rate the extent to which their organizations conduct scheduling, work estimation and milestones during process planning. The result in Figure 7 shows that the mean values of these variables are quite large, which means that the level of use of these approaches is high. Through content analysis [25] of the respondents′ comments on the questions, two factors are identified to contribute to the high application of these approaches: one is the overall organizational culture; and the other is the time pressure on projects.

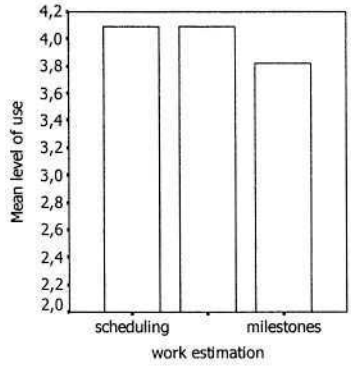


Fig. 7. Application of process planning issues

F7: The overall level of reuse in WebSys development process is low.

The concept of reuse has been widely discussed in the literature and there are several studies that have demonstrated its actual benefits [11, 13, 29]. The major motivation for reusing software artifacts is to decrease development costs and cycle time by reducing the time and effort required to build these reused artifacts. Many researchers, e.g. [18] have also suggested the improvement of software quality can be attained by reusing high quality software artifacts. The aim of this part in the survey is to uncover how reuse is practically applied in WebSys development.

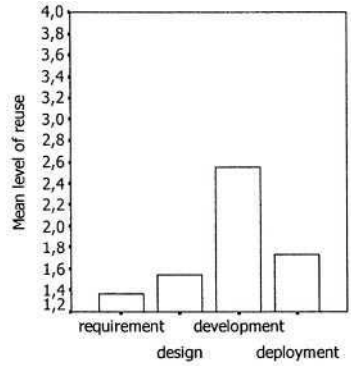


Fig. 8. Level of reuse in WebSys development process

Respondents were asked to give the level they reuse software artifacts in the different phases of the development process. In Figure 8 it is clear that the overall level of reuse is quite low. Among four major phases, development phase has the highest value 2.5, which is still lower than 3.

At first glance this result is somewhat surprising. In our opinion there should be a wide adoption of reuse due to immediacy of WebSys, as discussed in the introduction. However, as described in [16], the introduction and implementation of a reuse

program in a software organization is never a trivial task; both technical factors such as repositories, supportive tools, and non-technical factor such as process, human involvement are crucial for a successful reuse program. This may explain why reuse is not widely applied in WebSys development as many of preconditions are not in the place. Organizations developing WebSys have to spend time and effort to introduce reuse program before gaining actual time saving and improved quality.

4.3 Use of Engineering Methods for Reliability and Robustness

As stated in the introduction, reliability and robustness are important quality requirements that need to be engineered in the WebSys development process. As reliability has been long discussed and studied in the software community, many best current practice (BCP) exist for doing software reliability engineering [14], some more or some less strongly associated with robustness. This section discusses the level of use of such BCP in WebSys development.

Table 1. Investigated engineering methods for reliability and robustness

E1	Requirement validation	E10	Robustness patterns
E2	Failure definition and classification	E11	Trade-off analysis
E3	Specification review	E12	Reliability measurement
E4	Specification completeness and negative properties	E13	Robustness measurement
E5	Good modeling techniques	E14	Code review
E6	Component reuse	E15	Formal testing
E7	Input control	E16	Reliability growth modeling
E8	State control	E17	Registration of failure
E9	Exception handling	E18	Redundancy design

Engineering methods included in the survey. Table 1 lists all the investigated methods. Some of them are associated with both reliability and robustness, while others are more biased towards one of them.

The respondents were asked to rate their use of these eighteen engineering methods in their WebSys development practice on a scale from 1 (not at all) to 5 (very well). They are considered to be high-level users of a specific method if they rate their use of the method higher than 3. If this is the case, respondents were encouraged to comment on in which phase of the development process they employ these methods. These comments are used as inputs to the qualitative analysis. We haven't investigated time and efforts spent in applying these methods in this survey in order to avoid overloading the respondents. The main interest here lies in discovering critical areas (engineering methods) that we can work further with.

F8: The overall level of use of engineering methods and techniques for reliability and robustness is not high in the WebSys development process.

This fact is illustrated in Figure 9, where the percentage of high-level users of each method is listed. Of all the 18 engineering tasks, only two of them have a percentage of high-level users above 50%. They are E1 Interactive requirement validation and E15 formal testing.

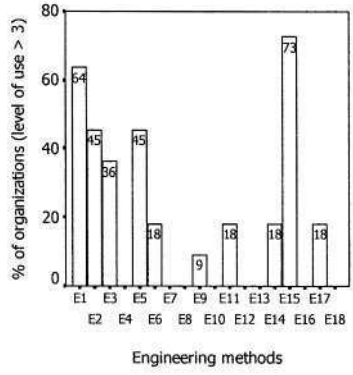


Fig. 9. Level of use of 18 engineering methods for reliability and robustness

F9: Compared to reliability, robustness is an even less addressed quality requirement in WebSys development.

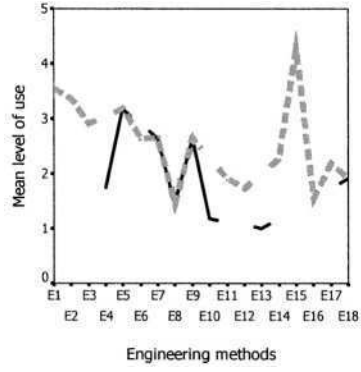


Fig. 10. Level of use of engineering methods for reliability versus robustness

As argued in the introduction, robustness is an important quality attribute for WebSys. However, our analysis demonstrated that it is not seriously treated in Web-Sys development. Firstly, in Figure 10 it is showed that the overall level of use of robustness related methods is even lower than reliability related methods. Secondly, among eight methods associated with robustness, five of them are also associated with reliability and three of them are biased towards robustness. The descriptive statistics in Figure 11 shows that these three methods E4, E10, and E13 actually have the lowest mean value for level of use among the 18 engineering methods.

The result from the survey is consistent with some prior works done about software robustness. An example is given in [17], where studies find that most developers have an incomplete understanding of how to build software systems with robust exception handling, or even the importance of good design with respect to handling errors and exceptional conditions. Content analysis on the comments provided by the respondents shows similar information in this survey. All the respondents who rated E7 input control, E8 state control and E9 exception handling above level 3 have reported that they employ these methods in the development phase (implementation and testing) rather than in the design phase.

	N	Minimum	Maximum	Mean	Std. Deviation
E1	11	2	5	3,55	1,128
E2	11	1	5	3,36	1,362
E3	11	1	5	2,91	1,446
E4	11	1	3	1,73	,786
E5	11	1	5	3,18	1,537
E6	11	1	4	2,64	,924
E7	11	2	3	2,64	,505
E8	11	1	3	1,45	,688
E9	11	2	4	2,64	,674
E10	11	1	2	1,18	,405
E11	11	1	5	1,91	1,375
E12	11	1	3	1,73	,647
E13	11	1	1	1,00	,000
E14	11	1	4	2,27	1,104
E15	11	2	5	4,27	1,104
E16	11	1	3	1,55	,820
E17	11	1	5	2,18	1,328
E18	11	1	3	1,91	,831
Valid N (listwise)	11				

Fig. 11. Descriptive statistics for level of use of reliability and robustness engineering methods

F10: Little effort is spent on reliability and robustness during design of WebSys.

Figure 12 shows that the overall level of use of methods employed in the design phase is quite low. Only one of them E5 has a mean value over 3. In addition, it is supposed that some methods such as E6, E7, E8, E9 should be, more or less, considered during design, but surprisingly, they are not addressed at all. All the respondents who rated them above level 3 have said that they use these methods during implementation and testing.

The absence of reliability and robustness considerations can lead to poor quality of the final products, and at the same time, delayed delivery time as the late detection of defects is well known to be extremely expensive [15]. Addressing reliability and robustness attributes in the design phase includes the following two aspects:

- 1. Introduce as few as possible design faults during design of WebSys
- 2. Detecting and removing as much as possible of the defects before entering the next phase

New methods and techniques invented and validated in these fields are critical to the development of reliable and robust WebSys with reduced time-to-market.

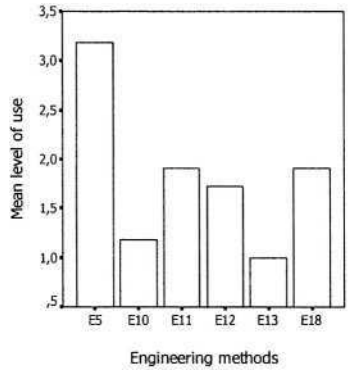


Fig. 12. Mean level of use of engineering methods during design

5 Summary and Future Research

Eleven organizations developing WebSys have been investigated in order to provide a picture of the status in current WebSys development, and to identify key areas in reducing time-to-market and improving reliability and robustness. Data has been collected through questionnaires and interviews. All the data collected are subjective, reflecting what the respondents perceive. Exploratory data analysis is employed to discover patterns underlying data through successive iterations in the analysis process.

Ten main findings are presented and discussed in three groups: features of the WebSys development projects, results related to time-to-market and use of engineering methods for reliability and robustness. We found that investigated organizations involve in development of different kinds of WebSys, with service-oriented applications developed most often. A myriad of Web related technology is mixed in WebSys development; Object Web and Web services represent the main future interests. As most of us believe, WebSys projects tend to be short and small. Time pressure becomes a dominating industrial driver.

In the survey, there is great agreement on that the development phase (implementation and testing) is the most time-consuming and the requirements phase is the least time-consuming. We argue that there are two ways to solve this bottleneck: one is through better method and tool supports in the development phase; and the other is to follow the best practice in software engineering, where it is said that additional investment in requirements and design will result in total time saving.

We found that iterative approach is popular in the investigated organizations, while the development process is still to a large extent ad hoc. Process planning issues such as scheduling, work estimation and milestones are well addressed and used at a high level, reflecting on the one hand a well-organized organizational culture and on the other hand the fact of market time pressure.

We found that the overall level of reuse is quite low in the whole WebSys development process. We argue that difficulties in introducing and implementing a systematic reuse program are an important explanation for such a low level of reuse. These difficulties are not analyzed and revealed by this survey, but it does offer an interesting issue for the future research.

According to the results of the survey, the overall level of use of engineering methods for reliability and robustness is not high, and robustness is even less addressed. In addition, we found that developers spend little effort on reliability and robustness during design. This naturally raises a question of how this inadequate handling of reliability and robustness will affect the quality of final products and total development time. This question can not be answered on the basis of the results of the present survey, but points out a critical area for future research. Other interesting area in future research is to explore the understanding of these methods and barriers to their use.

Acknowledgement. This paper is a part of the Web-based systems – reliability versus time-to-market (WebSys) project. The WebSys project is sponsored by the Norwegian Research Council.

References

1. Atzeni, P., Mecca, G., Merialdo, P., Crescenzi, V.: The Araneus Guide to Web-Site Development – Araneus Project Working Report (1999)
2. Berander, P., Wohlin, C.: Identification of Key Factors in Software Process Management – A Case Study. Proc. International Symposium on Empirical Software Engineering. ACM/IEEE CS Press (2003) 316-325
3. Bratthall, L., Runeson, P., Adelsward, K., Eriksson, W.: A Survey of Lead-time Challenges in the Development and Evolution of Distributed Real-time Systems. Information and Software Technology, Vol. 42 (2000) 947-958
4. Briand, L.C., Differding, C.M., Rombach, H.D.: Practical Guidelines for Measurement-based Process Improvement. Software Process Improvement and Practice, Vol. 2(4) (1996) 253-280
5. Conradi, R., Dybå, T.: An Empirical Study on the Utility of Formal Routines to Transfer Knowledge and Experience. Proc. European Software Engineering Conference. ACM/IEEE CS Press (2001) 268-276
6. Cooper, D.R., Schindler, P.S.: Business Research Methods. 8th edn. McGraw-Hill Higher Education (2003)
7. Datar, S., Jordan, C., Kekre, S., Rajiv, S., Srinivasan, K.: New Product Development Structures and Time To Market. Management Science, Vol. 43(3) (1997)
8. Demarco, T., Lister, T.: Peopleware – Productive projects and Teams. Dorset House Publishing Co, New York (1999)
9. Greer, D., Bustard, D., Sunazuka, T.: Effecting and Measuring Risk Reduction in Software Development. NEC Journal of Research and Development, Vol. 40(3) (1999)
10. IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990. Corrected edition (1991)

11. Keepence, B., Mannion, M.: Using Patterns to Model Variability in Product Families. IEEE Software, Vol. 16(4) (1999) 102-108
12. Kenneth, S.N.: Appling Cross-Functional Evolutionary Methodologies to Web Development. Lecture Notes in Computer Science, Vol. 2016. Springer-Verlag, Berlin Heidelberg New York (2001) 49-57
13. Lim, W.C.: Effect of Reuse on Quality, Productivity and Economics. IEEE Software, Vol. 11(6) (1994) 23-30
14. Lyu, M.R.: Hankbook of Software Reliability Engineering. McGraw-Hill (1996)
15. Mansurov, N.N., Probert, R.L.: Improving Time-To-Market Using Tools and Techniques. Computer Networks and ISDN (2000)
16. Maurizio, M., Ezran, M., Tully, C.: Success and Failure Factors in Software Reuse. IEEE Transactions on Software Engineering, Vol. 28(4) (2002) 340-357
17. Maxion, R.A.: Improving Software Robustness with Dependability Cases. Twenty-Eighth Annual International Symposium on Fault-Tolerance Computing (1998)
18. Melo, W.L., Briand, L.C., Basili, V.R.: Measuring the Impact of Reuse on Quality and Productivity in Object-Oriented Systems. Technical Report CS-TR-3395, University of Maryland, Dep. of Computer Science (1995)
19. Murugesan, S., Deshpande, Y., Hansen, S., Ginige, A.: Web Engineering: A New Discipline for Development of Web-based Systems. Lecture Notes in Computer Science, Vol. 2016. Springer-Verlag, Berlin Heidelberg New York (2001) 3-13
20. Olsina, L., Lafuente, G., Rossi, G.: Specifying Quality Characteristics and Attributes for Websites. Lecture Notes in Computer Science, Vol. 2016. Springer-Verlag, Berlin Heidelberg New York (2001) 266-278
21. Porter, M.E.: Competitive Strategy – Techniques for analyzing Industries and Competitors. The Free Press, New York (1980)
22. Pressman, R.S.: Software Engineering: A Practitioner's Approach. 5th edn. McGraw-Hill, England (2000)
23. Pressman, R.S.: What a Tangled Web We Weave. IEEE Software, Vol. 17(1) (2000)
24. Schneider, G., Winters, J.P.: Appling Use Cases, A Practical Guide. Addison Wesley, Object Technology Series (1998)
25. Seaman, C.B.: Qualitative Methods in Empirical Studies of Software Engineering. IEEE Transactions on Software Engineering, Vol. 25(4) (1999) 557-572
26. Solingen, R.V., Berghout, E.: The Goal/Question/Metrics Method: A Practical Guide for Quality Improvement and Software Development. McGraw-Hill (1998)
27. Tukey, J.W.: Exploratory Data Analysis. Addison-Wesley (1997)
28. Wheelwrigt, S.C., Clark, K.B.: Leading Product Development – The Senior Manager's Guide to Creating and Shaping the Enterprise. The Free Press, New York (1995)
29. Zamir, S.: Handbook of Object Technology. CRC Press, Boca Raton (1999)

Achieving CMMI Level 2 with Enhanced Extreme Programming Approach

Tuomo Kähkönen[1] and Pekka Abrahamsson[2]

[1] Nokia Research Center
P.O. Box 407, FIN-00045 NOKIA GROUP, Finland
tuomo.kahkonen@nokia.com
[2] VTT Technical Research Centre of Finland
P.O. Box 1100, FIN-90571 Oulu, FINLAND
Pekka.Abrahamsson@vtt.fi

Abstract. The relationship between agile methods and Software Engineering Institute's CMM approach is often debated. Some authors argue that the approaches are compatible, while others have criticized the application of agile methods from the CMM perspective. Only few CMM based assessments have been performed on projects using agile approaches. This paper explores an empirical case where a project using Extreme Programming (XP) based approach was assessed using the CMMI framework. The results provide empirical evidence pointing out that it is possible to achieve maturity level 2 with approach based on XP. Yet, the results confirm that XP, as it is defined, is not sufficient. This study demonstrates that it is possible to use the CMMI for assessing and improving agile processes. However, the analysis reveals that assessing an agile organization requires more interpretations than normally would be the case. It is further concluded that the CMMI model does not always support interpretations in an agile context.

1 Introduction

Agile software development approaches have generated a lot of interest in the field of software engineering in the last few years. A number of studies have shown that agile solutions are a viable option for many software companies producing software in a volatile business environment. Volatility has been contrasted with the stability. In fact, one of the more interesting debates in software engineering community is concerned with two apparently very different approaches for software process improvement: CMM[1] promoted by Software Engineering Institute (SEI) [1] and Extreme Programming developed by Beck [2]. CMM is often seen as the arch-type of traditional SW development and contradicted with agile development practices. Many authors have suggested that organizations should develop methods that combine agile and traditional elements [3-10]. Some authors have also argued that in principle the

[1] CMM and Capability Maturity Model are registered in the US Patent and Trademark Office. CMMI and SCAMPI are service marks of Carnegie Mellon University. Term CMM is used in this article to include both Software Capability Maturity Model (SW-CMM) and Capability Maturity Model Integration (CMMI).

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 378–392, 2004.

CMM and agile approaches are compatible [3, 8, 10-14]. However, it has proven to be difficult to combine these approaches in practice [3, 10] and many important limitations in the existing agile methodologies, like XP, have been pointed out from the CMM perspective [7, 8, 13-20]. Yet, only few if any studies have performed a CMM-based assessment on an agile project. For this reason currently it is not well understood how to build methods that that combines these two approaches in practice.

The primary purpose of this paper is to increase understanding about the relationship between XP and CMMI. This paper reports results from a study that analyzed using CMMI as the frame of reference a software development project using an enhanced XP based process. The project assessed was rated at a CMMI maturity level 2. The results of this study confirm the theoretical comparisons between XP and CMM [8, 13, 16, 21, 22] claiming that XP does not fulfill CMM requirements. However, the results also show that it is possible to construct a process that fulfills CMMI requirements by adding practices to XP. These additions are outlined in detail so that other organizations can benefit from the results. It is claimed that the results are applicable to certain extent to other agile methods as well. Finally, the challenges of using CMMI for assessing and improving agile processes are discussed.

This paper is organized as follows. Next section presents the research objectives and outlines the objectives of the CMMI. The third section introduces the empirical case project and the fourth section presents the key findings of the study. Section five discusses the implication of these findings and relates them to the literature. The paper in concluded with final remarks.

2 Research Objectives and Methods

This paper aims at increasing understanding about the relationship between XP and CMMI. This is achieved by performing an actual CMMI assessment on a project using an agile method. First research objective is to demonstrate how an enhanced XP based process can satisfy the criteria set by CMMI. This is done comparing the actual practices performed in the project to CMMI requirements practice by practice. The second research objective is outline the challenges of using CMMI for assessing and improving agile processes. This is performed by using the insight and experiences gained from the assessment process.

The research approach used in this study is different from other similar studies [8, 13, 16, 21, 22] in two important aspects: First, the evaluation of the CMMI requirements is done within the context of a real life project. Second, the analysis is taken down to the more detailed practice level from the goal level. This more detailed approach is expected to deliver more reliable results.

A real life project assessment was selected over desk exercise because CMMI highlights that when it is used to enhance existing processes, professional judgment must be used to interpret the CMMI practices. The practices must be interpreted using an in-depth knowledge of CMMI, the discipline, the organization, the business environment, and the specific circumstances involved. [1, 23] Thus it is not possible to obtain universally applicable results portraying that "a method X is or is not CMMI compliant" but the method adequacy and institutionalization must be assessed case by case.

The analysis in this paper covers specific goals and practices of the CMMI maturity level 2 process areas excluding Supplier Agreement Management. The generic goals are not included in the analysis because the research focus is in the method rather than its implementation and institutionalization in the case organization.

CMMI was selected to be the assessment framework firstly because it offers major improvements to SW-CMM regarding iterative and risk driven development[7, 24], and thus it can be supposed to be more aligned with the agile development ideas that SW-CMM. And secondly, because SW-CMM is not developed by the SEI any longer and CMMI is known to replace it in few years time. The next section introduces CMMI briefly.

2.1 CMMI

Capability Maturity Models in general contain the essential elements of effective processes for one or more disciplines. These elements are based on the concepts developed by Crosby [25], Deming [26], Juran and [27] Humphrey [28]. CMMI integrates systems engineering, software engineering, and integrated product and process development in one model. The purpose of CMMI is to provide guidance for improving organization's processes and enables the organization to better manage the development, acquisition, and maintenance of products or services. [1]

Maturity level is a central concept in CMM. It is *a priori* defined evolutionary plateau of process improvement. Each maturity level stabilizes a part of the organization's processes. At level 2, i.e. managed, the organization has ensured that its processes are planned, documented performed monitored, and controlled at project level [1].

Each maturity level in the CMMI contains several process areas that have two types of goals: specific and generic. Specific goals apply to one process area and address the unique characteristics that describe what must be implemented in order to satisfy the purpose of the process area. Generic goals apply to all process areas and address the implementation and institutionalization of the process area. [1]

Specific and generic goals are the only required components of the CMMI. In addition there are expected components, specific and generic practices, describing what organization will typically implement to achieve the goals. The actual practices in an organization assessed must be interpreted using an in-depth knowledge of CMMI, the discipline, the organization, the business environment, and the specific circumstances involved. The organization does not need to implement the practices as described in CMMI model. It is acceptable to implement an alternative practice that fulfills the same purpose.[1, 23] The context must be taken in account when evaluating adequacy of proposed alternative practice. The same practice may be adequate in one situation (e.g. small project) and the required goals are fully achieved but in different situation (e.g. large project) the same practice may prove to be inadequate [29].

2.2 Research Setting

The assessed project was VTT's eXpert project [30]. The assessment was performed after the project had finished, system testing was completed and the software product

was in actual use. The scope of assessment was set *a priori* at CMMI level 2. The Supplier Agreement Management key process area (KPA) was excluded, because the project did not have any subcontracting.

Used assessment method (Nokia CMMI-Based Process Assessment, CMMI-B) is based on SCAMPI (Standard CMMI Appraisal Method for Process Improvement) [31] and is supposed to be ARC (Appraisal Requirements for CMMI) Class B compliant [32]. Although the level 2 rating achieved in this assessment is not official SEI rating as only Class A assessment can produce ratings for benchmarking [32], the CMMI-B assessment results are expected to be fairly close to the results of Class A compliant appraisal.

Assessment team included three persons: The first author as the assessment team leader with experience from several CMMI assessments and the second author and a third person as assessment team members. The second author is also trained assessor. The assessment team had available all the material produced by the team including story cards and flip charts. The assessment team familiarized with the material before the interviews and had it available during the rating session.

The assessment team interviewed the project manager, two developers, customer, system test specialist and the business manager. Assessment team members took notes from the interviews to process specific templates. The interviews were also tape recorded (but not transcribed) for research and verification purposes. The interviewees were notified that the results could be published as a research paper.

Ratings were done immediately after assessment and the objective evidence found was written down in an Excel sheet practice by practice.

3 Case Description: The eXpert Project

The eXpert project's implementation phase was carried out 3.2.2003 – 11.04.2003 in VTT Electronics Oulu, Finland. The purpose of the project was to develop an intranet application for managing research information based on its logical structure. A team of four developers was acquired from the University of Oulu to implement the project. Table 1 describes the roles of different people/groups involved in the eXpert project. The project had a fixed time contract: 8 weeks in calendar time and 1000 hours effort. The time and effort were thus fixed. The flexibility was reserved for the delivered functionality. The project used an approach suggested by Lippert [33]. XP practices [for more detailed description of the XP practices, see 2, 34, 35]) were extended with some additional practices:

- Before the project started to develop code, a separate planning team had worked with the initiation of the project. This work continued during the project in Steering Group meetings. Steering group had three meetings: at the beginning of the project, in the middle and after the project.
- A two-day XP workshop was held for the project team before the start of the project. The project team was walked through the XP practices and the tools to be used with the manager and agreed on the practices to be followed in the project.
- One project mission was to collect data from XP process for research purposes, so there were enhanced data collection mechanisms in place. The collected data included time, defect and data [36]

- In configuration management (CM) area some additional practices were introduced. These included written CM plan, light CM audit procedure at the end of each iteration and a set of explicit change request/error sheets.
- Project had daily wrap-up meetings to discuss progress, plans and problems.
- XP supposes that project reflects at times how it is doing and tries to find ways to improve [2]. The eXpert project took this further by organizing a semi-formal post-mortem workshops after every iteration according to the guidelines proposed by Dingsøyr and Hanssen [37]
- The project did some additional documentation not typically done in XP projects. The planning team elaborated an implementation plan and the project manager authored a written project plan. Project manager also maintained a spreadsheet called Task Book that contained release plan and planned and actual effort spent for each task. Minutes were taken from the steering group meetings. There was also a CM plan, CM audit checklist, change request log and an error log to help CM activities. Architecture, database and user interface description documents were written during the last iteration and a system test report was made from the system testing. The results of the post-mortem sessions were recorded and displayed on the wall. Later these notes were transformed to a document.
- In the end of each iteration the project team had a pre-release testing session. After that the product was released to volunteered end-users for testing and getting feedback. External specialist designed the system testing procedure for the final product. First initial systematized testing was done after iteration 3 and second more comprehensive testing was done at the end of the project.

Table 1. Roles in the eXpert project

Role	Purpose
Planning Team	Worked prior to the project; defined projects' scope etc.
Steering Group	Members were from the university and the research institute including the project team.
Business Manager	Started the project and owned the results. Was responsible for providing all needed facilities and resources for the project.
Customer	Key user of the system under construction. Had the best understanding what the system should do.
Project Manager	Was team member responsible for project management.
CM Specialist	Team member who handled CM issues.
Metrics responsible	Team member, ensured that the metrics required were collected.
Research responsible	Team member, ensured that the research question the team undertook to solve was addressed.
System Test Specialist	Planned an executed system tests.
End User	17 volunteered end users tested each release and reported found bugs and improvement ideas to the customer.

4 Assessment Results

The eXpert project was rated at CMMI level 2. All specific goals (SG) were fully satisfied although there were minor findings and interpretation issues at specific practice (SP) level. This chapter presents the rationale for the rating of the specific goals practices by practice in the tables 2-14. Each table presents the specific goal on the top, practice definition on the left, and the rationale for rating on the right. The goal and practice descriptions are from [1]. Based on the rating rationale presented in the tables, the reader can verify the accuracy of the proposed CMMI level 2 rating for the project.

Table 2. Requirements Management SG1

SG 1: Requirements are managed and inconsistencies with project plans and work products are identified.	
SP1.1: Develop an understanding with the requirements providers on the meaning of the requirements	Pre-project planning team defined the project goals and the initial set of the user stories. During the project the customer had the responsibility to provide the requirements. Planning Game was used to communicate the requirements to the team.
SP1.2: Obtain commitment to the requirements from the project participants	This was achieved in planning meetings.
SP1.3: Manage changes to the requirements as they evolve during the project	The customer who worked as a part of the team had an active role in managing changes to the requirements. He maintained change request Excel sheet continuously. Changes were communicated to the team in planning game where also the impacts of the change were analyzed. When a story changed, old story card was archived.
SP1.4: Maintain bidirectional traceability among the requirements and the project plans and work products	Continuously maintained Task Book stated which user stories/tasks are implemented in each release. Releases were uniquely identifiable based on a baseline in the CM system. This enabled bi-directional user story/task – release traceability. There was also manual traceability between user stories and unit tests.
SP1.5: Identify inconsistencies between the project plans and work products and the requirements	Team and the customer together checked the consistency of the plans and the requirements in the planning game. CM audit checked after each iteration that all planned/reported user stories and tasks had been implemented.

Table 3. Requirements Management SG2

SG2: Estimates of project planning parameters are established and maintained	
SP2.1: Establish a top-level work breakdown structure (WBS) to estimate the scope of the project	This practice was not applicable because the customer was not able to define requirements exactly in the beginning of the project. The project used an alternative practice. It had fixed effort and variable scope. Planning game was used to define the scope of the work for each iteration.
SP2.2: Establish and maintain estimates of the attributes of the work products and tasks	Tasks were estimated only for the next iteration. The short estimation cycle with frequent feedback helped to make accurate estimates although estimates based on expert opinions.
SP2.3: Define the project life-cycle phases upon which to scope the planning effort	Project had incremental fixed release schedule set by planning team. Every iteration formed one phase in the project.
SP2.4: Estimate the project effort and cost for the work products and tasks based on estimation rationale	The total effort of the project was fixed. How the effort was used was estimated according to XP procedures in planning meetings. As the project had a fixed effort, fixed price contract, the planning variable was the scope of the work.

Table 4. Project Planning SG1

SG1: A project plan is established and maintained as the basis for managing the project	
SP1.1: Establish and maintain the project's budget and schedule	The project had a fixed budged. The schedule was established in planning meetings and documented in Task Book. Tasks were used to determine what can be accomplished within an iteration.
SP1.2: Identify and analyze project risks	Project manager identified project risks that were documented in the project plan and discussed in steering group. The actions originated from the risks were discussed in post mortem meetings.
SP1.3: Plan for the management of project data	CM plan identified configuration items and how to manage them. Team agreed practices needed to manage other data (e.g. story cards).
SP1.4: Plan for necessary resources to perform the project	Planning team did rough estimate of persons needed and the project duration. Project resources were documented in the project plan. Project had an opportunity to use various specialists in VTT as needed. The procedure for this was in project plan.
Plan for knowledge and skills needed to perform the project.	This planning was done by the planning team and documented in the implementation plan.

SP1.5: Plan the involvement of identified stakeholders	Planning team had planned this in the implementation plan.
SP1.6: Establish and maintain the overall project plan content	A written project plan existed according to standard VTT project plan guidelines. Project plan was updated after every iteration. Schedules was maintained in Task Book.

Table 5. Project planning SG2

SG2: Commitments to the project plan are established and maintained.	
SP2.1: Review all plans that affect the project to understand project commitments	Project was independent from others, so it did not need to review any plan.
SP2.2: Reconcile the project plan to reflect available and estimated resources	Reconciling was done in planning meetings. The scope of the work was adjusted to mach the resources.
SP2.3: Obtain commitment from relevant stakeholders responsible for performing and supporting plan execution	Commitments were obtained in steering group review.

Table 6. Project monitoring and control SG1

SG1: Actual performance and progress of the project are monitored against the project plan.	
SP1.1: Monitor the actual values of the project planning parameters against the project plan	Actuals (user Stories/tasks/hours) were collected in the Task Book Excel sheet alongside with the estimates.
SP1.2: Monitor commitments against those identified in the project plan	Project commitments were monitored and adjusted in planning meetings. Problems endangering those commitments were handled in wrap-up meetings.
SP1.3: Monitor risks against those identified in the project plan	Planned risk mitigation activities were performed. The risks were reviewed in SG meetings.
SP1.4: Monitor the management of project data against the project plan	CM audits checked the configuration items after every iteration. SG reviewed that documents are done.
SP1.5: Monitor stakeholder involvement against the project plan	Management monitored stakeholder involvement as a part of their work. The number of stakeholders was limited and contacts with them frequent.

SP1.6: Periodically review the project's progress, performance, and issues	Green bar on the wall indicated task progress continuously. Progress and issues were discussed in wrap-up and planning meetings. Every release made the progress visible. Steering group reviewed the progress in their meetings.
SP1.7: Review the accomplishments and results of the project at selected project milestones	Each release can be considered as a milestone in this case. Review was done in CM audit, post mortem session and in the next planning meeting.

Table 7. Project monitoring and control SG2

SG2: Corrective actions are managed to closure when the project's performance or results deviate significantly from the plan.	
SP2.1: Collect and analyze the issues and determine the corrective actions necessary to address the issues	Issues were identified in daily wrap-up meetings and in post mortem sessions. In addition manager visited the project team daily asking for possible issues. Management actions were taken as needed.
SP2.2: Take corrective action on identified issues	Management actions were done, post mortem sessions resulted actions.
SP2.3: Manage corrective actions to closure	If the problem was not solved, it was discussed again in wrap-up meeting or post mortem session. Notes from the previous post mortem sessions were on the wall.

Table 8. Measurement and analysis SG1

SG1: Measurement objectives and activities are aligned with identified information needs and objectives	
SP1.1: Establish and maintain measurement objectives that are derived from identified information needs and objectives.	The planning team and manager defined initially the measurement objectives. One measurement objective was to collect research data on XP. The measurement objectives were communicated verbally to the project manager.
SP1.2: Specify measures to address the measurement objectives	Planning team defined an initial set of measures to be collected. The project manager defined some additional metrics that he needed (Task Book).
SP1.3: Specify how measurement data will be obtained and stored	Project Manager had explicit responsibility to define data collection. Paper-pen method was used to collect the data daily. Some data was obtained from CVS and the Task Book.
SP1.4: Specify how measurement data will be analyzed and reported	Operative metrics analysis and reporting was built in the Task Book. Research data was analyzed separately from the project.

Table 9. Measurement and analysis SG2

SG2: Measurement results that address identified information needs and objectives are provided	
SP2.1: Obtain specified measurement data	Based on the measurement records, the data was obtained very accurately.
SP2.2: Analyze and interpret measurement data	Operative analysis was done in Excel based on Task Book data. Mostly these were simple graphs.
SP2.3: Manage and store measurement data, measurement specifications, and analysis results	Operative metrics and analysis were stored on file server. This was because also the manager needed to access it but he did not have access to CVS repository.
SP2.4: Report results of measurement and analysis activities to all relevant stakeholders	Manager had access to measurement results all the time. Measurement results were presented in post mortem sessions and steering group meetings.

Table 10. Process and product quality assurance SG1

SG1: Adherence of the performed process and associated work products and services to applicable process descriptions, standards, and procedures is objectively evaluated.	
SP1.1: Objectively evaluate the designated performed processes against the applicable process descriptions, standards, and procedures	On-site customer followed that team follows the agreed processes. In post mortem session manager and customer evaluated with the team the process and identified possible deviations and improvement needs. Manager followed the metrics in order to identify deviations from the process.
SP1.2: Objectively evaluate the designated work products and services against the applicable process descriptions, standards, and procedures	Manager and steering group reviewed the documents. They checked e.g. that correct templates have been used and they contain all applicable items (e.g. VTT project plan guidelines). Customer checked using samples in CM audits that agreed coding standards had been used.

Table 11. Process and product quality assurance SG2

SG2: Noncompliance issues are objectively tracked and communicated, and resolution is ensured	
SP2.1: Communicate quality issues and ensure resolution of noncompliance issues with the staff and managers	Manager and customer communicated and handled the identified quality issues with the team in post mortem sessions and daily meetings. Manager tracked quality issues. The issue was brought up again if no improvement happened. Quality issues were discussed also in steering group meetings.

SP2.2: Establish and maintain records of the quality assurance activities	There were two types of records: configuration audit reports and post mortem session notes on the wall.

Table 12. Configuration management SG1

SG1: Baselines of identified work products are established.	
SP1.1: Identify the configuration items, components, and related work products that will be placed under configuration management	Configuration items and their storage were stated explicitly in CM plan. The documents were later stored in file server instead of CVS. A naming convention for different version of the documents was agreed and used. The customer had Change Request Log on his workstation.
SP1.2: Establish and maintain a configuration management and change management system for controlling work products	Project used CVS to store source code. Change management system was manual. Customer maintained a Change Request Log where all changes and their status were tracked.
SP1.3: Create or release baselines for internal use and for delivery to the customer	Baselines were created for every release.

Table 13. Configuration management SG2

SG2: Changes to the work products under configuration management are tracked and controlled.	
SP2.1: Track change requests for the configuration items	All requirement change management was done through the customer who had a Change Request Log that he used for himself to track the changes. Error log was used to manage error corrections for release candidates/releases. CM audit ensured that agreed changes were actually implemented.
SP2.2: Control changes to the configuration items	Release baselines in CVS were frozen. Team members were allowed to make a new version of any file or document any time. It was possible to change old versions of documents that located on file server but this risk was acknowledged.

Table 14. Configuration management SG3

SG3: Integrity of baselines is established and maintained	
SP3.1: Establish and maintain records describing configuration items	Change Request list, Error list, automatic change history in CVS, manually updated change history in document.
SP3.2: Perform configuration audits to maintain integrity of the configuration baselines	Customer performed a light but sufficient configuration audits after every release. He had a checklist that was filled as he went through the items. The filled checklist was archived.

5 Discussion

There are two important factors that helped the eXpert project to achieve CMMI maturity level 2. Firstly the project used additional practices that are not part of normal XP process and did some additional documentation. If these practices were not been in use and the documents would not have been there, several goals would not have been rated fully satisfied. Secondly, the size of the project was small and the co-operation with the team members so intense that many light practices were sufficient for the particular situation. If the project had been larger or the communication within the team had been less intense, some of the practices would have been inadequate for the situation.

These results show that although there are several weaknesses in XP from SW-CMM/CMMI perspective, it is possible to achieve CMMI level 2 using a process that is based on XP and is extended with additional practices. Attention should be paid on the generalizability of this result. It can't be concluded that the use of eXpert process automatically lead organization to CMMI level 2 maturity. This is because the adequacy of the practices depends on the context where they are used. For larger projects, or projects operating in otherwise more challenging environments, some additional/alternative practices may be needed.

It was found out that CMMI can be used to assess processes that are combining agile and traditional elements. However, based on the assessor experiences, this is a very challenging task. Although the assessor had experience both from CMMI assessments and XP, interpreting mandatory and expected CMMI elements in the XP context was not always easy or straightforward. Also, because the informative information in CMMI was of little help, the role of assessor interpretations was emphasized.

Many of the interpretation issues in this assessment were culminating on knowledge management. CMMI and agile methods manage knowledge created during the projects differently [8]. CMMI places emphasis on explicit, documented knowledge [7, 24]. Many practices that explicitly create documents are expected and also the definition of a work product in CMMI suggests that the work products must be concrete artifacts files or documents [1].

Research has shown that software development is highly knowledge-intensive work [e.g., 38, 39] and centric role of tacit (i.e. undocumented) knowledge in agile method has been highlighted [7, 40, 41]. The role of tacit knowledge is seen to be very important in knowledge creation process in general [42-44]. The tacit knowledge is manifested in different kinds of intangible artifacts and concrete artifacts like documents present only a tip of the iceberg of the whole knowledge [44].

The question the assessor has to answer is whether an alternative practice, that relies on tacit knowledge, and can be considered working and institutionalized, is acceptable or not. Currently when there are no common guidelines for interpreting adequacy of agile practices, different assessors may end up with different ratings.

Although professional judgment is an integral part of CMMI assessment, Turner [8] has suggested that there are two different CMMI schools regarding agility – a conservative, by-the-letter group and a liberal, concept-oriented group. This makes ratings more assessor-dependent thus hampering the reliability of the assessment results. This limitation is mitigated in this study by reporting the rationale for the

rating level achieved. To our believe, the CMMI vs. agile discussion will benefit from this type of approach

A great deal of attention has been paid for the different requirements management approaches in CMM and agile methods. The traditional approach of having complete, consistent, precise, testable and traceable requirements may work fine in a situation where the requirements are stable but when the change rate increases, the traditional approach encounters difficult update problems [6]. The adequacy of XP requirements management practices is questioned especially when applied to component-based software development or large organizations [17]. XP is not considered to offer a generic solution that fulfills CMM requirements for requirements management [15, 16, 18]. On the other hand Turner [8] claims that CMMI and agile methods are not conflicting in requirements management process area. The results of this study confirm this latter interpretation at least in projects that fall within the scope of the assessment reported.

6 Conclusions

Agile software development and CMMI have been seen as conflicting views to software development. To challenge this view, this paper reported results from a study where a project using enhanced XP approach was assessed using the CMMI framework. The results of this study confirm the theoretical comparisons between XP and CMM [8, 13, 16, 21, 22] claiming that XP, by the book, does not fulfill CMMI requirements. However, the results also show that it is possible to construct a process that fulfills CMMI requirements by adding additional practices to XP. Various authors have suggested that there is a need for methods that combine agile and more traditional elements [3-10]. The results of this study confirm that it is possible to construct such methodologies in practice.

In this particular case the organization used in addition to the XP practices many traditional project management practices to initiate and steer the project and provided needed training and mentoring for the project team. The project used light documentation practices, but more documentation was done than what XP suggests. Some lightweight versions of typical traditional practices were introduced in testing and configuration management. Project team improved their processes using post mortem sessions at the end of each iteration.

This study evidenced that CMMI is one possible framework that can be used as a helping tool when building methods that combine agile and more traditional elements. Especially CMMI can be used as tool for checking that all relevant aspects are covered in the method. However, CMMI should be applied cautiously because the interpretation of the CMMI requirements is not always straightforward in the context of agile practices. One important reason for the interpretation issues was found out to be different conception of tacit knowledge in CMMI and agile methods. Despite the observed challenges, CMMI can be a valuable tool when building up processes that combine agile and traditional elements for extending the current scope of applicability of the agile methods.

References

[1] *Capability Maturity Model® Integration (CMMI), Version 1.1:* Carnegie Mellon University, 2001.

[2] K. Beck, *Extreme Programming Explained: Embrace Change.* Upper Saddle River, NJ: Addison-Wesley, 2000.

[3] M. Paulk, "Agile Methodologies and Process Discipline," *CrossTalk The Journal of Defense Software Engineering,* pp. 15-18, 2002.

[4] M. Fowler, "Is Design Dead," in *Extreme Programming Examined,* G. Succi and M. Marchesi, Eds.: Addison-Wessley, 2001, pp. 3-18.

[5] B. Henderson-Sellers, "Agile or Rigorous OO Methodologies: Getting the Best of Both Worlds," *Cutter IT Journal,* vol. 15, pp. 25-33, 2002.

[6] B. Boehm, "Get Ready for Agile Methods, with Care," *IEEE Computer,* pp. 64-69, 2002.

[7] B. Boehm and R. Turner, *Balancing Agility and Discipline.* Boston: Addison-Wesley, 2003.

[8] R. Turner and A. Jain, "Agile Meets CMMI: Culture Clash or Common Cause?," presented at 2nd XP and 1st Agile Universe Conference, Chigago, IL, 2002.

[9] J. Highsmith, "What Is Agile Software Development?," *CrossTalk The Journal of Defense Software Engineering,* pp. 4-9, 2002.

[10] D. Reifer, "XP and the CMM," *IEEE Software,* pp. 14-15, 2003.

[11] R. Glass, "Agile Versus Traditional: Make Love, Not War!," *Cutter IT Journal,* vol. 14, pp. 12-18, 2001.

[12] H. Glazer, "Dispelling the Process Myth: Having a Process Does Not Mean Sacrificing Agility or Creativity," *CrossTalk The Journal of Defense Software Engineering,* pp. 27-30, 2001.

[13] J. Martinsson, "Maturing XP Through the CMM," presented at XP2003, Genova, Italy, 2003.

[14] C. Vriens, "Certifying for CMM Level 2 and ISO9001 with XP@Scrum," presented at Agile Development Conference, Salt Lake City, Utah, 2003.

[15] L. Wagner, "Extreme Requirements Engineering," *Cutter IT Journal,* vol. 14, pp. 34-38, 2001.

[16] M. Paulk, "Extreme Programming from a CMM Perspective," *IEEE Software,* pp. 19-26, 2001.

[17] P. Allen, "Light Methodologies and CBD," *Component Development Strategies,* vol. XI, pp. 1-16, 2001.

[18] J. Nawroski, M. Jainski, and B. Walter, "Extreme Programming Modified: Embrace Requirements Engineering Practices," presented at IEEE Joint International Conference on Requirements Engineering (RE'02), 2002.

[19] W. S. Humphrey, "Comments on eXtreme Programming,", http://computer.org/SEweb/dynabook/HumphreyCom.htm, 2000.

[20] J. R. Nawrocki, B. Walter, and A. Wojciechowski, "Comparison of CMM level 2 and eXtreme programming," presented at 7th European Conference on Software Quality, Helsinki, Finland, 2002.

[21] R. Jeffries, "Extreme Programming and Capability Maturity Model", http://www.xprogramming.com/xpmag/xp_and_cmm.htm, 2000.

[22] J. Martinsson, "Maturing Extreme Programming Through the CMM," in *Department of Computer Science.* Lund: Lund University, 2002.

[23] M. B. Chrissis, M. Konrad, and S. Shrun, *CMMI Guidelines for Process Integration and Product Improvement.* Boston: Addison-Wesley, 2003.

[24] W. Royce, "CMM vs. CMMI: From conventional to Modern Software Management," *Rational Edge,* 2002.

[25] P. Crosby, *Quality is Free.* New York: McGraw-Hill, 1979.

[26] W. Deming, *Out of the Crisis.* Cambridge, MA: MIT Center for Advanced Engineering, 1986.
[27] J. M. Juran, *Juran on Planning Quality.* New York: MacMillan, 1988.
[28] W. S. Humphrey, *Managing the Software Process.* Reading, M.A.: Addison-Wesley, 1989.
[29] M. Paulk, "Using the Software CMM in Small Organizations," presented at The Joint 1998 Proceedings of the Pacific Northwest Software Quality Conference and the Eighth International Conference on Software Quality, Portland, Oregon, 1998.
[30] P. Abrahamsson, "Extreme Programming: First Results from a Controlled Case Study," presented at 29th Euromicro Conference, Belek, Antalaya, Turkey, 2003.
[31] *Standard CMMI Appraisal Method for Process Improvement (SCAMPI), Version 1.1: Method Definition Document.* Pittsburg: Carnegie Mellon University, 2001.
[32] *Appraisal Requirements for CMMI (ARC) V1.1.* Pittsburg: Carnegie Mellon University, 2001.
[33] M. Lippert, P. Becker-Pechau, H. Breitling, J. Koch, A. Kornstädt, S. Roock, A. Schmolitzky, H. Wolf, and H. Züllighoven, "Developing Complex Projects Using XP with Extensions," *IEEE Computer,* pp. 67-73, 2003.
[34] K. Beck, "Embracing Change With Extreme Programming," *IEEE Computer,* vol. 32, pp. 70-77, 1999.
[35] R. Jeffries, A. Anderson, and C. Hendrickson, *Extreme Programming Installed.* Upper Saddle River, NJ: Addison-Wesley, 2001.
[36] W. S. Humphrey, *A discipline for software engineering.* Reading, Mass.: Addison Wesley, 1995.
[37] T. Dingsøyr and G. K. Hanssen, "Extending Agile Methods: Postmortem Reviews as Extended Feedback," presented at 4th International Workshop on Learning Software Organizations, Chicago, Illinois, USA, 2002.
[38] L. Prusak, *Knowledge in organizations.* Oxford: Butterworth-Heinemann, 1997.
[39] M. Robertson, C. Sørensen, and J. Swan, "Survival of the leanest: Intensive knowledge work and groupware adaptation," *Information Technology & People,* vol. 14, pp. 334-352, 2001.
[40] T. Kähkönen and P. Abrahamsson, "Digging into the Fundamentals of Extreme Programming - Building the Theoretical Base for Agile Methods," presented at Euromicro 2003, Belek, Antalaya, Turkey, 2003.
[41] A. Cockburn, "Agile Software Development Joins the "Would-Be" Crowd," *Cutter IT Journal,* vol. 15, pp. 6-12, 2002.
[42] M. Polyani, *The Tacit Dimension.* New York: Anchor, 1967.
[43] I. Nonaka and H. Takeuchi, *The Kowledge Creating Company: How Japanese Companies Create the Dynamics of Innovation.* New York: Oxford University Press, Inc., 1995.
[44] I. Tuomi, *Corporate Knowledge: Theory and Practice of Intelligent Organizations.* Helsinki: Metaxis, 1999.

Usability Assessment of an Extreme Programming Project: Close Co-operation with the Customer Does Not Equal to Good Usability

Timo Jokela[1] and Pekka Abrahamsson[2]

[1] P.O. Box 3000
FIN-90014 University of Oulu, Finland
timo.jokela@oulu.fi
[2] VTT Technical Research Centre of Finland
P.O. Box 1100
FIN-90571 Oulu, Finland
pekka.abrahamsson@vtt.fi

Abstract. Extreme programming (XP) is a development paradigm that is becoming widespread in the field of software engineering. Very few – if any – empirically validated research results have been reported on the relationship between usability engineering and XP. To understand to which extent XP guides to development of software that is usable, a usability process assessment was conducted on a controlled XP project. The analysis reveals that XP pays almost no attention on the usability of the software, apart from some actions that can be regarded as implicit usability evaluations. The analysis shows further that in XP the team transfers the responsibility of the product's usability to the customer. It is argued, however, that an XP team is in trouble when an enlightened customer sets value on usability requirements. These findings bear significant implications on research and practice. These implications are discussed and a promising avenue for solution is identified.

1 Introduction

Usability is a software quality characteristic, defined as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" [1]. Usability engineering – also called user-centered design and human-centered design - is a paradigm that addresses specifically on designing usable software. The key references to usability engineering are Jacob Nielsen's book [2] and ISO 13407 [3].

Agile software development methods and principles are a focus of growing interest in the field of software engineering. The best known of these methods is Extreme Programming (XP), an agile method developed by Beck [4].

Also the usability engineering community has recently shown an interest in agile software development and XP. For example, the recent Usability Professional Asso-

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 393–407, 2004.

ciation conference had a workshop on the UCD practices in agile methods [5]. However, very few – if any – empirically validated research results have been reported on the relationship between usability engineering and XP.

Agile development places emphasis on intensive communication between different stakeholders and provides rapid feedback based on working software: "We value individuals and interactions over process and tools – working software over comprehensive documentation" (http://www.agilemanifesto.org). The interesting question is whether this customer-centeredness would also mean good usability. The questions that this paper seeks to address are:

- To which extent XP guides to design software that is usable?
- Would the customer-centeredness of XP mean also user-centeredness?

These basic questions formed the starting point for our study. This study aims at filling the gap by providing scientifically grounded empirical data about the relationship between usability and XP.

In this study, a case project - an XP-based software development project - was analyzed from usability engineering viewpoint. The project was not just 'any' XP project but a professionally run, 'by the book' implementation of the XP process. The project's goal was to gather empirical data about the effectiveness of the XP process and the quality of the resulting product. For several reasons set out below, we decided that an analysis of this project should provide interesting and valuable information about the relationship between XP and usability engineering.

Firstly, the analysis would be based on an empirical case which closely followed the ideal XP process. Secondly, the project was organized and run by professionals well-versed on XP. Thirdly, the analysis of usability engineering was not explicitly considered before or during the project – the analysis was initiated only after the project was finished. In this way, the project was unbiased from the usability engineering viewpoint. Fourthly, the project was a clear success from the software engineering point of view. The project delivered a fully functional software system within the limits of the timeline and the resources. The quality of the system in terms of development-time and post-release defect-density was excellent. The development process was assessed at the CMMI level 2 [6] and the software code was deemed to have an acceptable level of quality. Both assessments were performed by independent teams of external experts. The customer was satisfied with the project's outcome.

The study was carried out by analysis of the XP development process, not by examining the usability of the end product. Why this choice? The rationale is as follows. Good usability of an end product can be ensured only by systematic usability engineering activities during the development cycle. A product can have some good usability features also without systematic usability engineering activities. In such a case, however, good usability is more or less an outcome of coincidence, which is based on the intuitions of the customer and the developers. Therefore, analyzing the end product had not provided new insight into the relation between XP and usability.

This paper is organized as follows. First, we introduce in brief the basics of XP and the XP case study setting. Then we provide an overview of usability capability as-

sessment methods, and justify our selection. Then results of the assessment of the case study are presented, and conclusions drawn. In end of the paper, we discuss the implications for practitioners and suggest directions for further research.

2 The XP Case Project

The XP process is characterized by short development cycles, incremental planning, evolutionary design, and its ability to respond to changing business needs. The method itself is built around an apparently easy-to-understand set of practices, which have been documented in the literature. These practices include the planning game, small releases, metaphor, simple design, testing (test-driven development), refactoring, pair programming, collective ownership, continuous integration, 40-hour work week (also known as sustainable pace) and on-site customer, fair rules and open workspace. In addition, spikes [7] are also often associated to the XP method's practices.

The XP method is designed to meet the needs of a small (i.e., less than 10 developers) skilled team working in a co-located office together with the customer developing software that is not safety-critical on an object-oriented technology [4] This type of situation is what can be called an ideal surrounding for the XP method or what Boehm [8] calls an agile home ground. The case project was carried out in an environment that meets this description.

2.1 The Setting of the Case Project

A team of four software designers was assembled to implement a system (code-named eXpert). The system was designed to document and manage research data and reports at a large research institute [see details on the study in 9]. The consortium had already ordered the development of a similar system from an IT provider. Delivery of that system was promised for late 2003. EXpert, however, aimed to provide an interim subset of this functionality using XP programming in just eight weeks during spring of 2003.

The four developers were 5-6th year university students with 1-4 years industrial experience in software development. The team members were well-versed in java programming language and object-oriented analysis and design approaches. Two weeks prior to the project launch, the team performed a self-study by studying two books on the basics of XP. [i.e., 4, 7] A two day hands-on training in XP practices, the development environment and software configuration management tools was organized to ensure that the team had a sound understanding of XP issues and the technical environment.

The team worked in a co-located development environment. The customer (i.e., a representative from the research institute) shared the same office space with the development team. The office space and workstations were organized according to the suggestions made in the XP literature to support efficient teamwork. Unused book-

shelves, for example, were removed in order to have a maximum amount of empty wall space for user stories and task breakdowns, architecture descriptions, and so on.

2.2 The Case Project Implementation

The eXpert project followed the XP by the book with some additional practices (project planning, explicit software configuration management procedures and roles, data tracking and post-mortem analysis techniques). In what follows the three principal mechanisms (planning game, programming and pre-release evaluation and acceptance testing) in which the functional requirements were realized and to which the customer participated are detailed.

Planning game. The planning game produced requirements for the development cycle in the form of 'user stories'. In practice, user stories were functional requirements, although many descriptions included the word 'user':

- "User must be able to create a workspace."
- "User must be able to create a virtual folder."
- "User must be able to create a virtual file."

The customer was the key voice in the planning game. The customer determined the user stories that should be implemented in each release. No other demands, such as verifiable quality requirements, were explicitly required.

Programming/pre-release evaluation. First, the user stories were divided into 'tasks' for more detailed planning of the software implementation. The main part of this activity was iterative programming – 'pre-release evaluation' cycle. After a set of user stories was implemented, the customer gave informal feedback on the designs. Part of the feedback was comments on the user interface: e.g. "the number of buttons should be smaller in this window".

Designs were modified accordingly until the software team felt that the software had matured enough to be accepted as a release.

The software designers had no style guides or other user interface design guidelines at their disposal. They based their UI design solutions on their intuition or common sense – "we used terms that sounded sensible". They had some basic training in user interface design, which may have influenced the user interface design ("probably the user interface courses at the university were somewhere in the back of my mind").

Acceptance testing. A process called 'acceptance testing' was carried out in the final phase. It was not, however, a true acceptance testing in the sense that the release was in any way accepted. 'Acceptance testing' was in fact a qualitative evaluation by the customer. If errors were detected, this information was used as input for the next design cycle where the problems were to be corrected.

The acceptance testing included usability related feedback as well. They were comments on the design similar to the comments made in the pre-release evaluation, such as "This button should not be red, and should be moved from here to here".

After customer had accepted the release it was put out for a team of 17 users for evaluation and defect-detection. The feedback included some user-interface related feedback as well. The process was not, however, systematic and the emphasis was not directed towards usability of the product. The target was to discover whether the system and the functions included in the release worked or not.

3 Choosing the Assessment Model

Several usability capability maturity (UCM) models have been introduced from the early 90's. Following the capability maturity model trend, the UCM models are aimed at assessing the status of usability engineering practice in a product or system development organization. The principal idea of the usability capability maturity model is that it describes the typical behavior exhibited by a company at a level of capability for each of several organizational dimensions. The result of an assessment is a set of ratings which describe organizational strengths and weaknesses in UCD. In the field of software engineering, process assessment models such as CMMI [6] and ISO 15504 [10] are aimed to produce similar results from systems and software engineering viewpoints.

A number of usability capability assessment approaches have been identified and categorized [11]. Table 1 presents the usability maturity models in a rough chronological order. The table also identifies usability capability assessment categories. The process assessment category includes assessment approaches that address solely process issues. Some of the process assessment approaches are compliant with standard software process assessment; some have diverged features. The generic assessment approaches may include process aspect but also other aspects, such as skills, awareness (of usability) and roles of usability specialists.

Our interest in this study was to understand the relationship between XP and usability engineering in terms of activities (processes): "Does a project really include effective usability engineering activities?" Therefore, an appropriate assessment approach should represent the process assessment category (see categories in Table 1).

Further, our specific focus was engineering – not organizational, support etc. – processes. Moreover, our interest was on the substance – not the management – of the engineering processes. In other words we wished to analyze to which extent usability activities were carried out, not how they were managed. Thus, our approach differs from the process capability evaluation in many regards. In fact, when thinking specifically in terms of the *capability levels* of traditional process assessment, we can say that our interest was on 'below level 1' issues.

Yet, we did not choose traditional process assessment (ISO 18529 or SPA-HSI). While it is effective on analyzing managerial and organizational level issues, it is less concrete at the level of identifying usability engineering at substance level. Our

choice was the UCD-PA method because it is specifically developed for the purpose of the assessment of the usability substance of a development project.

Table 1. Usability Capability Maturity Models

Model	Category	Developer
Trillium [12]	Process assessment (non-standard)	Bell Canada
Usability Leadership Maturity Model [13]	Generic assessment	IBM (US)
Humanware Process Assessment [14]	Process assessment (close to CMM)	Philips (UK)
User Centred Design Maturity [15]	Generic assessment	The Ergonomics and Saftety Research Institute, ESRI), Loughborough University (UK)
Usability Maturity Model: Human-Centredness Scale [16]	Generic assessment	European INUSE project (further refined in the European TRUMP project)
ISO 18529 Human-Centred Lifecycle Process Descriptions [17]	Process assessment (SPICE compliant)	International Standarization Organization (ground work carried out in the European INUSE project)
UCD performance assessment [11]	Process assessment (non-standard)	Oulu University (Finland)
Procedures for Usability Engineering Process Assessment [18]	Generic assessment	DATech (Germany)
COEDA	Process assessment (SPICE compliant) + generic assessment	Mitsubishi Research Institute, NTT advanced technology and Otaru University of Commerce (Japan)
QIU[19]	Process assessment (SPICE compliant)	HFICMM project (UK)

3.1 The UCD-PA Assessment Method

In the following, we describe the main features of the UCD-PA assessment approach: first the reference process model, second the performance ('assessment') dimensions, third the rating levels and finally the assessment procedure.

Reference Process Model

The UCD-PA model [11] defines usability engineering through a set of 'outcomes' that should be produced during a usability engineering life-cycle. Outcomes are typically – but not necessarily always - concrete deliverables (documents) that should be produced during a usability engineering lifecycle. The outcomes are categorized around usability engineering processes, as

Fig. 1 illustrates. The usability engineering processes feed data into the interaction design process, shown in the centre of the model.

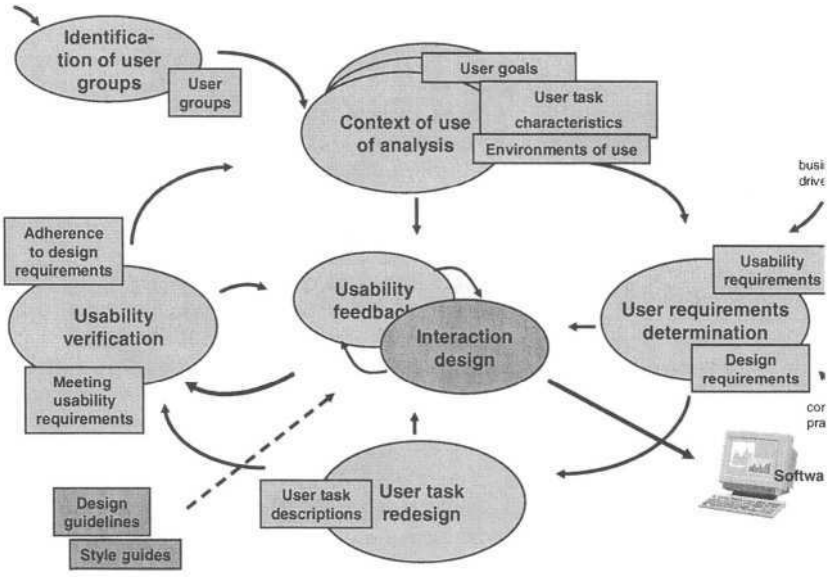


Fig. 1. The usability engineering model of UCD-PA [20]

The core substance of the UCD-PA model is the definitions of the outcomes. Those definitions are presented in Table 2.

Rating Dimensions

The assessment of a development project is conducted by examining the outcomes that are produced during a software development project. The model determines three different dimensions through which each outcome is examined:

- The quantity of an outcome. The more extensively an outcome is produced, the higher performance rating it gets.
- The quality of an outcome. With this dimension, the quality and validity of the outcomes is examined.
- The integration of an outcome. The more extensively the outcomes are communicated and incorporated in other relevant processes, the higher rating is given to integration.

Table 2. Definitions of the outcomes in a usability engineering process model of UCD-PA

Outcome	Description
User groups	The various user groups of the software should be identified. The relevant characteristics of each user group should be described. These may include job descriptions, knowledge, language, physical capabilities, anthropometrics, psychosocial issues, motivation for using the system, priorities, etc. An identifiable name should be given to each user group and the importance of each group (how many users) should be determined.
User goals	User goals should be described in terms of what users need to accomplish with the software, not in terms of the equipment, functions or features of the software. It is important to understand that the user goal is a result, not a description of a task.
User task characteristics	Users achieve the goals through tasks. The characteristics of the tasks describe the nature of the tasks, for example frequency, duration of performance, criticality for errors, and whether the tasks are carried out under pressure or in a stressful situation
Environments of use	The operational environment where the software is used should be regarded as relevant. Environment descriptions may include technical, physical and social factors
Usability requirements	Usability requirements are the required performance of the software. Usability requirements should be given in terms of effectiveness, efficiency and satisfaction in a quantitative way, as defined in [1].
Design requirements	Whereas usability requirements give the drivers for the design, design requirements set the restrictions that should be considered when the user interface elements are designed. Typically these include style guides and company or project standards.
User task redesign	In this process, the design should comply with how users are to carry out the tasks to achieve their goals with the new software being developed. It is essential to understand that better usability means better work practices. This design process produces a set of descriptions of the improved work practice of the users: i.e. how the user will carry out the tasks with the new software.
Usability feedback	Usability feedback is qualitative feedback on the usability of the design proposal. This outcome is typically an iterative set of evaluation results which identify those parts of the design solutions that work, and those which should be improved.
Verification against the design requirements	The design solutions of the user interface are checked against the design requirements and a report of the adherence is produced.
Verification against usability requirements	The software is evaluated in order to determine to what extent the software meets the defined usability requirements.

The basic dimension is ‘quantity’. It determines to which extent an outcome is produced. Examination of the other dimensions is meaningful only if an outcome is produced at least partially. The rating of the ‘quality’ dimension is determined based

on whether appropriate usability methods are used and how professionally they are used.

Usability practitioners often complain that the results of their work are not considered in the design decisions. The 'integration' dimension examines this viewpoint. One can quite reliably check this by interviewing the person conducting usability activities: it is very visible to them whether their work has impact on the design of the software.

The rating scales are derived from process assessment: none, partially, largely and fully.

We emphasize that the examination of a development process through outcomes does not necessarily mean that the outcomes should not be documents. Usability engineering activities basically could be implicit: e.g. the stakeholders could share the same understanding on who are the different user groups of the end product. However, it is not very believable that all the outcomes of usability engineering would exist but not documented. For example, if measurable usability goals (one intermediate outcome of usability engineering) – i.e. figures - are produced, they most probably would be documented.

Assessment Process

The assessment process is qualitative and interpretive research [21], yet similar with traditional process assessment. The project documentation is examined and the stakeholders of the project are interviewed. The data gathered is interpreted and the reviewed and agreed with the interviewees. The ratings are given based on the findings using professional judgment.

4 The Results of the Assessment

We present the results dimension by dimension. We first examine the basic dimension, quantity: to which extent the outcomes were produced. Then we examine the quality and integration of those outcomes that were produced at least partially.

We emphasize that the results are method independent – i.e. the analysis is not based on some specific usability methodology. One of the reasons for choosing the usability engineering assessment model (section 3.1) is specifically its method-independence.

4.1 Production of Outcomes

The results of the analysis of the quantity of the outcomes are presented in Table 3. One can summarize that almost no outcomes were produced. The only outcomes that were produced were 'usability feedback' and 'verification against the design requirements'; even these were produced only partially.

Table 3. Results: the extent to which usability engineering outcomes were produced

Outcome	Finding	Rating
User groups	The interviewees said that the users are "researchers" (the application is to be used in a research institute). However, no specific user categories were identified. While the estimated number of users would be "several hundreds" it is not credible that they would make a totally homogeneous group.	Not produced
User goals	The first activity of the XP process, the planning game, produces 'user stories'. Although some of those stories included the term 'user', the stories proved to be more or less functional requirements. Neither explicit thinking nor analysis of the goals of users – i.e. what users would accomplish with the system – was carried out.	Not produced
User task characteristics	User task characteristics are related to the determination of the user goals. If the user goals were not determined, the task characteristics could neither have been determined	Not produced
Environments of use	No explicit analysis of the environment of use was performed.	Not produced
Usability requirements	While user goals were not determined, the logical consequence is that no usability requirements could have been determined	Not produced
Design requirements	No explicit UI design guidelines, such as style guides, were provided to the designers. The designers "used their own experience and skills" for decisions on UI design solutions. (However, one can claim that the customer had some implicit design requirements in mind. These became apparent when the customer gave feedback on the 'style' in the acceptance test phase.	Not produced
User task redesign	As the tasks and goals of users were not determined, no explicit task redesign was done in practice	Not produced
Usability feedback	The representative of the customer gave feedback on the UI design, for example "I asked for the buttons to be removed from here" and "the number of the buttons should be reduced". This can be regarded as partial usability feedback	Partially produced
Verification against the design requirements	Although explicit design requirements were not determined, some verification type of activity was clearly done as part of the acceptance testing: designs were checked against the implicit design requirements of the customer: "the color of this button should not be red".	Partially produced
Verification against the usability requirements	No quantitative usability verification was performed while no usability requirements were set.	Not produced

Scale: {not, partially, largely, fully} produced

4.2 Ensuring Quality

Examining the quality of an outcome is meaningful only if an outcome is produced. The results of the analysis of the quality of the two outcomes that were partially pro-

duced – usability feedback and verification against the design requirements - are presented in Table 4. The outcomes were produced based on the personal judgment of the on-site customer, and no recognized usability methods were used. However, because the on-site customer belongs to the user population of the software that was developed, we give a rating 'partially ensured quality'.

Table 4. Results: the quality of the process of producing the outcomes

Outcome	Finding	Rating
Usability feedback	Feedback was based on the subjective judgment of the on-site customer. No recognized usability evaluation method was used.	Partially ensured
Verification against the design requirements	The verification was based on the subjective judgment of the on-site customer. No recognized usability method was used.	Partially ensured

Scale: {not, partially, largely, fully} ensured

4.3 Integrating the Results

Examining the integration of an outcome is meaningful only if an outcome is produced. The results of the analysis of the integration of the two outcomes – usability feedback and verification against the design requirements - are presented in Table 5. The level of integration was high: the designers did take into account the outcomes (feedback and verification results) and changed the designs accordingly.

Table 5. Results: the integration of the outcomes into designs

Outcome	Finding	Rating
Usability feedback	All feedback was considered by the designers and the designs were changed accordingly.	Fully integrated.
Verification against the design requirements	All feedback was considered by the designers and the designs were changed accordingly.	Fully integrated.

Scale: {not, partially, largely, fully} integrated.

4.4 Summary

The results can be summarized as follows: *The results of the few usability engineering activities with poor quality were fully considered in the design of the software.*

Yet, as described earlier, the results do not indicate that the process used for software production is poor. In fact, the project was rated to CMMI level two demonstrating a strong software engineering capability. The results merely demonstrate that usability of the end product was not a true concern in this particular project.

The result seemed to be a surprise to the development team and also to the project management, the initiator of the assessment (one of the authors). The expectation was that close co-operation with customer would automatically mean good usability, too. However, the results clearly show that even a close and frequent co-operation between the developers and the customer as such does not ensure good usability at all.

As discussed earlier, we did not find it sensible to evaluate the usability of the product because it had not revealed much of the relationship between XP and usability. The informal usability feedback about the system after it has now been in use clearly indicates that the system has usability problems.

5 Discussion

Our analysis reveals that XP pays almost no attention on the usability of the software, apart from some actions done by the customer that can be regarded as implicit usability evaluations (usability feedback and verification against design requirements). From these results we can conclude that usability has not been an issue when XP was developed. This probably is not a surprise because no representatives from the human-factors, interaction design and usability communities were not invited to participate in the formation of the Agile Alliance [22].

It must be emphasized that this result does not mean that the software that is built would be necessarily bad in terms of usability. On the contrary, it well may be that a software could include a number of good solutions in terms of usability. What the results mean is that the potential good usability of the software would be more or less a coincidence: it would be based (1) on the intuition of the customer and (2) on the designers' intuition in their design skills. In other words, the level of usability of the software is more or less a coincidence; it can be worse or better, depending entirely on the intuitions of the customer and designers.

A considerable set of new activities should be added to the XP process, to make the process truly take usability issues seriously. We argue, however, that the logical fit should not be a problem: user and task analysis could support the production of user stories. Further, user stories could be extended to have usability requirements. Style guides and other user interface design guidance could be provided to the software developers (the creation of these guidelines should be done as a company level activity). The specific challenge here is not to make the XP process less agile: the usability methods should be light and efficient as well. Although XP emphasizes minimal documentation, we do propose some documentation of the outcomes of the usability engineering activities. The documentation, however, needs not to be extensive: a list of the user groups of the product; a list of the key user tasks (is a different thing than a list of functions!); a list of measurable usability targets etc.

On the other hand, there is one good point in the XP setting for effective usability engineering. The few usability related actions were conducted by the customer (although implicitly and with questionable quality). This led to the positive impact that those actions had a full impact on the design of the software. This is seldom the case in a typical setting where usability engineers work within a design team – the designers tend to take the customer's voice more seriously than the voice of a member of a design team. Using the terminology of the UCD-PA model, the integration worked. Thereby, the quality aspect should be addressed. Systematic usability evaluation methods could be used during the software development as well as in acceptance testing. Those activities implicitly exist in the process. Here, no new activity needs to be established – just to transform these activities towards better quality.

We found the UCD-PA method work effectively in this kind of analysis. Examining the outcomes through the three dimensions of the UCD-PA model - quantity, quality and integration – proved to be useful and pinpoint relevant issues. First, it was rather straightforward to judge whether the outcomes defined by the model were produced (the quantity dimension). Even the ratings were rather easy to determine in this case. Second, the other two dimensions of the UCD-PA model - quality and integration – also proved to be useful by pinpointing relevant issues. The few usability actions were carried out by the customer without any systematic and professional approach – this was address by the quality viewpoint. On the other hand, the integration dimension revealed that all of the few, unprofessionally carried usability actions had a full impact on the design of the software.

Based on the way how the UCD-PA model separates these findings, one can draw conclusions about the challenges of improving usability engineering in XP process. As discussed above, for example in case of usability feedback, one should pay attention to improving the quality of the feedback since the integration aspect proved to be efficient.

6 Conclusions

Extreme programming is a development paradigm that is becoming widespread in software engineering. To understand to which extent XP guides to development of software that is usable, a usability process assessment was conducted on a controlled XP project. The results reveal that XP pays almost no attention on the usability of the software, apart from some actions that can be regarded as implicit usability evaluations. On a positive side, however, these few actions – although carried out with questionable quality – had an impact on the design of the software, which is often not so common in the traditional usability engineering community.

One limitation of this study is that it was based on the analysis of just a single XP case project. However, the case project was specifically planned to be a 'by the book' one, and therefore should be a good representative of an XP project. Yet, there is no single 'right way' of doing XP – this has also been recognized in the XP literature. Each real XP project has its own specific characteristics.

What do the results mean for software practitioners who work in an XP environment and are interested to address usability issues? We argue that if the customer is not interested – or more likely, does not understand usability issues – the usability efforts would be based on the interest of the design team. While usability engineering would mean effort expenditure and tasks that that would not be requested by the customer, the consequence may be that usability engineering would not take off.

On the other hand, if the customer would be enlightened from usability issues and would start requiring usability, the position of the design team could be changed even dramatically. How to implement the usability requirements that would be demanded by the customer? Understanding this is bound to become increasingly important in the agile software development community. We conclude that a promising avenue in the search of solutions is to adopt ideas from the usability engineering community that have been brought up in this paper. In turn, the usability engineering community needs to explicitly address the "light but sufficient" paradigm that underlines agile and XP types of processes. Incorporating high-ceremonial practices that require a lot of effort with little concrete as an outcome is in danger of not getting the customer acceptance despite of importance of the software systems' usability.

References

[1] ISO/IEC, "9241-11 Ergonomic requirements for office work with visual display terminals (VDT)s - Part 11 Guidance on usability," ISO/IEC 9241-11: 1998 (E) 1998.

[2] J. Nielsen, *Usability Engineering.* San Diego: Academic Press, Inc., 1993.

[3] ISO/IEC, "13407 Human-Centred Design Processes for Interactive Systems," ISO/IEC 13407: 1999 (E), International Organization for Standardization, Genève, Switzerland 1999.

[4] K. Beck, *Extreme programming explained: Embrace change.* Reading, MA.: Addison Wesley Longman, Inc., 2000.

[5] G. Macomber and T. Rauch, "UCD in the Age of "Web Years", XP, and Agile Programming Methods: Towards "Agile User-Centered Design"," presented at UPA 2003, 2003.

[6] CMMI, "CMMI℠ for Systems Engineering/Software Engineering, Version 1.02. Continuous Representation," Software Engineering Institute, Carnegie Mellon University, Pittsburgh 2000.

[7] R. Jeffries, A. Anderson, and C. Hendrickson, *Extreme Programming Installed.* Upper Saddle River, NJ: Addison-Wesley, 2001.

[8] B. Boehm, "Get Ready For The Agile Methods, With Care," *Computer,* vol. 35, pp. 64-69, 2002.

[9] P. Abrahamsson, "Extreme programming: First results from a controlled case study," presented at Euromicro 2003, Antalya, Turkey, 2003.

[10] ISO/IEC, "15504-2 Software Process Assessment - Part 2: A reference model for processes and process capability," ISO/IEC TR 15504-2: 1998 (E), International Organization for Standardization, Genève, Switzerland 1998.

[11] T. Jokela, *Assessment of user-centred design processes as a basis for improvement action. An experimental study in industrial settings.* Oulu: Oulu University Press, 2001.

[12] F. Coallier, R. McKenzie, J. Wilson, and J. Hatz, "Trillium Model for Telecom Product Development & Support Process Capability, Release 3.0. Internet edition," vol. 1999, Internet edition ed: Bell Canada, 1994.

[13] G. A. Flanagan, "Usability Leadership Maturity Model (Self-assessment Version). Delivered in a Special Interest Group session," Denver, USA 1995.

[14] B. Taylor, A. Gupta, W. Hefley, I. McLelland, and T. Van Gelderen, "HumanWare Process Improvement - institutionalising the principles of user-centred design. In Tutorial PM14 H Human-centred processes and their impact," presented at Human-Computer Interaction Conference on People and Computers XIII, Sheffiel Hallam University, 1998.

[15] K. Eason and S. D. Harker, "User Centred Design Maturity. Internal working document," Department of Human Sciences, Loughborough University, Loughborough, Internal working document 1997.

[16] J. Earthy, "Usability Maturity Model: Human Centredness Scale. INUSE Project deliverable D5.1.4(s). Version 1.2.," Lloyd's Register of Shipping, London IE2016 INUSE Deliverable D5.1.4s, 1998.

[17] ISO/IEC, "18529 Human-centred Lifecycle Process Descriptions," ISO/IEC TR 18529: 2000 (E) 2000.

[18] DATech, "DATech-Prüfbaustein. Usability-Engineering-Prozess, Version 1.2," Deutsche Akkreditierungsstelle Technik e.V., Frankfurt/Main 2002.

[19] ISO/IEC, "A specification for the process assessment of human-system issues. Version 2.1, committee draft," ISO/IEC TC 159 / SC 4 N 704 / WG 6 2002-09-19 2002.

[20] T. Jokela, "Making User-Centred Design Common Sense: Striving for an Unambiguous and Communicative UCD Process Model," presented at NordiCHI 2002, Aarhus, Denmark, 2002.

[21] H. K. Klein and M. D. Myers, "A set of principles for conducting and evaluating interpretative field studies in information systems," *MIS Quarterly,* vol. 23, pp. 67-94, 1999.

[22] L. L. Constantine, "Process Agility and Software Usability: Toward Lightweight Usage-Centered Design," vol. 2003: Constantine & Lockwood, Ltd, 2002.

Empirical Evaluation of Agile Software Development: The Controlled Case Study Approach

Outi Salo and Pekka Abrahamsson

VTT Technical Research Centre of Finland,
P.O. Box 1100, FIN-90571 Oulu, Finland
{Outi.Salo, Pekka.Abrahamsson}@vtt.fi

Abstract. Agile software development, despite its novelty, is an important domain of research within software engineering discipline. Agile proponents have put forward a great deal of anecdotal evidence to support the application of agile methods in various application domains and industry sectors. Scientifically grounded empirical evidence is, however, still very limited. Most scientific research to date has been conducted on focused practices performed in university settings. In order to generate impact on both the scientific and practical software engineering community, new approaches are needed for performing empirically validated agile software development studies. To meet these needs, this paper presents a controlled case study approach, which has been applied in a study of extreme programming methodology performed in close-to-industry settings. The approach considers the generation of both quantitative and qualitative data. Quantitative data is grounded on three data points (time, size, and defect) and qualitative data on developers' research diaries and post-mortem sessions.

1 Introduction

Agile software development has rapidly gained a lot of interest in the field of software engineering. Agile methodologies [see e.g., 1], including extreme programming (XP) [2], emphasize principles such as incremental software development with short iterations, adaptation to changing requirements, close communication, self-organizing teams and simplicity [3].

A fair amount of anecdotal evidence have been published on agile software development methods in the form of descriptive articles, reports, and lessons-learned [e.g., 4-6]. Typically, these sources argue for the effectiveness of agile methods and their practices in specific cases. However, systematic empirical research on agile principals and methods is still mostly missing [7]. An increasing number of empirical evidence on agile methods and their practices (e.g. pair programming and test-first approach) has emerged in the last few years to meet the "urgent need to empirically assess the applicability of these methods, in a structured manner" [7, p. 198]. However, there is apparent confusion among researchers concerning experimentation *per se*. Experiments are often confused with case studies and vice versa. For example,

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 408–423, 2004.

experiments seem to lack the collection and analysis of empirical data for their confirmation [8]. The empirical research strategy also seems to be used in a somewhat vague manner in these studies [cf. 9]. In other words, the strategy chosen is not explicated, which undermines the validity of such a study. Yet there is an empirical body of knowledge available. Most of this knowledge consists of surveys [e.g., 7, 10, 11], case studies [e.g., 12-16] and experiments [e.g., 17-22]. These studies provide evidence on a variety of aspects concerning agile software development methods and techniques.

In order to generate impact on both the scientific and the practical software engineering community, new approaches are needed for performing empirically validated agile software development studies. To meet these needs, this paper presents a controlled case study approach, which has been applied in a study concerning XP methodology performed in close-to-industry settings. The approach considers the generation of both quantitative and qualitative data. Quantitative data is grounded on three data points (time, size, and defect) and qualitative data on developers' research diaries, postmortem reviews, and final interviews.

This paper is composed as follows. The following section gives a brief account of the current state of empirical software engineering literature and identifies the main empirical research approaches. This is followed by a presentation of the controlled case study approach and the principal lessons-learned from the application of the approach including empirical evidence. The paper concludes with final remarks.

2 Related Literature

Empirical studies conducted in the field of software engineering aim at providing a scientific and thus more rational basis for evaluating, predicting, understanding, controlling and improving the tools, methods and techniques used in software engineering [23]. The influence of assumptions and alternative explanations can be eliminated through empirical research, which also serves for exploring and finding explanations for new phenomena in order to develop and support new theories [24].

Empirical research includes both qualitative and quantitative research approaches. The necessity and the diverse difficulty of empirical software engineering are acknowledged [e.g., 23-25] while it is all too often that, in practice, the decisions are still made without any empirical justification [8, 26]. Empirical studies include various forms of research strategies [25] and can be categorized, for example, into surveys, experimentations and case studies [e.g., 27]. Surveys are used for collecting quantitative or qualitative data from a sample group of population by interviewing or using questionnaires. Whereas case studies [e.g., 28-30] also use both qualitative and quantitative data, experiments are purely quantitative, since their focus is on the behavior of measurable variables. In the following, these research strategies are discussed based on the concepts introduced by Wohlin et al. [31] and Bratthall and Jørgensen [32].

One of the key differences between the research strategies is to be found in the *level of control.* For example, "experiments sample over the variables that are being

manipulated, while the case studies sample from the variables representing the typical situation" [31, p. 12]. The experimentation approach can be characterized as "a form of empirical study where the researcher has a control over some of the conditions in which the study takes place and control over the independent variables being studied" [25, p. 456]. The control can be further divided into execution control and measurement control. While case studies lack only execution control, the survey approach lacks also measurement control. In addition, the research strategies also vary regarding their *research environment:* while a survey can usually be conducted as a desktop survey, an experiment is usually carried on in a laboratory or university environment. A survey can also be conducted on-line, yet under the control of the researcher. Case studies, on the contrary, are performed in a real-life context. The focal project aims at producing "real" outcomes. The environment in which a case study is performed is, however, uncontrollable. Other factors distinguishing the research strategies from each other are *investigation cost* and *ease of replication.* The investigation cost rises from surveys to case studies and towards large experiments dramatically. Furthermore, surveys and experiments are easier and far cheaper to replicate than case studies.

The case study approach enables an investigation of extensive phenomena, providing data about, e.g., an entire process or a project. However, the unique nature and uncontrollability of variables tend to cause problems in the generalization of results. Any findings can thus be blamed for "unknown confounding factors", and it is difficult to compare the results [33] as is also their interpretation [27]. The multiple case study approach [34] attempts to address these problems to a certain degree. Yet, the costs of this approach are often significantly higher than those of a single case study, the ability to perform replication reliably is not necessarily guaranteed, and the confounding factor problem and interpretation challenges remain as well.

The research perspective of experiment is usually relatively limited. The focus is on viewing the behavior of a specific set of variables in a defined context. As the experiments are usually conducted in a laboratory or classroom settings they have been criticized for being too unrealistic to allow their results to be transferred to industry [e.g. 33, 39]

Action research [35] can be seen as one form of case study [36]. It focuses more on what practitioners do rather than what they say they do. Moreover, action research produces knowledge for guiding practice [37], which is the principal aim of any empirical research. Unlike in the case study approach, in action research the modification of reality requires a possibility to intervene [38]. Yet, action research has similar limitations to those in the case study approach regarding the generalization of research results, and it may be even more costly due to the fact that it requires action taking and monitoring.

The rallying points of the empirical research strategies described above are to be found in their scientific and systematic approach and concern with quantitative data [27], and their strategy of seeking and validating research results through data collection, analysis and interpretation. In order to overcome the inadequacies of the different empirical research strategies various solutions have been proposed. Wohlin *et al.* [27] encourages a simultaneous use of different research methods. An interplay

of various research strategies is likely to yield the most benefit [40]. Therefore, using experiments to complement case studies is often suggested [33]. As stated above, conducting multiple case studies is rarely possible because of the high costs and the difficulty of finding similar enough cases [32]. However, multiple data sources should be used in case studies to provide a higher degree of validity [32].

Wohlin *et al.* [27] have proposed some guidelines for deciding between the experiment and the case study approach. Since both approaches are suitable for comparing two software engineering methods, the choice depends on the scale of the evaluation. Wohlin *et al.* [27] argue that a case study suits industrial evaluations particularly well for the reason that it enables avoiding scale-up problems and the study itself is capable of detecting a more widespread and long-term impact. The experiment, again, should be chosen as the research approach if the research is more concerned with studying the reasons for certain phenomena or evaluating the differences between two or more methods.

Table 1 presents the phases (first column) and tasks (second and third columns, respectively) of the different empirical research strategies. In Column 2, the Quality Improvement Paradigm (QIP) [41] approach for conducting empirical studies [27] is described. The steps are drawn from [31, 42]. The last column presents the case study research approach according to Eisenhardt [43].

Table 1. Empirical research approaches presented in literature

Generic phases of empirical research	Experimental research	Case study research
Design	**Characterize** Current Situation/Baseline setting Topic selection Background research	**Getting started** Definition of research question Possibly a priori constructs Neither theory nor hypothesis
	Set Goals Formulation of goal in a quantifiable manner	**Selecting cases** Specific population Theoretical, not random, sampling
	Choose Process Setting of research context Formulating hypothesis Determining variables Identifying subjects Setting of instrumentation	**Crafting instruments and protocols** Multiple data collection methods Qualitative and quantitative data combined Multiple investigators
Implementation	**Execute** Prepare study Execute study (collect the data) Validate data	**Entering the field** Overlap data collection and analysis Flexible and opportunistic data collection methods
	Analyze Perform statistical analysis Visualize analysis results Study outcomes Accept/reject hypothesis and draw conclusions	**Analyzing data** Within-case analysis Cross-case pattern search using divergent techniques

Generic phases of empirical research	Experimental research	Case study research
Learning	**Package** Report findings Store data & analysis for further use	**Shaping hypothesis** Iterative tabulation of evidence for each construct Replication, not sampling, logic across cases Search evidence for "why" behind relationships
		Enfolding literature Comparison with conflicting literature Comparison with similar literature
		Reaching closure Theoretical saturation when possible

The traditional empirical research approaches (Table 1) serve their purpose in a wide range of research domains. Agile software development is, however, characterized by rapid iterative cycles and continuous changes in the process and in the product requirements. If the empirically validated scientific data about agile software development is to be generated, the research approach needs to be able to adapt to these settings. This creates the need to effectively combine the experimental, case study and action research approaches. This paper proposes a combined approach, i.e. a controlled case study approach, which will be presented in the following section.

3 A Controlled Case Study Approach

In this section the controlled case study approach is presented. It represents a research approach that, for one thing, is particularly suitable for the study of agile methodologies and, for another, generates impact on both scientific and practical software engineering communities. In conjunction with the introduction of the approach, an empirical case is laid out in which the approach has been applied. This facilitates understanding how the approach is designed to work when conducting research on agile software development.

The controlled case study approach was applied in a software development project called eXpert, in which a team of four developers implemented a system for managing the research data obtained over years at a Finnish research institute (section 4). The research goal of the study was two folded. First, the aim was to empirically evaluate the Extreme Programming (XP) method in practical settings. Second, the research aimed at applying the controlled case study approach in order to assess its suitability for studying agile methodologies. The details of the study can be found in [12].

While the controlled case study approach strives for replication (experimentation) and in-depth data collection (case study), it also has the ability to change the process

(action research) in a close-to-industry setting in which also business pressure is present. It therefore contains some of the features typical of laboratory experiments, such as a *high degree of control* over independent variables, execution and measurement, and environmental conditions. Furthermore, the *ease of replication* of the controlled case study approach is currently under scrutiny in a connection with a replication of the approach being executed at the moment.

A particularly interesting issue regarding the controlled case study approach is to be found in the dual goal structure of the outcome: 1) fully functional software system or a software product, and 2) research data on selected aspects. Accordingly, the focus of the research is on *evaluating the entire process* of software development when using Extreme Programming or some other agile approaches as the development method. The research considers both *quantitative and qualitative* data and respective data collection techniques.

Figure 1 presents the dynamics, relationships and phases of the proposed research approach designed for the evaluation of agile software development methods in practical settings. It is an iterative research approach incorporating an effective utilization of multiple research methods. These are the experimental and case study research strategies supplemented with the action research [see e.g., 44] perspective.

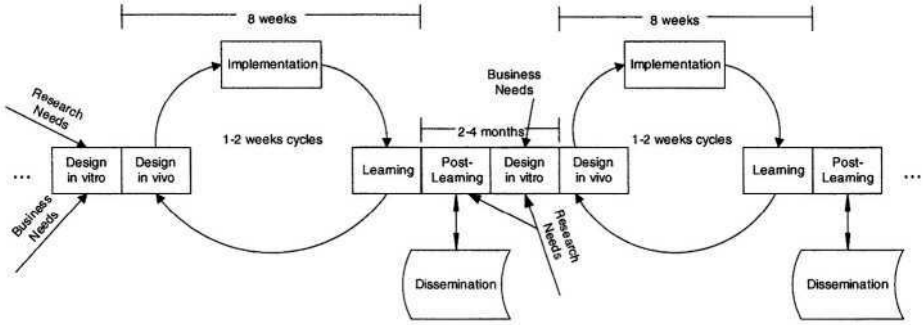


Fig. 1. Controlled case study approach

The phases, steps and outcomes of the empirical research strategy are presented in the following sub-sections. It will also be explained how the different phases and steps were applied in the empirical study (eXpert).

3.1 Design-Phase

The design phase is divided into a design *in vitro* and design *in vivo* phases. The design *in vitro* phase occurs prior to the project under research. It initializes the research and enables setting the focus on the most relevant topics in terms of both business and research. It also includes the steps that need to be done only once during the research such as identification of the research needs and selection of subjects, and

that does not yet involve the subjects of the study, i.e. the project team in this case. The design *in vitro* is intended to last from one to few months at the most.

The *in vivo* design phase is a fixed part of the software development project under research. It is iteratively applied at the beginning of every 1-2 week development cycle (Fig. 1). These cycles need to be synchronized with the actual iterations of the agile software development process. This ensures the ability to refocus the research, to identify additional measures and variables, and to improve data collection techniques, for instance, during the project.

Table 2 presents the steps and outcomes of the design phase of the controlled case study approach.

Table 2. Design phase steps and outcomes

Design phase	Steps	Outcome
Design *in vitro*	Identification of research and business needs	List of potential topics
	Background research	Topics with the highest scientific and business impact
	Target setting	Goal of current research
	Background research/Baseline setting	Current knowledge on chosen topic
	Setting of research context Multiple investigators	Research environment Researchers and other research parties involved
	Identifying subjects using theoretical (not random) sampling	Target project, project members
Design *in vivo*	Neither theory nor hypothesis: Determining variables Multiple data collection methods: Qualitative and quantitative data	Preliminary variables Updated variables Plan for data collection
	Setting of instrumentation	Data collection tools Project documentation templates Training material/Standards Software development tools

In the *in vitro* design phase the needs of both research and industry are determined to support the eventual topic selection. The needs are further elaborated on the basis of preliminary background research (e.g. desktop research). The eXpert project was preceded by an extensive literature review of agile/XP methods [45], highlighting potential research topics. Once the topic for the current research has been selected, a focused background research may be needed. In the eXpert study this included mapping the empirical research done in the field of agile software development methods. This survey revealed that there was very little empirical evidence available of the applicability of agile/XP processes, though some experiments and case studies had been performed regarding certain practices. Therefore, the goal of the research was set: to evaluate the XP software development process as a whole, and to set a baseline for future replications and more focused research endeavors.

Goal setting was followed by setting the research context. This phase includes selecting the environment for the research project and defining the research parties

participating in the process. In addition, the subjects for the research are chosen. In the eXpert project, convenience sampling [31] was used for selecting the nearest and most convenient subjects for the study. In our case, this meant selecting the most experienced of the university students available to be included in the project team. The reason for this was that it was concluded that the validity of the research conducted with experienced students would be comparable with research conducted with practitioners in industry [e.g., 46].

Though no hypotheses are yet set, the variables and methods for their collection need to be defined at the beginning of the *in vivo* design phase (i.e., at the beginning of each iteration). Multiple data collection methods are recommended for multiple sources of information [32]. In eXpert, the applied qualitative data collection methods included group interviews, postmortem reviews and developer's diaries. Quantitative data was grounded on three data points: Time, size and defect as suggested by Humphrey [47]. Time was tracked by minute on XP practices and tasks, size was tracked in terms of lines of code and defects were categorized and recorded systematically. Thus, the generation of both qualitative and quantitative research data was ensured.

Finally, the instrumentation is chosen and prepared, including data collection tools, project documentation templates, standards, training material and guidelines as well as the physical facilities and technical implementation environment for the target project. *In vivo* design requires the ability to identify and develop new data collection mechanisms and to fine-tune the existing ones if needed. In principal, the controlled case study approach calls for redirecting the research in a systematic, controlled and recorded fashion.

3.2 Implementation Phase

The implementation phase is very intense taking only eight weeks. The time frame and developer effort usage are therefore fixed. Flexibility is reserved for delivered functionality, which is in accordance with the principles of agile software development. The aim of software development project is to produce a fully functional software system or a software product[1] for an actual customer. The research process itself focuses on collecting qualitative and quantitative research data, which can be used for several purposes. The research data, in the optimum case, benefits the team as well. While the researchers are mainly interested in studying the agile software development process or some specific part of it, the team can (and should) use the data for software process improvement (SPI) [48]. If the team does not find the data useful, the data collection, even if carefully followed, is bound to become error-prone, as it has been the case in personal software process (PSP) research [e.g., 49]

Implementation begins with the finalization of instrumentation (i.e., data collection devices) and training the subjects into their tasks, i.e. the agile development process

[1] Due to time constraints the systems or products produced are relatively small in terms of size and effort use, e.g. less than 10000 lines of code or 1000-1500 hours of development effort.

under investigation, software development tools and data collection procedures. Training should not take more than 2 days. The actual implementation time data collection is performed on a daily basis using the simplest possible mechanisms. In eXpert, paper, pen and a notebook was used. This was supplemented with a set of simple spreadsheets. The validity of quantitative data is continuously monitored during the project by the project manager, metrics responsible, on-site customer, and the management of customer organization. Data is collected not only for storing purposes but it is also actively used via analysis and visualization mechanisms. This enables the analysis to overlap the collection process in an effective and iterative manner.

Table 3 presents the steps and outcomes of the implementation phase in the controlled case study approach.

Table 3. Implementation phase steps and outcomes

Implementation phase	Steps	Outcomes
	Preparing study	Training Infrastructure: Installing of tools, development of templates
	Executing study	Collected research data
	Solving business problem	Working software system
	Validating data	Validated research data
	Analyzing and visualizing the data from current iteration/Overlapping data collection and analysis/Iterative tabulation of evidence for each construct	Analyzed data Visualized data

3.3 Learning Phase

The learning phase is divided into learning and post-learning phases (Fig. 1.). The learning phase includes the steps taken *during* the software development process at the end of every iteration in the Agile process. The post-learning phase takes place only *after* the actual intense software development process, when all the research data is available. Thus, the learning phase is concerned with improving the current software development process as well as the research process whereas the post-learning process aims at systematic dissemination of research data.

Table 4. presents the steps and outcomes of the learning and the post-learning phases of the controlled case study approach.

The principal mechanism of the learning phase includes postmortem reviews [29], and learning through analyzing and interpreting the collected data. Postmortem reviews are used for enhancing the XP software development process according to the experiences of the related stakeholders in the project. The postmortem review aims at detecting positive and negative issues vis-à-vis the previous iteration. In this phase, the software developers may propose various alterations, which are prioritized and agreed on. In eXpert, the postmortem reviews recurred five times, i.e. after every

iteration. The postmortem review was used not only for enhancing the practices of the software development project but for adapting the research mechanisms to the project. For example, data collection tools were improved and variables were advanced during the project.

Table 4. Learning phase steps

Learning phase	Steps	Outcomes
Learning	Post-mortem reviews/analysis	Suggestions for process enhancement Process enhancements
	Interpretation of analyzed and visualized data from the previous iterations	Experiences of the process Feedback for following *in vivo* design phase as e.g. suggestions for data collection improvement
Post-Learning	Group interview	Developer insights, experiences
	Analyzing and visualizing data from entire project (quantitative and qualitative)	Generalizations & conclusions => scientific dissemination => industrial application
	Store data & analysis for further use	Replications & New controlled case study research projects
	Draw conclusions/Theoretical saturation when possible	Enhanced controlled case study process
	Report findings/Dissemination	
	Identification of future research opportunities	
	Possible involvement of external researchers for data analysis on new perspectives	

The post-learning phase includes several steps designed to feed the next project and to analyze collected data for dissemination purposes. For example, a group interview is conducted to survey the experiences of software developers. All data sources, such as post-mortem session recordings, interviews, spreadsheets and developer's diaries are analyzed and the outcomes stored for further use. Further use also includes identification of the most efficient ways of utilizing collected data. It is often only after the project has ended that some of the data use possibilities are detected. Data analysis may also include the involvement of external researchers enabling the investigation of data from their perspective, which is likely to encourage cooperation between different researchers and promote future studies. While this may be regarded as a radical suggestion, it is based on the assumption that case studies of this sort yield more data than a single team of researchers can utilize effectively.

The aim of the post-learning phase is to generate knowledge for scientific and practical use alike. The post-learning phase also enables an explicit consideration of alterations to the research process, which is why this phase lends itself to launching the planning for the following research projects.

4 Applying the Controlled Case Study Approach: Lessons-learned

The controlled case study approach was applied in the eXpert software development project to evaluate its suitability for researching agile software development methodologies. The lessons-learned section identifies the principal issues that can be used for improving the proposed approach.

Design Phase

The training material and the implementation plan were the principal issues that were documented prior to the launching of the project. This was seen as an effective means of ensuring sound kick-off for the project, and the documentation was also considered something that could be effectively re-used in subsequent replications. The most important finding of the *in vitro* design phase was the sampling of project members. The validity of research with students is, to a high degree, dependent on their level of knowledge concerning software engineering practices [46]. Recruiting more experienced subjects would thus be worth some effort in advertising and persuasion. The qualitative and quantitative research data from the postmortem reviews and final interviews revealed that detailed instructions, e.g. coding standards, should have been created for the project team prior to the project, i.e. in the design *in vitro* phase.

The *in vivo* design phase proved to be an efficient means of improving the data collection capabilities. The postmortem analysis [50] technique – an improvement and reflection device - which was used for evaluating the project after each software release was found an effective way of identifying issues that needed improvement, not only from the practical business perspective but also from research point of view. The post-mortem analysis is performed in the learning phase and it feeds into the design *in vivo* phase. For example, it was identified that team presence was something that could have an influence on the project outcome. The team presence factor indicated the time spent within project facilities, since no work was to be performed outside. This was realized quite early (after the first release) and appropriate measures were taken, i.e. proper instrumentation, to ensure that also this data point was captured throughout the project.

As an example, postmortem findings on time tracking were found to be directly related with the research data collection. During the project, a total of 14 negative and 4 positive comments were given on time tracking. These comments led to various enhancements made in recurring *in vivo* design phases. For example, improvements were made on data collection instructions to ensure a higher degree of data reliability. The spreadsheet used for collecting working hours was also updated with additional column for tasks not related to actual project work, such as coffee breaks.

The exploratory approach to data collection resulted in a more extensive base for establishing the baselines of agile methodologies. Instead of concentrating on just one practice at a time, e.g. pair programming, data collection covered the whole process and provided data for a broad spectrum of analysis.

Implementation Phase

It was found encouraging for the developers to realize that collected data was monitored on a daily basis by several stakeholders. In fact, this proved to underline the importance of data collection and researcher commitment, thus reducing the amount of missing data. Rapid monitoring also revealed various forms of ambiguous data, which could then be immediately revised together with the developers, and either corrected or interpreted more accurately. Based on the experiences gained, the key to success here is the ability to incorporate developers as co-researchers with their own research agenda and interest. In the eXpert case, the developers were responsible for investigating the effect of postmortem review technique within the context of extreme programming.

Yet, it should be emphasized that the principal goal and outcome of the implementation phase is working software. The data should work to yield benefits and not to hinder the progress of the team in practice.

Learning Phase

Although it would have been beneficial to use the data also for software process improvement purposes, this was not achieved in the eXpert project. Due to the focus being mainly on the research aspect, process improvement relied on postmortem reviews only. Moreover, the postmortem review results are solely based on related stakeholders' experiences and opinions. Even though they are valuable as such, they are deemed to fall short without proper tracking mechanisms and therefore lack data for their confirmation. To ensure and improve this aspect of the controlled case study approach, it is suggested that the learning phase is complemented with on-time explicit data interpretation sessions with software developers. These can be embedded in postmortem review sessions. Due to the tight implementation schedule, this may, however, prove difficult.

Regarding postmortem reviews, the bottom-line is the monitoring of whether the suggested and agreed process changes were actually carried out. It is worthwhile to record all the postmortem reviews and also to monitor whether the findings are still valid after the next iteration.

The time reserved for post-learning was a few months, which was found to be too short for effective dissemination. The eXpert-project results are currently being disseminated as the replication study is already in progress. The danger lies in mixing the results of one study with those of another. Thus, explicit emphasis needs to be placed on research discipline. It also needs to be noted that the number of research perspectives one can manage is limited and there will always be data that remains un-analyzed for a long period of time, which is thus in danger of becoming obsolete.

The learning phase, especially postmortem reviews, proved a useful mechanism for improving not only the software development process but also the research process during the project.

5 Conclusions

While agile software development methods have gained wide-spread interest in the field of software engineering, an empirical validation of the ideas presented by agile proponents is very limited. It was claimed that if Agile solutions were to generate impact on both the scientific and practical software engineering community, new approaches to empirically validated agile software development studies would be needed.

To meet these needs, this paper presented a controlled case study approach. The novelty to be found in this approach is twofold: It produces working software and it combines several research strategies aiming at producing valid research data on selected research topics. Both goals need to be treated as equally important or else the close-to-industry setting will not apply. The approach is drawn from experimentation (strives for replication), case study (strives for in-depth nature) and action research (strives for detecting and reacting to changes in the process). The controlled case study approach is explicitly designed to meet the needs of agile software development research: this is done by placing emphasis on the iterative and incremental nature of software development in very short development cycles. The approach considers the generation of both quantitative and qualitative data. Quantitative data is grounded on three data points (time, size, and defect) and qualitative data on developers' research diaries, postmortem sessions, and final interviews.

The controlled case study approach was applied in order to be validated in an XP case study. The results of the case study are in the process of being disseminated at present and another project code-named zOmbie is currently in progress. Plans are already being made for a third replication. While eXpert and zOmbie have been conducted predominantly with student subjects, future studies have been designed to include representatives of industry developing their own software in a specified research setting.

Although the approach presented in this paper strives to increase the degree of measurement and execution control in close-to-industry settings, it does not overcome the principal limitation of case studies – the inability to generalize results. Pure experiments and traditional case studies are also definitely needed and called for in the area of agile software development. It is, however, claimed by the authors, that it is the interplay of all these approaches that will yield better results in the scientific community for the benefit of software industry. For example, the pair programming technique is currently undergoing a series of empirical investigations in terms of planned experiments in different parts of the world. Using the controlled case study approach we are able to test, verify and invalidate (or validate) parts of the findings made in these studies. This is especially the case when investigating the longer term impact of a particular technique and its interplay with other techniques in the industrial context. It would be more difficult to do this in a purely experimental setting. Furthermore, the in-depth nature of the controlled case study approach enables the identification of procedures, processes, techniques and methods that could be placed under experimental evaluation.

References

[1] B. Boehm and R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed:* Addison-Wesley, 2003.

[2] K. Beck, *Extreme Programming Explained: Embrace Change*: Addison Wesley Longman, Inc., 2000.

[3] A. Cockburn, *Agile Software Development.* Boston: Addison-Wesley, 2002.

[4] P. Schuh, "Recovery, Redemption, and Extreme Programming," *IEEE Software,* vol. 18, pp. 34-41, 2001.

[5] J. Rasmusson, "Introducing XP into Greenfield Projects: Lessons Learned," *IEEE Software,* pp. 21-28, 2003.

[6] O. Murru, R. Deias, and G. Mugheddu, "Assessing XP at a European Internet Company," *IEEE Software,* pp. 37-43, 2003.

[7] M. Lindvall, V. R. Basili, B. Boehm, P. Costa, K. Dangle, F. Shull, R. Tesoriero, L. Williams, and M. V. Zelkowitz, "Empirical Findings in Agile Methods," presented at XP/Agile Universe 2002, Chicago, IL, USA, 2002.

[8] M. V. Zelkowitz and D. R. Wallace, "Experimental Models for Validating Technology," *Computer,* pp. 23-31, 1998.

[9] J. Noll and D. C. Atkinson, "Comparing Extreme Programming to Traditional Development for Student Projects: A Case Study," presented at XP2003, Genova, Italy, 2003.

[10] B. Rumpe, "Quantitative Survey on Extreme Programming Projects," presented at XP2002, Alghero, Sardinia, Italy, 2002.

[11] B. Tessem, "Experiences in Learning XP Practices: A Qualitative Study," presented at XP2003, Genova, Italy, 2003.

[12] P. Abrahamsson, "Extreme Programming: First Results from a Controlled Case Study," presented at 29th Euromicro Conference, Belek-Antalya, Turkey, 2003.

[13] D. Karlström, "Introducing Extreme Programming - An Experience Report," presented at XP 2002, Alghero, Sardinia, Italy, 2002.

[14] F. Maurer and S. Martel, "On the Productivity of Agile Software Practices: An Industrial Case Study,"., 2002.

[15] M. M. Müller and W. F. Tichy, "Case Study: Extreme Programming in a University Environment," presented at 23rd International Conference on Software Engineering, Toronto, 2001.

[16] W. A. Wood and W. L. Kleb, "Exploring XP for Scientific Research," *IEEE Software,* pp. 30-36, 2003.

[17] A. Janes, B. Russo, P. Zuliani, and G. Succi, "An Empirical Analysis of the Discontinuous Use of Pair Programming," presented at XP2003, Genova, Italy, 2003.

[18] S. Heiberg, U. Puus, P. Salumaa, and A. Seeba, "Pair-Programming Effect on Developers Productivity," presented at XP2003, Genova, Italy, 2003.

[19] K. M. Lui and K. C. C. Chan, "When Does a Pair Outperform Two Individuals?," presented at XP2003, Genova, Italy, 2003.

[20] M. M. Müller and O. Hagner, "Experiment about Test-first programming," *Software,* vol. 149, pp. 131-135, 2002.

[21] L. Williams, R. R. Kessler, W. Cunningham, and R. Jeffries, "Strengthening the Case for Pair Programming," *IEEE Software,* vol. 17, pp. 19-25, 2000.

[22] M. Rostaher and M. Hericko, "Tracking Test First Pair Programming - An Experiment," presented at XP/Agile Universe 2002, Chicago, IL, USA, 2002.

[23] V. R. Basili, R. W. Selby, and D. H. Hutchens, "Experimentation in Software Engineering," *IEEE Transactions on Software Engineering,* vol. SE-12, pp. 733-742, 1986.
[24] W. F. Tichy, "Should Computer Scientists Experiment More?," *Computer,* pp. 32-40, 1998.
[25] V. R. Basili and F. Lanuble, "Building Knowledge through Families of Experiments," *IEEE Transactions on Software Engineering,* vol. 25, pp. 456-473, 1999.
[26] N. Fenton, "Viewpoint Article: Conducting and Presenting Empirical Software Engineering," *Empirical Software Engineering,* vol. 6, pp. 195-200, 2001.
[27] C. Wohlin, M. Höst, and K. Henningsson, "Empirical Research Methods in Software Engineering," in *Empirical Methods and Studies in Software Engineering, Lecture Notes in Computer Science,* R. Conradi and A. I. Wang, Eds.: Springer, 2003, pp. 7-23.
[28] A. Birk, Dingsøyr, T., Stålhane, T., "Postmortem: Never Leave a Project without It," *IEEE Software,* vol. 19, pp. 43-45, 2002.
[29] T. Dingsøyr, Hanssen, G. K., "Extending Agile Methods: Postmortem Reviews as Extended Feedback," presented at 4th International Workshop on Learning Software Organizations (LSO'02)), Chicago, Illinois, USA, 2002.
[30] B. Collier, DeMarco, T., Fearey, P., "A defined process for project post mortem review," *IEEE Software,* vol. 13, pp. 65-72, 1996.
[31] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering.* Boston: Kluwer Academic Publishers, 2000.
[32] L. Bratthall and M. Jørgensen, "Can you Trust a Single Data Source Exploratory Software Engineering Case Study?," *Empirical Software Engineering,* vol. 7, pp. 9-26, 2002.
[33] D. I. K. Sjøberg, B. Anda, E. Arisholm, T. Dybå, M. Jørgensen, A. Karahasanovic, and Vokác, "Challenges and Recommendations When Increasing the Realism of Controlled Software Engineering Experiments," in *Empirical Methods and Studies in Software Engineering, Lecture Notes in Computer Science,* R. Conradi and A. I. Wang, Eds.: Springer-Verlaag, 2003, pp. 24-38.
[34] R. K. Yin, *Case Study Research Design and Methods,* 2nd ed: Sage Publications, 1994.
[35] D. Avison, F. Lau, M. Myers, and P. A. Nielsen, "Action Research," *Communications of the ACM,* vol. 42, pp. 94-97, 1999.
[36] J. B. Cunningham, "Case study principles for different types of cases," *Quality and quantity,* vol. 31, pp. 401-423, 1997.
[37] P. Oquist, "The epistemology of action research," *Acta Sociologica,* vol. 21, pp. 143-163, 1978.
[38] G. I. Susman and R. D. Evered, "An Assessment of the Scientific Merits of Action Research," *Administrative Science Quarterly,* vol. 23, pp. 582-603, 1978.
[39] C. Potts, "Software-Engineering Research Revisited," *IEEE Software,* vol. 10, pp. 19-28, 1993.
[40] A. Kaplan, *The conduct of inquiry: Methodology for behavioral science.* New York: Chandler, 1964.
[41] V. R. Basili, "Software Development: A Paradigm for the Future," presented at COMPSAC '89, Orlando, Florida, 1989.
[42] T. Sandelin and M. Vierimaa, "Empirical Studies in ESERNET," in *Empirical Methods and Studies in Software Engineering, Lecture Notes in Computer Science,* R. Conradi and A. I. Wang, Eds.: Springer, 2003, pp. 39-54.
[43] K. Eisenhardt, "Building Theories from Case Study Research," *Academy of Management Review,* vol. 14, pp. 532-550, 1989.

[44] C. Eden and C. Huxham, "Action Research for the Study of Organizations," in *Studying Organization: Theory & Method,* S. R. Clegg and S. Hardy, Eds. London: SAGE Publications Ltd, 1999, pp. 272-288.

[45] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, "Agile Software Development Methods: Review and Analysis," VTT Electronics, Espoo VTT Publications 478, 2002.

[46] M. Höst, B. Regnell, and C. Wohlin, "Using Students as Subjects - A Comparative Study of Students and Professionals in Lead-Time Impact Assessment," *Empirical Software Engineering,* vol. 5, pp. 201-214, 2000.

[47] W. S. Humphrey, *A discipline for software engineering:* Addison Wesley Longman, Inc., 1995.

[48] S. Zahran, *Software Process Improvement: Practical Guidelines for Business Success:* Addison-Wesley, 1998.

[49] P. M. Johnson, "The personal software process: A cautionary case study," *IEEE Software,* vol. 15, pp. 85-88, 1998.

[50] T. Dingsøyr, Moe, N.B., Nytrø, Ø., "Augmenting Experience Reports with Lightweight Postmortem Reviews," presented at 3rd Int'l Conference on Product Focused Software Improvement (Profes 01), Kaiserslautern, Germany, 2001.

Good-Enough Software Process in Nokia

Kari Känsälä

Nokia Research Center, P.O. Box 407, FIN-00045 Nokia Group, Finland
kari.kansala@nokia.com

Abstract. Small software development teams are much more productive than large teams. That's been shown time after time. So how does in Nokia – with more than half of our R&D work devoted to software development – face this dilemma? The answer relies on applying the most appropriate software process to a particular business context: the good-enough software process.

1 Introduction

The ability to improve time-to-market, quality, and productivity of software R&D will help ensure Nokia's future business success. The functionality, time-to-market, and quality of software are largely driven by business-related targets and constraints [1]. Thus, the only main characteristic of software that the software community can improve itself is productivity.

In his famous book "Software Engineering Economics" [2], Barry Boehm proved with substantial data that the most important software R&D productivity factor by far is the capability of software developers and development teams. This was once again confirmed by recent case studies [3] from the world's largest software project database, containing data from 6300 projects: small software development teams are much more productive than large teams.

The software productivity dilemma in large companies like Nokia is very real. How does Nokia approach large-scale software development while preserving the productivity of individual and small teams?

When more than a couple of people work together, they have to agree how they cooperate. These agreements are typically called software processes. Nobody likes processes per se, but they are inevitable when large teams or collections of small teams work together. So the question arises: What is a "good-enough" software process in a certain business context in a Nokia business group?

2 Business-Driven Software Process

The Software Process Improvement (SPI) Group in Nokia Research Center's (NRC) Software Technology laboratory has tackled the issue of good-enough software processes in cooperation with Nokia's business groups.

In 1996-2002 the SPI Group gradually created a software process knowledge base, recently called **"Business Framework for SW Process",** on the company intranet

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 424–430, 2004.

[4]. The purpose has been to map software processes (including aspects of size, complexity, technologies, and methods) to the business situations, where they are most useful (see Fig. 1).

Depending on where a specific business or technology unit is operating within the business framework for software processes, various approaches can be applied to improve software processes.

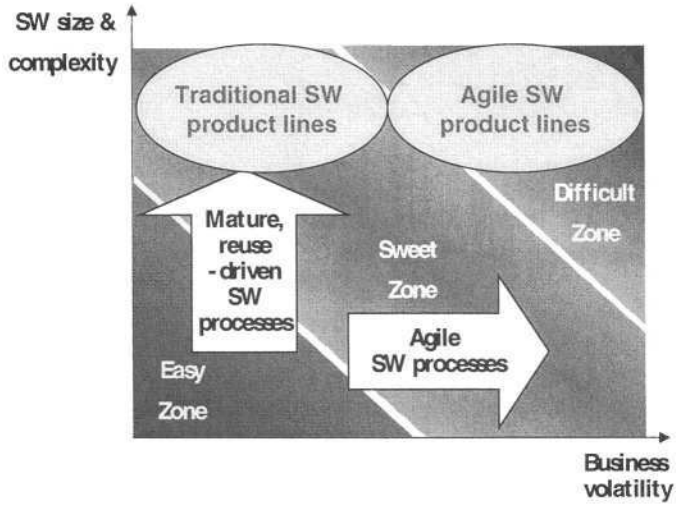


Fig. 1. In the "Business Framework for Software Process", software process information is organized in a matrix, where the x-axis represents increasing business volatility, and the y-axis represents increasing software size and complexity.

The **easy zone** refers to situations, where software with small size and/or low complexity is developed (or maintained) in a stable business situation. Within that zone there are no special SPI challenges.

The **sweet zone** refers either to development large/complex software in a stable business situation, or to development of small-scale software in a volatile business situation. Within these zone there are several optional viable and proven SPI settings.

The **difficult zone** refers to large/complex software in a volatile business situation. This zone is still an unanswerable challenge for the SPI community.

The traditional Nokia approach to tackle the increasing software size and complexity issue has been to increase the maturity of software processes (e.g. based on Capability and Maturity Models) and to increase software reuse (e.g. based on software family engineering).

The business volatility dimension, whose relative importance for Nokia has dramatically increased after the GSM era of the telecom business, requires other means to supplement maturity and reuse-driven software process approaches. Agile software engineering has become the hottest topic in the world of software process research during the last couple of years. There are also other minimalistic approaches, like the Theory of Constraints.

Software collaboration, i.e. not relying solely on own software R&D, but collaborating with Nokia-external software parties, is yet another dimension in the software productivity game.

These alternative approaches and their application in Nokia are discussed in more detail in the following chapters.

3 Traditional Approaches

3.1 Integrated Capability and Maturity Model (CMMI™)

CMM for Software (SW-CMM) [5] has been extensively used in Nokia since 1994. In 1996-2001 there was a corporate policy to measure the software process maturity using SW-CMM as facilitated by the Nokia SW Process Initiative (NSPI) [6], and the analysis clearly proved that those R&D units that reached SW-CMM maturity levels also improved the most the predictability and quality of their software development.

Since the termination of the NSPI in 2001 two things have happened. The international CMM community decided to integrate all CMM-related approaches into one approach called CMMI (CMMI Integration) [7], and Nokia Networks decided to transfer from SW-CMM to CMMI during 2002 [8]. The CMMI User Group was established as a part of Nokia Systems and Software Engineering Forum (NSSEF) to facilitate CMMI methods (Fig. 2) usage in Nokia. Since then CMMI has been widely used also in other business groups of Nokia [9].

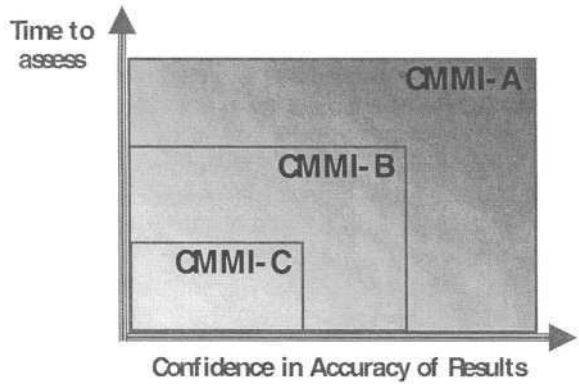


Fig. 2. Nokia's CMMI-based assessment methods portfolio includes: CMMI-C for quick self-assessments, CMMI-B for full-fledged internal assessments, and CMMI-A for the SEI-authorized internal/external assessments.

3.2 Software Family Engineering

Another way of increasing software productivity in a stable business setting is to increase software reuse. In the software engineering literature this approach is called software family (or software product line) engineering (SFE in the following). This approach has been one of the reasons for Nokia's business success, starting from the

DX200 platform as the basis of almost all NET products in the 90's, and more recently with the successful ISA and Symbian platforms as the main basis for NMP's products, as presented in [10]. There are certainly many challenges in this kind of a product development setting, some of which are shown in Fig.3.

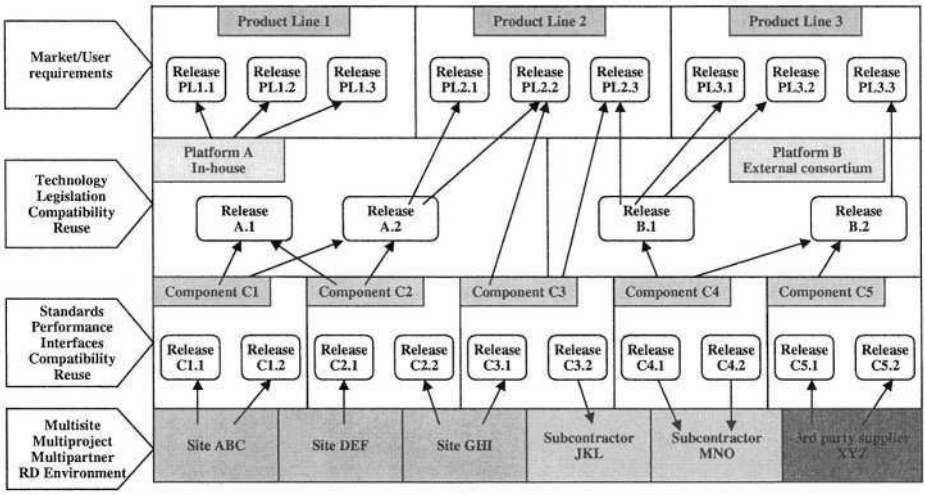


Fig. 3. Reuse driven product development setting in a business group of Nokia.

Key challenges in any reuse (driven product development) setting typically include (this is certainly not an exhaustive list):

- Requirements flow and traceability from product lines to "component units", i.e. units preparing reusable assets (e.g. components)
- Synchronization of product programs in product lines with component development projects in component units
- Integration, releasing and validation ("testing") flow from component units to product lines

NRC has done extensive SFE research since the mid-90's in a series of large European ITEA research projects: ARES, ESAPS, and CAFÉ [11] in order to be able to offer more advanced SFE approaches to Nokia's business groups. In the current FAMILIES project one of the targets is to combine the CMMI with SFE [12] to be able to interpret the CMMI unambiguously in reuse settings, which has really been a major challenge in process assessments of large business groups like in Fig. 3.

4 New Approaches

4.1 Agile Software Engineering

Agile software engineering has become a hot topic in software engineering everywhere in the world and also in Nokia, because it utilizes the high productivity of

small teams and offers flexibility in recent volatile business settings. A current casestudy identified eight agile software engineering approaches in Nokia [13]. In the above-mentioned CAFÉ project the issue of utilizing agile software engineering in large-scale software development was investigated, resulting to a library of agile software development patterns (Fig. 4), which has been applied in several Nokia business groups.

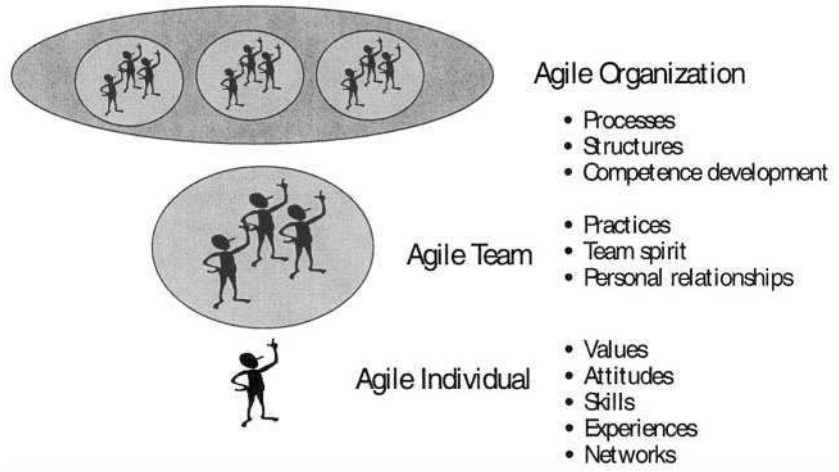


Fig. 4. Agile software development patterns have been classified at three levels: organizational, team, and individual.

A similar way of "avoiding" unnecessary processes is to act with as few processes as possible. One of these approaches is the Theory of Constraints (TOC) [14], which is based on performing root-cause analysis of current software development, identifying the worst bottlenecks to hinder higher productivity, and then removing those bottlenecks one by one. Concerning Nokia's software communities this approach was first used in Nokia Mobile Phones, where it has been successful in terms of solving acute product development bottlenecks in a current volatile situation of that business. Naturally the greatest risk of this kind of approach is to focus on symptoms instead of real root causes, but Nokia Mobile Phones has successfully avoided this risk.

4.2 Software Collaboration

Nokia can no longer rely solely on its software development, but has to collaborate with a large number of external parties like software subcontractors, partners, commercial off-the-shelf software vendors, and even the Open Source software community. One piece in this game is to be able to evaluate those parties from the Nokia viewpoint. A systematic set of approaches, quite similar with the internal CMMI approaches (Fig. 2) has been developed and widely utilized for that purpose.

5 Putting It All Together

With all these good approaches available, the main challenge is: how to put them all together in an actual, quickly changing business setting in business and technology units of Nokia's business groups.

This requires insight from both the management in the business and the SPI practitioners, and the bilateral ability to cooperate to find and deploy the winning combination.

In our understanding, the overall situation should be quite similar in any large industrial corporation, and thus in our opinion similar overall guidelines concerning "good-enough" software process should be applicable there as well.

Acknowledgements. The author of the paper, Kari Känsälä was a Senior R&D Manager at the Software Technology laboratory of NRC in 1994-2003, where he headed a group of consultants supporting software/system process improvement in Nokia's business units, and concurrently he coordinated the Nokia Systems and Software Engineering Forum. In the beginning of 2004 he is moving to a similar position in Technology Platforms of Nokia.

Many of his NRC colleagues, especially Tuomo Kähkönen, Kirsti Saikku and Osmo Vikman, have given valuable input for this paper.

References

[1] Känsälä, K.: Software processes and their effect on productivity. Tieturi seminar on Software Productivity, June 9-10, 2003, Helsinki, Finland.
[2] Boehm, B.: Software engineering economics. Prentice-Hall, Englewood Cliffs, 1981. 767 p.
[3] Hemens, A.: Some case studies in the use of metrics to support process improvement. 8th European SEPG Conference, June 16-19, 2003, London, UK. 5 p.
[4] Känsälä, K.: Managing software engineering knowledge. European SEPG™ 2001, June 11-14, 2001, Amsterdam, the Netherlands.
[5] Paulk. M.C., et.al.: Capability Maturity Model℠ for Software, Version 1.1. February 1993. SEI/CMU-93-TR-24.
[6] Känsälä, K.: Practices for managing a corporate-wide SPI program. European SEPG™ 1999, June 7-10, 2001, Amsterdam, the Netherlands.
[7] SEI/CMU - CMMI Integration project: http://www.sei.cmu.edu/cmmi/
[8] Leinonen, T.: Software engineering under tight economic constraints. Profes 2002, December 9-11, 2002, Rovaniemi, Finland.
[9] Känsälä, K. & Voltti, P. How Nokia moved from SW-CMM to CMMI in a year. European SEPG™ 2003, June 16-19, 2003, London, UK.
[10] Haie, A. Global Software Product Lines and Infinite Diversity. 1st Software Product Line Conference (SPLC1), August 28-31, 2000, Denver, CO, USA, 19 p.
[11] ITEA/CAFÉ project: http://www.esi.es/en/Projects/Cafe/cafe.html
[12] Känsälä, K.: Assessing the maturity of Nokia's software product lines. Seminar on Product Family Development. April 7-10, 2003, Dagstuhl, Germany.

™ European SEPG is a European Community registered trademark of the ESPI Foundation.
℠ CMM and Capability Maturity Model are service marks of Carnegie Mellon University.

[13] Vanhanen, J, Kähkönen, T. & Jartti, J. Practical Experiences of Agility in the Telecom Industry. XP 2003, May 25-29, 2003, Genova, Italy.
[14] Avraham Y. Goldratt Institute: http://www.goldratt.com/

An Ideal Process Model for Agile Methods

Marcello Visconti[1] and Curtis R. Cook[2]

[1] Departamento de Informática, Universidad Técnica Federico Santa María, Valparaíso, Chile
visconti@inf.utfsm.cl
[2] Computer Science Department, Oregon State University, Corvallis, Oregon, USA
cook@cs.orst.edu

Abstract. We present a software process model that can be effectively applied to the agile software development context. Our approach builds an ideal agile software process model starting from the principles established by the agile community in what is known as the Agile Manifesto. The practices defined by the ideal model can be used to assess the various agile methods to determine any missing or under-emphasized practices or practices needing improvement. We compared the practices defined for two of the most popular agile methods (XP and Scrum) with the ideal model and found that these two methods do not explicitly address in an organized way all the principles in the Manifesto. While these approaches do include practices to regularly review the effectiveness of the particular method and tune the method's behavior accordingly for a particular product being developed, they lack explicit practices to perform a retrospective focusing on the process, with the goal of adapting and improving the method itself and its future application. We propose a simple practice that could be added to these methods that would address this apparent oversight. This practice would also provide the ability to leverage what was learned from previous projects to the future projects,. We believe the proposed ideal model is complete, flexible, and can be used to assess and propose simple process improvement actions for agile methods. We also point out some open questions about the best way to share the knowledge gained from retrospectives and to do end of project reviews.

Keywords. Agile methods, process models, assessment, XP, Scrum

1 Introduction

There has been a rapidly growing interest in agile methods for developing software. This interest has been sparked by the many reported successes of agile methods in situations where classical methods have been unsuccessful [2,4,5,6,7,8,11,10,13,14]. Extreme Programming (XP) [3,7] and Scrum [10,14] are two of the most popular agile methods.

Agile methods have been classified as *lightweight* methodologies in recognition of their minimizing bureaucracy and documentation, short iterative cycles, ease of response to changing requirements, and close interaction with the customer. Agile methods are lightweight in the sense that much of the documentation and many of the

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 431–441, 2004.

activities typically done in traditional software development, but that are not directly related to the product being developed, are not performed. For example, documenting that a particular milestone has been completed is not done; rather than designing to allow for all possibilities that may occur, an agile method implements the simplest design that meets the immediate customer needs. Agile methods are considered people-focused in sharp contrast to the process-focused *heavyweight* classical software development methods. Agile development teams are free to organize in any way that they feel will enable them to best accomplish the task. Further, in agile methods the developers work closely with the customer, frequently delivering the increments of the product the customer feels are most important.

Even though agile methodologies are lightweight, they do have a defined process. Some agile proponents may consider this an oxymoron because they believe that agile methods present an alternative to a process-centered approach. They feel that their lightweight and lean development methodologies are distinctly different from the heavyweight, bureaucratic and disciplined plan-driven methodologies [4]. Researchers have investigated the relation between process improvement models and agile methods. However, most of this work has compared the components of their process models with agile methods, such as XP and Scrum, to determine the degree to which these agile methods satisfy the components in their models. In an example of this approach Paulk [9] compared XP practices with SW-CMM[sm1]. He concluded that agile processes generally focus on technical issues while SW-CMM[sm1] generally focuses on management issues and therefore considered them complementary. Turner [11] considered general agile methods in the context of CMMI[sm1] and process improvement. He concluded that agile process "fit into the realm of process improvement" under a "more essential and less literal" interpretation of CMMI[sm]. Another study by Kane and Ornburn [8] investigated the degree to which XP and Scrum satisfy the process areas of the CMMI[sm]. Their comparison showed that XP and Scrum were strongest in the Project Management category and weakest in the Process Management category. This is not surprising considering the effort devoted to short-range planning in XP and Scrum and the lack of focus on process of these agile methods.

Another approach to studying the relation between process and agile methods is to develop a process model for agile methods. In this paper we present a process model for an *ideal* agile method. This ideal model had to embody the essential properties of agile methods since it would be used as the standard in assessing the *agility* of agile methods. Fortunately, the agile community has come together to form the Agile Alliance [2] and produce an Agile Manifesto [1] (see Table 1) that states the principles of agile methods. Our ideal model is based on the principles of the Agile Manifesto.

This paper presents our ideal process model for agile methodologies and uses it to evaluate two of the most popular agile methods, XP and Scrum. The ideal process model is adapted from a software process framework originally designed for constructing a process model for a particular software process or task [12]. Rather than focusing solely on the process dimension of the product or service, that

[1] SW-CMM[sm], CMMI[sm] are service marks of Carnegie Mellon University

framework integrates process, quality assurance, and usability/customer satisfaction. This equal focus on the three dimensions seems to better reflect the customer orientation and other aspects of agile methods than a framework with only a process dimension. The next section presents the structure of the ideal agile process model. In Section 3 we use the ideal process model to evaluate XP and Scrum as defined in standard references. Our evaluation uncovered several agile principles not explicitly addressed in both of these methods, showing particularly that they lack explicit practices to perform retrospectives focusing on the process. In Section 4 we show how the addition of a simple practice to XP and Scrum would address this weakness. Finally, we present our conclusions and future work.

Table 1. Agile Principles

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
Welcome changing requirements, even late in development. Agile process harness change for the customer's competitive advantage
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale
Business people and developers must work together daily throughout the project
Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation
Working software is the primary measure of progress
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely
Continuous attention to technical excellence and good design enhances agility
Simplicity – the art of maximizing the amount of work not done – is essential
The best architectures, requirements, and designs emerge from self-organizing teams
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

2 Ideal Agile Process Model

Our proposed ideal agile software process model is an adaptation of a software process model framework [12] for a single process task and whose main purpose is to serve as a guide for the identification of key practices that drive software process assessment and improvement for that task. In this section, we describe the original framework and explain how it was adapted and applied in the agile software development context.

2.1 Background: Software Process Model Framework

The software process modeling framework was designed to overcome process-only assessments shortcomings, help produce more useful information to proceed with

planning for process improvement, and construct models for single-focus process areas. Rather than focusing entirely on process, the framework incorporates assuring the quality and usability of the end products produced or customer satisfaction with the service provided by the process. The framework has three dimensions (Core Process, Quality Assurance, Usability/Customer Satisfaction) and four phases (Identify, Monitor, Measure, and Feedback) associated with each dimension. Table 2 illustrates the framework. We see that the structure of the framework reflects not only the process itself, but also the quality and usability/customer satisfaction of the products or service produced by the process. Note that there are separate Identify and Monitor phases in each dimension, but combined Measure and Feedback phases. The reason for the combined Measure and Feedback phases is that activities that support measurement and feedback are dimension-independent.

Table 2. Software Process Model Framework

Phase	Dimension		
	Core Process	**Quality Assurance**	**Usability/Customer Satisfaction**
Identify	Define important practices of process for generating product or providing service	Define important quality assurance practices for product or service	Define important practices for product usability or customer satisfaction
Monitor	Monitor adherence to process	Monitor quality assurance activities	Monitor usability/customer satisfaction activities
Measure	Define, collect and analyze measures		
Feedback	Generate, evaluate, prioritize and incorporate recommendations		

The process models developed using this framework are different from other process maturity models in several important ways: in the model developed using the framework there is equal focus on process, quality assurance and usability/customer satisfaction; the framework is specifically geared for developing a model for a particular process or task rather than encompassing the entire development process; and, the framework is flexible and the models developed using the framework can easily be tailored to the particular process and the particular organization. More details about the original process framework, how it differs from other models and application examples, are given in [12].

2.2 Proposed Ideal Agile Process Model

The Agile Manifesto [1] helped us overcome the problem of what properties of agile development methods to include in developing an ideal agile process model. The Agile Manifesto principles are the product of a consensus among all the main agile proponents. Hence, rather than attempting to capture the essential aspects of all or the

most commonly used agile methods in our model, we could focus on capturing the essential aspects of the Agile Manifesto. Thus, using the process model and the Agile Manifesto we were able to form an initial version of the ideal model for an agile-compliant software process, i.e. a process that satisfies both the requirements outlined in our earlier software process modeling framework and the principles expressed in the Manifesto.

Table 3. Ideal Agile Process Model

Phase	Dimension	
	Core Process	**Customer Satisfaction**
Identify	• Deliver working software frequently • Constant interaction between customers and developers • Maximize the amount of work not done • Face-to-face communication in development team • Constant rate of development • Self-organized teams	• Satisfy customer • Welcome changing requirements • Continuous attention to technical excellence and good design • Provide environment and support needed
Monitor	• Early and continuous delivery of valuable software • Business people and developers work together daily • Trust developers to get the job done • Proper environment and support provided	• Early and continuous delivery of valuable software • Business people and developers work together daily
Measure	• Working software is primary measure of progress	• Working software is primary measure of progress
Feedback	• At regular intervals, team reflects on how to become more effective, tunes and adjusts its behavior	• Satisfy customer • At regular intervals, team reflects on how to become more effective, tunes and adjusts its behavior

The original process model framework separates Quality Assurance and Usability/Customer Satisfaction. However, the agile principles stress user and customer involvement, and in agile methods software quality assurance and customer satisfaction are tightly related to whether the final working software does or does not meet customer expectations; hence, the quality assurance and usability/customer satisfaction dimensions are combined into a single dimension. We call this single dimension Customer Satisfaction. Table 3 presents the Ideal Agile Process Model (IAPM) showing the mapping of the agile principles to the phases and dimensions of the model. The Model emphasizes the distinction between the process (Core Process dimension) and the product (Customer Satisfaction dimension). This distinction is

important because some agile practices apply to the product, some to the process and some to both. Hence, a comparison of the IAPM with a particular agile method will show clearly any weakness on either dimension making it more difficult to overlook a potential practice not explicitly addressed.

3 Application of Ideal Agile Process Model

In this section we illustrate the application of the proposed IAPM to two of the most popular agile methods: XP and Scrum, with the goal of determining to what extent each of the practices or components as defined for XP and Scrum satisfies the principles prescribed in the Manifesto, i.e. *how agile* is the method. It is important to note that for this evaluation we are using the practices and components as defined in the standard references for XP [3,7] and Scrum [10,14]. The application will demonstrate that the proposed ideal agile model helps in identifying strengths and weaknesses for both methods in terms of their fitness to the agile principles expressed in the Manifesto; it also indicates potential areas of improvement, in terms of simple practices that could be added to the agile method while preserving the agile spirit.

3.1 Application to XP and Scrum

A summary of main practices and components of XP and Scrum is given in Table 4. These practices and components were taken directly from the standard references [3,7,10,14]. For convenience of the reader, brief descriptions are included for each practice and component. The last 2 items for XP and the last 4 items for Scrum are components or key properties of the method.

Table 5 compares the proposed ideal agile process model with XP and Scrum. Rather than using an assessment questionnaire to determine the degree of satisfaction, an assessment technique typically used in process improvement efforts, we use a lightweight approach. In this approach the practices and components of each method are matched with the practice or practices of the ideal model they satisfy. Both XP and Scrum have explicit practices that address most of the four phases for each dimension. However, the one ideal practice that neither method explicitly addresses is the feedback phase in the core process dimension. While XP and Scrum have explicit practices that evaluate how well product development is proceeding, they do not have explicit practices for evaluating how well the process itself is working.

It is important to be clear about the message in Table 5. It says that according to the practices and components given in the standard references for both XP [3,7] and Scrum [10,14] there is no regular explicit feedback about the process. However, this lack of regular explicit feedback appears to be in the definition of each method and not in the actual practice or way in which these methods are taught. Several proponents of these methods [18,19] encourage the use of retrospectives at the end of each iteration in XP or sprint in Scrum. In these retrospectives the team reflects on what happened during the last iteration and if necessary, tunes its process accordingly.

We believe that both methods should include some simple practice to address this, and propose such a practice in next section.

Table 4. XP and Scrum – Summary of Practices and Components

XP	Scrum
• On-site customer – have an actual user on the team full-time to answer questions • Planning game – predicting what will be accomplished (release planning) and determining what to do next (iteration planning) • Sustainable pace/40-hour week – working overtime when it is effective, normal work maximizes productivity • Testing – developers continually write unit tests that must run for development to continue and customer write tests to demonstrate that functions are finished • Simple designs – the team keeps the design exactly suited for the current functionality of the system • Pair programming – two programmers, sitting side by side, at the same machine • Collective code ownership – any pair of programmers can improve any code at any time • Coding standard – all code in the system looks as if it was written by a single, very competent individual • Metaphor – common vision of how the program works • Refactoring – continuous design improvement, focusing on removal of duplication, increasing cohesion, lowering coupling • Small releases – first, team releases running, tested software chosen by customer, at every iteration; then, team releases to end users frequently • Continuous integration – system fully integrated at all times, multiple builds frequently • Whole team – all contributors are members of one team, including customer, manager, testers • Big visible chart – basic management tool	• Product owner – prioritizes the product backlog • Scrum master – responsible for making sure a scrum team lives by the values and practices of scrum –protects the team from overcommiting • Sprints – series of iterations that determine project progress • Product backlog – master list of all functionality desired in the product • Scrum teams – commits to completing top priority items during a sprint • Sprint backlogs – top priority items during a sprint • Sprint planning meetings – determination of which tasks will move from product backlog to sprint backlog • Daily scrums – brief daily meetings • Sprint review meetings – scrum team shows what they accomplished during the sprint – demo of new features • Empirical management –clear visibility into project based on first hand observation and backlog graphs • Self-organizing teams • No requirements changes during sprint • No specific engineering practices prescribed • Sprint backlog graph – used to reflect progress

The assessment does point out one big difference between XP and Scrum, namely that XP is mostly engineering-oriented and Scrum mostly management-oriented. This is illustrated in Table 5 in which XP does not address the need to provide and monitor proper environment and support (marked (1) and (2) in the table), while Scrum does it through the Scrum Master. The opposite occurs for continuous attention to technical excellence and good design: where XP does address it through a number of engineering practices (marked (3) in the table), Scrum does not consider any particular practice. Another important difference is that XP stresses user involvement more than Scrum does: while users are on-site as part of the team at all times in XP,

they do not belong to the Scrum team and the interaction is performed indirectly through the product owner and not on a daily basis (marked (4) in the table). This suggests that both agile methods can be used jointly given the complementary nature of their practices. This is consistent with the changes to XP proposed by Miller [15], where a new set of 19 practices (some old, some new, some changed) instead of the original 12 is presented and organized as joint practices, programmer practices, management practices and costumer practices.

Table 5. Ideal Agile Process Model – Mapping of Practices for XP and Scrum

Phase/Dimensions	Ideal Agile Practices	XP Practices	Scrum Practices
Identify/Core Process	**Deliver working software frequently**	Small releases and continuous integration	Sprints
	Constant interaction between customers and developers	Whole team and on-site customer	Scrum teams, product owner
	Maximize the amount of work not done	Planning game	Sprint planning meeting
	Face-to-face communication in development team	Whole team and pair programming	Daily scrum
	Constant rate of development	Sustainable pace/40-hour week and small releases	Sprint
	Self-organized teams	Whole team	Scrum team
Monitor/Core Process	**Early and continuous delivery of valuable software**	Small releases and continuous integration	Sprint review meetings, empirical management
	Business people and developers work together daily	Whole team and on-site customer	(4)
	Trust developers to get the job done	Pair programming, collective code ownership, metaphor	Scrum teams and sprint review meetings
	Proper environment and support provided	(2)	Scrum master
Measure/Core Process	**Working software is primary measure of progress**	Small releases, continuous integration and big visible chart	Sprint review meetings, empirical management and sprint backlog graph
Feedback/Core Process	**At regular intervals, team reflects on how to become more effective, tunes and adjusts its behavior**		
Identify/Customer Satisfaction	**Satisfy customer**	Testing	Product backlog, sprint review
	Welcome changing requirements	Small releases and continuous integration	Product backlog
	Continuous attention to technical excellence and good design	Testing, simple design, pair programming, refactoring and coding standards	(3)
	Provide environment and support needed	(1)	Scrum master

Monitor/Customer Satisfaction	**Early and continuous delivery of valuable software**	Small releases and continuous integration	Sprint review meetings, empirical management
	Business people and developers work together daily	Whole team and on-site customer	(4)
Measure/Customer Satisfaction	**Working software is primary measure of progress**	Small releases, continuous integration and big visible chart	Sprint review meetings, empirical management and sprint backlog graph
Feedback/Costumer Satisfaction	**Satisfy customer**	Testing, small releases, continuous integration	Sprint review meetings
	At regular intervals, team reflects on how to become more effective, tunes and adjusts its behavior	Small releases, continuous integration and big visible chart	Sprint review meetings, empirical management and sprint backlog graph

3.2 Proposed Improvements to XP and Scrum

Table 5 shows that when XP and Scrum were compared to the IAPM, both methods lack an explicit practice to provide feedback about the process of applying the method. We feel the feedback is important because it provides a way to:

- determine how well an agile method is working
- improve or tailor the agile method
- determine which parts of the particular agile methods are being carried out
- collect body of knowledge about the experiences with and changes to a particular agile method and pass this knowledge to others in the organization.

A next question is how to add a feedback practice to XP and Scrum in a way that preserves the agile spirit.

One simple way that this could be done is to add a process review practice similar to the idea of post-mortem analysis from traditional development, but applied at the end of each iteration in XP or sprint review in Scrum instead of only at the end of the project. There still could be a final review when the project is finished. This end of iteration review will provide an opportunity to identify any problems in the method application and implement a change for the next iteration. Then at the end of the next iteration the change can be evaluated. This approach to a feedback practice seems to be in harmony with the lightweight philosophy of agile methods.

The periodic reviews could be used to analyze any particular agile practice and its implementation. For instance, customer involvement, agile rules enforcement, impediments or process/method-related obstacles that interfere with doing the work, and so forth.

Our proposal is consistent with what has been proposed by other authors. Miller [15] suggests a new XP joint practice named Retrospectives, where he proposes that collective reviews should happen in an organized way at least after every release, probably after every iteration. He also points out that everybody should take part in

the retrospective, both the technical and the non-technical people. Fowler [16] proposed viewing process changing over time as a part of process self-adaptivity; further, he suggested that it can be deepened if teams do a more formal review at major project milestones following the project retrospective sessions outlined by Kerth [17]. Keith proposed that the review focus on the questions: what did we do well? what have we learned? what can we do better? what puzzles us? Fowler [16] also points out that in XP reviews are neither emphasized, nor part of the process, and proposes making them one of the XP practices. Rising [18] has suggested the use of periodic checkups at the end of an increment, such as a sprint in Scrum, and then using the information gathered at these checkups to uncover patterns to improve the process.

There are other questions about taking these reviews and retrospectives to the next level and improving the process. For example, how to share the information gathered during the feedback sessions in an agile way? Some of the proposed alternatives [18] include: repository, web site, meetings, training courses, story-telling. Another important question is what is the best way to close a project and leverage the lessons learned at the end of a project? Fowler [16] proposed holding 2-3 day off-site formal process reviews. We think that further research can be conducted aimed at answering these questions in order to keep improving the agile methods both on their conceptual definition as well as their practical application.

4 Conclusions and Future Work

We have presented an ideal agile software process model that can be used to assess *how agile* a particular agile method is. The model is based on the agile principles as defined in the Agile Manifesto, thereby avoiding favoring any particular agile method. We have used the model to assess two of the most popular agile software development methods: XP and Scrum, based on the explicit practices defined by their proponents.

The assessment of XP and Scrum has shown that as defined in the standard references both agile methods do not fully and explicitly comply with the principles defined in the Agile Manifesto, the most noticeable being neither method addresses in an organized way the need to reflect and evaluate how effectively an agile method process is working, in order to adjust its behavior to improve its effectiveness. The assessment also showed the differences between XP and Scrum, in terms of their orientation. While XP is mostly engineering-oriented, Scrum is mostly management-oriented.

We believe by separating the process and product, the proposed ideal agile process model captures the essential principles of agile methodologies and hence can be used to evaluate the strengths and weaknesses of agile methods and to propose new practices that could be added to improve them.

We have also identified two important open questions for further improvement of agile methods: how to share the information gathered in feedback sessions?, and what to do at the end of a project?

Finally, in the immediate future we plan on using the ideal model to assess other agile software development methods.

References

1. Agile Alliance. Manifesto for agile software development, http://agilemanifesto.org/, 2001
2. Agile Alliance, http://www.agilealliance.com/, 2001
3. K. Beck. Extreme Programming Explained: Embrace Change. Longman Higher Education, 2000
4. B. Boehm. Get ready for agile methods, with care. *IEEE Computer,* **35(1)** (2002) 64-69
5. A. Cockburn. Agile Software Development. Addison-Wesley, 2002
6. J. Highsmith. What is agile software development? *CrossTalk,* **15(10)** (2002) 4-9
7. What is Extreme Programming, http://www.xprogramming.com/xpmag/whatisxp.htm/, 2002
8. D. Kane and S. Ornburn. Agile development: weed or wildflower? *CrossTalk,* **15(10)** (2002) 30
9. M. Paulk. Extreme Programming from a CMM perspective. *IEEE Software,* **18(6)** (2001) 19-26
10. K. Schwaber and M. Beedle. Agile Software Development with Scrum. Prentice Hall, 2002
11. R. Turner. Agile development: good process or bad attitude? Proceedings of 4th International Conference on Product Focused Software Process Improvement PROFES 2002, Springer-Verlag *Lecture Notes in Computer Science* **2559** (2002) 134-144
12. M. Visconti and C. Cook. A meta-model framework for software process modeling. Proceedings of 4th International Conference on Product Focused Software Process Improvement PROFES 2002, Springer-Verlag *Lecture Notes in Computer Science* **2559** (2002) 532-545
13. P. Wendorff. Organisational culture in agile software development. Proceedings of 4th International Conference on Product Focused Software Process Improvement PROFES 2002, Springer-Verlag *Lecture Notes in Computer Science* **2559** (2002) 145-157
14. Scrum, http://www.controlchaos.com/, 2003
15. R. Miller. Demystifying Extreme Programming: "XP distilled" revisited, Part 1, http:// http://www-106.ibm.com/developerworks/java/library/j-xp0813/, 2002
16. M. Fowler. The new methodology. http://http://www.martinfowler.com/articles/newMethodology.html, 2003
17. N. Kerth. Project retrospectives: a handbook for team reviews. Dorset House, 2001
18. L. Rising. Using patterns to improve process improvement: the argument for a resident minstrel. JAOO 2002 Java Technology and Object-Oriented Software Engineering, Aarhus, Denmark, 2002
19. C. Collins, R. Miller. Adaptation: XP Style, Extreme Programming XP 2001, Cagliari, Sardinia, Italy, 2001

Experimental Development of a Prototype for Mobile Environmental Information Systems (MEIS)

Ari Keronen[1], Mauri Myllyaho[1], Pasi Alatalo[1], Markku Oivo[1], Harri Antikainen[2], and Jarmo Rusanen[2]

[1] Department of Information Processing Science, PL 3000,
FIN-90014 University of Oulu, FINLAND
{ari.keronen, pasi.alatalo, mauri.myllyaho, markku.oivo}@oulu.fi
[2] Department of Geography, PL 3000,
FIN-90014 University of Oulu, FINLAND
{harri.antikainen, jarmo.rusanen}@oulu.fi

Abstract. Environmental information systems (EIS) have been in professional use for quite a long time. Applications of this domain often include features of quite common location based services (LBS). This means utilization of space-, time- and content-oriented data for positioning and routing services with various maps. Firstly, new integrated technology and lower prices of related technology have made it possible for users to benefit from new mobile services with reasonable cost. Secondly, new kinds of ambient aware applications are active research and development topics. We define mobile environmental information systems (MEIS) as integrated mobile information systems used to study, monitor and exploit the nature as well as to guide users like tourists and biologists in the nature. In this paper we present our research in progress, where we have built the first MEIS prototype to be used as a basis for MEIS services that exploit LBS and ambient awareness. The main purpose of this experiment has been to explore the technologies and methods for efficient building of easily extendable mobile platforms that would support main MEIS functionalities. According to our literature survey no similar technical solution as a mobile phone service did not exist.

1 Introduction

Environmental information systems (EIS) combine knowledge of life sciences, information technology and information processing. The aim of EIS is to improve the effective and rational use of environmental resources, environmental protection and to collect environmental related data. With analyzing the collected data changes in the environment can be detected and they can be responded. By minimizing the damaging effect to nature caused by the use of environmental resources a durable and environmentally friendly development and environmental preservation can be achieved.

This paper presents a project of developing an implementation of Mobile Environmental Information Systems (MEIS). It is part of a larger research project studying the potential of wireless applications in the collection and management of environmental information carried out by professionals, as well as the use of these applications for tourism and educational purposes.

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 442–456, 2004.

Nunamaker [13] shows that also systems development can be considered as an important methodology in the information systems (IS) research. During the project one functional prototype has been developed following quite closely the IS research development process he presents, see Figure 1. Only the final steps of this process, like evaluation of the prototype by laboratory or field experiment, still need to be done.

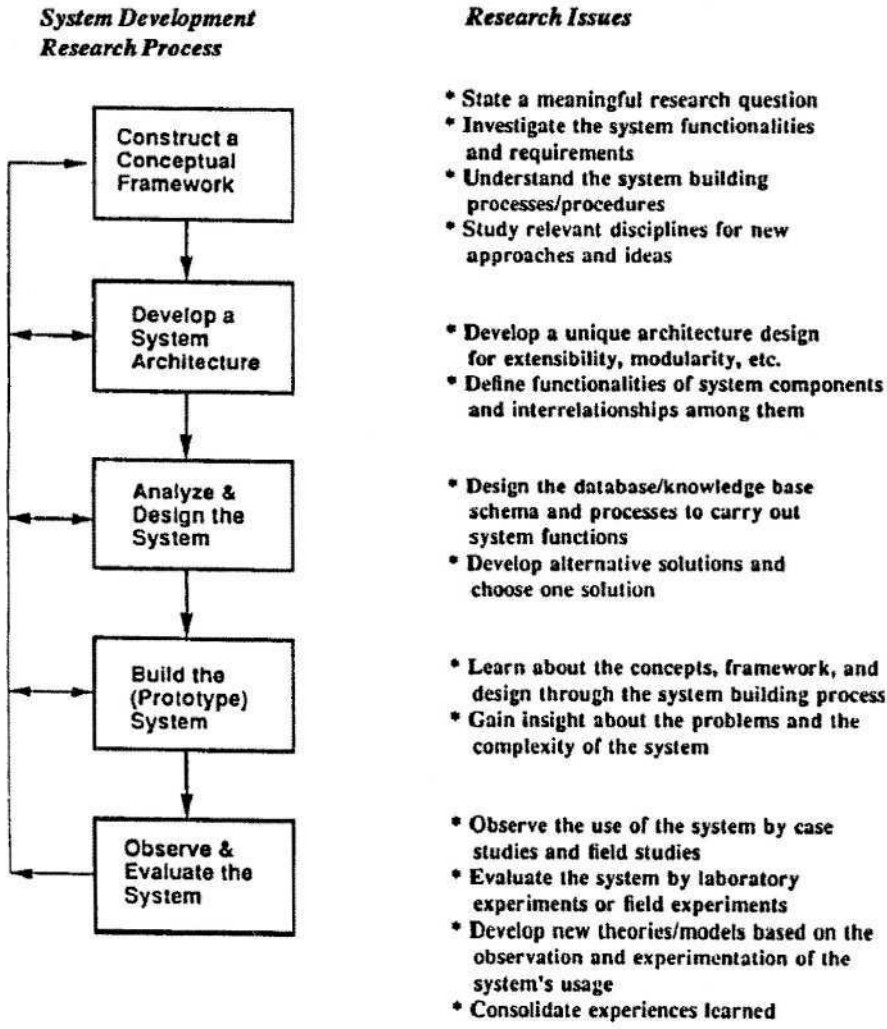


Fig. 1. A process for systems development research [13]

The first prototype to be developed was a location-based application for the visitors of the University's Botanical Garden, which is situated within the campus area of University of Oulu. Using the garden as a primary demonstration environment has several benefits, which include its close location to the University, its compact size,

and versatile nature. The role of the first implementation was to familiarize the development team with the programming environment, its specific tools and techniques, as well as to develop a set of basic components to be used as a basis for further applications.

This paper presents an explorative solution addressing the problems pertaining to contemporary wireless technology platforms. The prototype was mainly based on J2ME (Java 2 Micro Edition). The main strengths of the chosen platform include its ability to support robust user interfaces and functionality, such as map rendering. In addition to this, J2ME is scalable, open and independent of the operating system and lastly forward compatible with standard Java [17].

This paper is organized as follows. Section 2 describes two important key components of MEIS: location-based services (sometimes including context of use), and environmental information systems. Mobile and ubiquitous (or ambient aware) computing could also be added as the third key component of MEIS. Section 3 explains the research design and original design problems. Section 4 gives ideas how to add LBS functionality to the prototype, and there are some considerations about the testing of the mobile applications and from including open source software in to the development. In Section 5, we describe the implementation technologies Symbian and Java. Section 6 presents available positioning techniques and some considerations about using current versions of mobile Java with LBS. Finally, in Sections 7 and 8 we give more details from the MEIS prototype development and our findings, and end with some concluding remarks.

2 Mobile Environmental Information Systems (MEIS)

Wireless information systems are defined in [9] as "computing systems that provide the ability to compute, communicate and collaborate anywhere at any time". In this context, MEIS are defined as wireless information systems used to study, monitor and exploit the natural environment, and their interaction with human activities. From our point of view, location and ambient awareness are essential characteristics for the success of any environmental information transmitted via mobile applications.

The starting point for our study is the fact that the use of computers in environment-related research and monitoring is already well established. Tools like Geographical Information Systems (GIS), databases, and digital collections are nowadays essential for any large-scale study about the natural environment. However, as yet the usage of mobile computing devices in this domain is largely absent, and categories of typical location-based applications have seldom taken account of biological or environmental information and conditions. Currently, large amount of environmental data is being gathered and stored using traditional methods by a number of different organizations. Great deals of these data remain unused because of the lack of ways of disseminating the data and integrating it with other data sources [6, 22]. Consequently, there is a need for streamlining the data collection process towards on-line and real-time data management. The promotion of MEIS is an attempt to address all these issues by designing services that facilitate real-time data collection by a multitude of mobile users, as well as the utilization of existing environmental data sources in context-aware applications [2].

2.1 Location-Based Services (LBS)

The rapid development of wireless telecommunications and the Internet, in conjunction with improved positioning technologies, have given rise to a new class of mobile applications and services, which are location-based [15]. The widely adopted term Location-based Services (LBS) refers to information services that are accessible through a mobile handset and are based on the current geographic location of the mobile device. The conventional application areas of LBS include mapping, tracking, routing, data collection and public safety [3, 10, 21]. The common denominator of these applications is the use of a positioning capability as a value-adding factor. The combination of location information with either spatial or non-spatial databases, as well as other information services, has the potential to add another dimension to the conventional information resources.

The usefulness of a wireless application is no different to any other applications. The needs of the user must be met - the success of a wireless application depends on the suitability of the service content to the mobility of the user. In addition to this basic aspect, LBS have to meet several technical requirements pertaining mostly to high performance level, lightweight client-side hardware, scalability and large number of requests. LBS have to provide a high degree of responsiveness and immediacy of information to users (subject to security and connectivity constraints), they have to support clients with restricted processing power, memory and graphical output, and the LBS architecture must also provide the inherent scalability to serve a large number of potential users without any significant performance degradation [10].

Because the use of LBS amounts to dealing with large datasets, a special emphasis has to put on the design of the architecture and the division of labor between the server and client side. The more content (geographic data as well as additional information) is situated on the server side of the system, the more data flow is needed between server and client (i.e. the mobile device) and the more streaming capacity has to be available. This seems to give preference to a client-side system that is configured to concentrate as much data and tools on the client-side of the system as possible [19]. However, the conventional mobile service platforms, like WAP (Wireless Application Protocol), do not provide all the features required by complex application side functionality.

2.2 Environmental Information

Environmental data differs from other types of data in many respects. Often the data are related to a certain location and time interval, or in other words, they describe environmental objects with a spatial and temporal extent. These objects, also known as geographic objects, possess a geometry consisting of point-form, linear, flat or solid features [14]. These geographic objects can also be associated with a set of non-geographic attributes, which constitute the thematic information related to the objects. Thematic environmental information may encompass such information as vegetation, weather conditions or the density of hazardous substances in the air or soil.

Strictly speaking, geographic objects with environmental features cannot be regarded as sufficient environmental information by themselves, because a simple datum can only be considered as useful information when it is accompanied by relevant expertise and criteria for assessment and decision making [4]. From the

information system point of view, there is a similar distinction between two groups of environmental information systems – those that are information systems in the strictest sense (used to store and retrieve data) and those that are geared toward analysis and simulation of the data [6]. The more the information system is to serve the purposes of analytical processing, the more challenging become the issues of the accessibility and quality of data as well as data heterogeneity problems. Beside the limited processing, memory and visualization capacity of handheld devices, these constitute the main constraining factor to the development of MEIS. However, problems related to data will not be discussed in more detail in this paper.

3 Research Design

For this study an explorative and constructive research was used. Constructive research answers in general to the following question: "Can we build a certain artifact and how useful is a particular artifact?" [8] Constructive research is then implemented with the following processes. Building is a process of constructing an artifact for a special purpose. Evaluation is a process of determining how well the artifact performs. Our construction task was to build a program, which enables mobile users to receive information from the botanical garden's plants and possible paths while in the premises. The main research question was: "What is the best way to build such a program?" From the implementation point of view there were two mobile platforms and solutions available:

a) Using a PDA with commercial mapping tools and an attached GPS-device.
b) Creating the program using a mobile phone and a separate GPS used over a Bluetooth or serial connection.

We describe some of the selection criteria in the Table 1. After making the decision to select the alternative b, Java enabled mobile phone as a development platform we formulated the main hypothesis: "Java implementation of the mobile phones J2ME with MIDP are not yet mature enough to develop all the necessary features for the required LBS prototype".

Table 1. Comparison of PDA and mobile phone alternatives

Feature/Device	PDAs	Mobile phones
Programming with Java	- Java usually not included as default	+ Java virtual machine with the device
Pilot	+ completed soon (with commercial tools)	- requires much more development resources
Source code (pilot)	- may include many restrictions	+ under developer's own control
Users of the device	- business users	+ more common within each possible target group

3.1 Method and Research Setting

Mobile platforms are in general difficult test subjects for conducting reliable and accurate measuring, as an example measuring response times. Results are highly situation dependent and cannot be compared very well. This is mostly because of technical reasons. With smart phones there are actually two different operating systems (OS) running concurrently. Native application running environment (or OS) like Symbian may sometimes be on top of the phone's own real time operating system. In addition to that in (Nokia's) smart phones, java applications run inside of an execution environment (JRE), which resides on top of a native operating system (Symbian in our case). The core phone functionality controlling software is usually a proprietary real time operating system that schedules operating environment for other functions, such as running application programs. Both native and java virtual machine based applications are also not deterministic in nature with smart phones. Also phone's core functionality cannot be independently shut down.

The MEIS software prototype was done as a pilot study to develop and check the real requirements of the system and to try out ideas in a realistic setting and to evaluate their success. The strategy applied started with a study of the current state of the MEIS domain. The first prototypes application ideas were decided next, in an attempt to loosely determine the main classes of problems and situations on which mobile devices can produce a positive impact.

The original design problems related to the MEIS prototype are described in the following questions:

1. Is it possible to do *system testing* for this prototype only with the phone emulator or do we need a real device?
2. Can the open source software (OSS) projects provide some help for developing the prototype solution with the mobile phone?
3. Do we need to use phone's native OS (e.g. Symbian) to create part of the functionality, and how the information is transferred between native and Java programs?
4. Which location/positioning technique to use and how?

4 Designing Multipurpose Mobile LBS Platform with MEIS

New emerging mobile platforms allow developing real time service platforms instead of older techniques like WAP 1.0 and simple J2ME/MIDP 1.0 programs that are unable to support complex application functionality in most mobile devices. Services that allow browsing of maps using WAP or MMS are often slow and not fully interactive. Available commercial personal navigation platforms for mobile devices are closed platforms and they are limited with their functionality.

The most difficult design choices are made when functionality is developed gradually to support all necessary LBS functions. In many cases there are requirements that have to be solved before more complex functionality can be implemented. For example, location dependent processing cannot really be specified if positioning is not yet supported. The main purpose of the experimentation was to

develop a mobile LBS platform that would support initial MEIS operations. This does not however exclude other uses for this platform.

4.1 Functional Requirement Based Alternatives for LBS in Mobile Platforms

Emulator vs. real device. Implementing the functionality introduced in the previous section requires sufficient knowledge from developing programs with various mobile platforms. Simple functionality containing programs for LBS purposes can be easily developed for mobile emulators running on desktop platforms (PCs). Such mobile emulators provide only some reliable support for mobile development and will fail in some important areas. Emulators should not, for example, be considered as accurate testing platforms. Even if an emulator resembles closely as an actual smart phone, it can have the most needed features for development missing entirely. There are also cases when the mobile platform does not yet support directly some features or has only partial functionality available.

Open source software (OSS) evaluation. There were no open sources or openly available LBS platforms available especially for the smart phones that we are aware of. For the desktop platforms (e.g. Windows and Linux) there are large open source code bases that are helpful for building such functionality, but we found those to be "difficultly" portable to mobile platforms. They use many features that exist in desktop environments but are missing from more limited platforms. Porting such code can be very hard even with clean room development, where no source code is actually required. Developing a feature rich LBS-program requires however the use of complex code. Much time and effort is needed, if such libraries containing necessary code are developed from scratch. Therefore a preferred method is to use various open source libraries, if possible.

When open source development or software will be used, then licensing needs to be also considered. Most commonly used open source license, Free Software Foundations GPL, requires all sources to be published if program is released and distributed somewhere. Also using LGPL (GNU Lesser General Public License) licensed libraries will require more extra work to be done since there are some rules to be followed.

For the commercial purposes open source development in general can therefore be somewhat counter productive since mobile platforms are functionally simpler than desktop environments. Therefore, when desktop based OSS is used in the development, it may require a lot of source code to be published beyond original intention. Additionally it is common for the mobile platforms that changes are made extensively for the code, and programs are more compact. LGPL rules do not take very well into concern the problems of mobile platforms and are more suited to software development in desktop environments.

5 The Description of Native and Java Implementation Alternatives

Until now we have developed a LBS platform prototype using J2ME and native Symbian OS C++. Experimentation required use of latest mobile devices available

and use of platforms supported by them. Series 60 provides a development platform that makes possible to do feature rich applications for mobile devices [1], which was chosen for the experimentation. Even while it is a somewhat limited with resources compared to desktop environments, it is possible to develop all the necessary application functionality into the programs of these smart phones. Series 60 is evaluated in the following paragraphs for developing a LBS platform in a small project. Appendix shows the Symbian OS based phones (at the time of the writing) and programming options for them. The next sections describe the development possibilities for the Series 60 mobile phones.

5.1 Symbian OS and Mobile Java Platforms

Series 60 is a Nokia Corporations platform that builds itself on top of Symbian OS version 6.1. It provides a special look and feel for smart phones implemented as libraries that is then licensed to third party mobile phone developers. Series 60 includes Java as J2ME and MIDP combination and is included to every smart phone firmware. Personal edition Java is implemented currently only to some high end Series 80 phones like Nokia 9200-series Communicators and to some UIQ platforms. Series 80 includes MIDP only as an optional package that is still in beta testing phase. The biggest difficulty from the application's portability point of view is that each mobile phone has its own implementation from the MIDP virtual machine.

5.1.1 Native Symbian OS

Symbian OS is being developed for mobile devices by the Symbian coalition. It uses its native C++ as the main development language and is a real object-oriented operating system by its design. Programs are developed modeling their event driven characteristics using active objects and asynchronous calls to different system services. This requires handling of lot of state machines and knowing response states from other parts of the system [7].

Program development for native Symbian requires many things to be learned from the Symbian perspective. Simply porting programs from other platforms can not be considered. For the native programs Symbian OS provides best interfaces available in the smart phones for the phone's resources. Documentation to these interfaces has not been previously available, because the platforms have not yet really matured. Recently a lot of examples from the application programming interface (API) usage are provided and textbooks are available for the most common tasks. Still, developing full featured applications using native language can require a lot of resources from small projects. Based on our experiences we claim that it is better to use a combination of native and J2ME or Personal Java platforms when developing mobile applications than only native platform alone. Section 6.1 explains more of this development method.

5.1.2 Personal Edition Java and Personal Profile

Personal edition Java is being developed by Sun Microsystems. It uses a connected device configuration (CDC) and requires a fully featured Java virtual machine (JVM) [1]. It is currently implemented to Sony-Ericsson P800 and Nokia 9200-series Communicators. Compared to other smart phones they are more business oriented and

are called as wireless information devices (WIDS). Personal Java is programmed using Java Development Kit 1.1 (JDK 1.1) and it is different from the preferable standard edition's JDK 1.2 of Java. Personal Java has since it was developed been replaced entirely with personal profile. Java virtual machines have somewhat limited access to platform-specific resources. For avoiding any specification implementation problems that may exist, Java Native Interface (JNI) has also been implemented for it. Allin [1] tells that there are some limitations for its use in Symbian OS, for example exception handling is quite different and can cause JNI functions to be something between of actual Java and native Symbian methods.

5.1.3 J2ME with Symbian OS

Java 2 Micro Edition (J2ME) uses a connected limited device configuration (CLDC). It has been aggressively optimized and scaled back for a VM (Virtual Machine) implementation in order to fit on small device like mobile phones [1, 11]. Symbian's goal is to use this VM to create a Java environment which then provides a richer, more open platform that is capable of delivering fully functional applications and services [1]. Also according to Symbian such an environment can be built over the CDC, but is harder than on the CLDC and kilo virtual machine (KVM) [1].

For Series 60 smart phones like Nokia 3650, which was mainly used for the prototype, there are a lot of advantages compared to the use of many other mobile phones. There are no strict heap sizes or installation memory size limits other than the general smart phone requirements. Also different versions from the applications can be installed, when needed. Testing on real mobile device is important since some of the phone's features, that are also mentioned here, may not be well known beforehand. Manufacturers may also provide proprietary java extensions for the user interface development and audio support. Usage of proprietary extensions ties development to a certain manufacturer's mobile device. Those should be considered only as a last phase of the development if they are absolutely required.

6 LBS Considerations for Positioning and Hardware

Triangulations based systems use fixed positions to measure and calculate the location. There are several existing services that use this technology for mobile phones. They use cellular base stations as fixed points. This kind positioning can achieve precision from 50-100 m in city area and couple of kilometers in sparsely populated areas where the cellular base stations are more far apart. Also WLAN and Bluetooth technologies can be used for triangulations. Studies are made to make positioning for these new technologies so precise that it would be less than a centimeter. Also with new technologies like in 3G networks the precision for cellular positioning can be quite accurate like 10 m in city area.

GPS (global positioning service) uses satellites to point the position. The accuracy of the result depend how many satellites are used for calculations. In normal use calculations are made from 6 satellites. With this technique the precision will be close to 10 m.

Hybrid positioning systems combines different technologies for positioning. They can for example combine triangulations technology indoors and in city area and use

GPS when in sparsely populated areas. The most common location technology options for the mobile handsets are described in Yin [23] and Varshney [20].

LBS may require the support of several concurrently working positioning methods. There are several *cell based positioning* techniques, but they are mostly limited to network operator's own use. Furthermore they may not work in every country and area. *GPS positioning* was the only possibility that was somewhat usable from the prototypes required accuracy point of view for our real time prototype. It has a wide coverage and works fine outside in the nature. It has some problems including less operational functionality in northern parts of hemisphere and in problematic terrains, where there are obstacles to the paths of radio waves. Some mobile phone manufacturers rely on concurrent usage of both previous positioning techniques and have named it assisted GPS (AGPS). Wireless local area network (WLAN/WiFi) and Bluetooth may also be applicable for the future positioning requirements.

6.1 Software Consideration for LBS in Mobile Java

New J2ME is promised to include a special Location API that will combine several positioning possibilities [12]. It requires a new java virtual machine conforming to CLDC 1.1 that includes primitive types for decimals and new floating point operations [18]. Current smart phones in 2003 include only CDLC 1.0 virtual machines. MIDP 2.0 improves J2ME platform by adding Bluetooth connections as an optional feature. Bluetooth can then be used to communicate with a GPS device, if there is no integrated GPS already present. MIDP 2.0 requires less than CLDC 1.1 to be included to smart phones from resource requirements perspective, so CLDC 1.1 is likely to be implemented some time later.

New mobile applications that need positioning are likely to also use or require floating point calculations. For example when data is transmitted from a GPS device, using NMEA messages to outside world, decimal values such as decimal latitude and longitude are used. Most of the mobile phone applications can manage without using decimals and operations altogether [5]. Therefore hardware acceleration for floating point operations is not currently implemented to CLDC 1.0 supported smart phones. CLDC 1.1 will support hardware accelerated floating point operations so the use of the software-based libraries will no longer be required [18].

Java interpreted functions can usually be extended by using native programs as was explained with CDC. J2ME does not provide any JNI like interface [1], The only possibility is to use inter process calling (IPC), as in our MEIS prototype, between two programs implemented with Java and Symbian C++. This requires using an undocumented network connection to another program that is a native language implementation. This is possible because both programs are being run at the same time in a multitasking environment. Introducing this kind of combined use of the platforms was conceived very early in MINNE project and no examples or previous use of this was known from anywhere else.

These IPC connection types are undocumented, and they are not fully working implementations and can work unexpectedly. Such an effect happens when the mobile phone suddenly stops responding. Resetting by removing batteries is not uncommon when the prototype was tested. The new CLDC 1.1 will include also a special k-virtual native interface (KNI) that makes native extension implementation possible only for virtual machine developers [18]. It is not usable for the application

developers working with J2ME. Figure 2 summarizes all of these technical considerations discussed above.

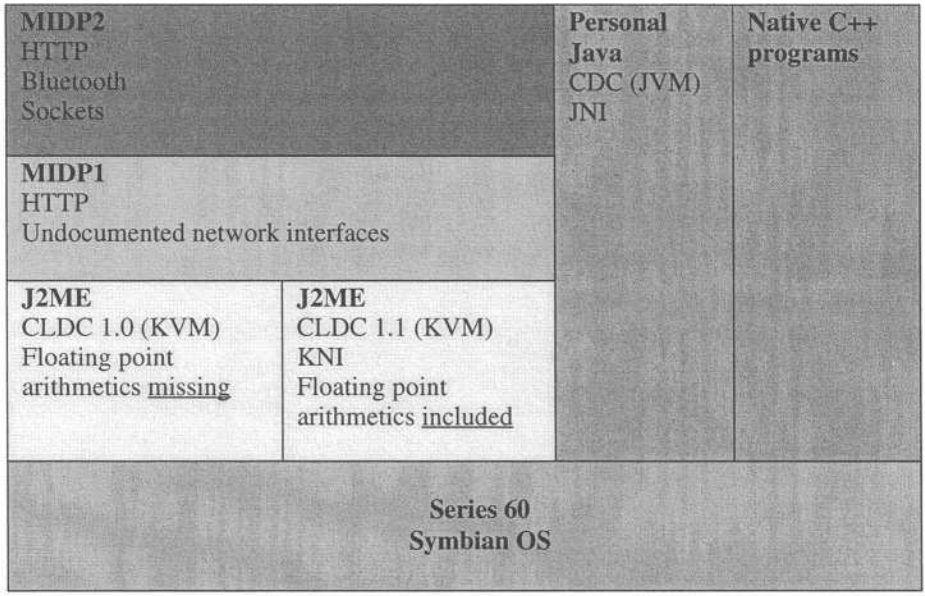


Fig. 2. Software platforms generally available in Symbian based mobile phones

7 Implementation of the Prototype

Implementation of the MEIS prototype was done using evolutionary steps bringing new features gradually to already implemented and tested parts. Mathematical library for floating point operations was the first step, since CLDC 1.0 without the float data type was the only available possibility. Using maps required development of the own software libraries for different kinds of map projection calculations to be implemented. LBS content browsing was done next, taking into concern possibilities of mobile device user interfaces and controls. For example the same directional keys were used differently depending on prototype's states and no touch screen functionality for a selection mechanism is expected. Table 2 presents the work effort during one year's prototype development (about 1 man year).

Table 2. The evolutionary steps of the MEIS prototype development

2002			2003								
10	11	12	1	2	3	4	5	6	7	8	9
Research of the OSS mathematics and map projection libraries			Float conversions (J2ME) and LBS map projections, prototype's Java browser in MIDP 2 (with emulator)			Transfer of the prototype to Nokia 3650's MIDP 1.0		Wireless network functionality: image transfer over the GPRS		Adding of the GPS features	

Implementation of the extendable protocol for exchanging messages over the wireless network was also needed. Prototype's current implementation is closer to protocols like SMTP and HTTP than XML, which is the most favored communication method this type of applications. By today, only static content from the MEIS sites was tested without using a database. Additional MEIS features are now easy to implement as new LBS services. Positioning is done using GPS over Bluetooth, which required implementation of the own native Symbian C++ program for the delivery of location coordinates to the java browser program. Figure 3 shows an example image from the MEIS prototype's browser.

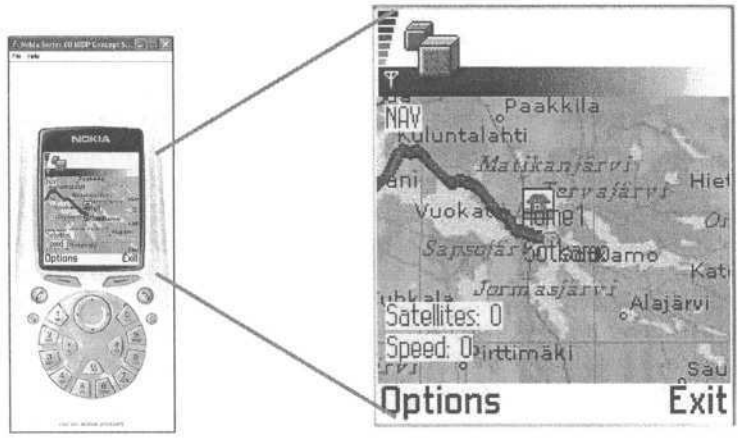


Fig. 3. An example display from the MEIS prototype (on the Nokia 3650's J2ME emulator)

7.1 Extending Architecture beyond the Browser

With the MEIS prototype, simple static content for LBS sites is accessed using current HTTP infrastructure. It is already possible to create LBS services, which would provide different scale maps and some information about the environmentally interesting location points visible on them. The user can select a location point and then be directed for seeing, for example, location dependent plant information. Network services and servers can support more complex functionality later on.

7.2 A Preliminary Evaluation of the First MEIS Prototype

The process of prototype development can be seen from Figure 4. Until now, the evaluation of the prototype (the final stage) is based on the experimentations of the project team. Further evaluations are planned in the near future with some potential end users.

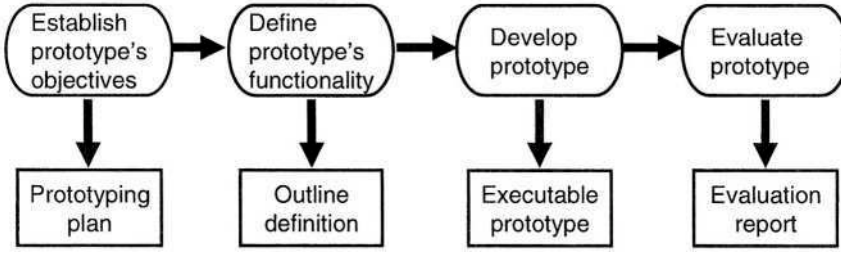


Fig. 4. Prototyping process [16]

Here are some answers to our original design problems:

- Based upon this prototype, it can be said that some features need to be developed with the native OS (Symbian) for example GPS-data transfer over the Bluetooth to the phone. Often, during many steps of the development, testing was possible only with the real device.
- OSS projects proved to be important "tools" for the prototype development. Without examples from them it wouldn't have been possible to create many LBS features for the MEIS prototype within one year's time (e.g. some calculations like map coordinate transformations).
- Programming requires detailed information (not presented in books) of how things should be done, for example handling strings correctly between both Java and Symbian platforms.
- Today's restricted Java environments of the mobile phones may lack some important characteristics, which are available for many other platforms. The best example was the absence of the floating point arithmetics. Its availability would have simplified the realization of various LBS map projections.
- At this stage GPS was found to be the most suitable option as a positioning technique because of the nature of our planned MEIS prototypes. Those will be mainly used outside in the nature - not in densely populated areas or in buildings where other positioning techniques are required. Connecting external hardware to the mobile phones over a Bluetooth seems to be the current trend, because only few models like Nokia 9210 support serial connection.

8 Conclusions

We have shown that it is possible to create a usable artifact from the MEIS service with interesting results. Also with the limited capabilities of today's mobile phones we could collect ideas and experiences from the application domains, which are forecasted to be growing fields of tomorrow. The biggest obstacle of our prototype's portability to other mobile phone models may result mainly from today's different implementations of the Java virtual machines (proprietary solutions for the each model) by the phone manufacturers. Anyway the first prototype has already produced a really interesting concept of a browser based mobile MEIS solution, which will be used for further evaluations in the mobile service development. Further work for

proving the MEIS concepts usefulness needs to be done, for instance, because of the following reason: to prove out our solutions possibilities and benefits while also dynamic, up to date environmental information is used.

References

1. Allin, J.: Wireless Java for Symbian Devices. John Wiley & Sons (2001)
2. Antikainen, H., Bendas, D., Marjoniemi, K., Myllyaho, M., Oivo, M., Colpaert, A., Jaako, N., Kuvaja, P., Laine, K., Rusanen, J., Saari, E., Similä, J.: Mobile Environmental Information Systems. Systems Analysis Modelling and Simulation (2003) to be published
3. Beaulieu, M., Cooper, M.: Wireless Internet Applications and Architecture. Addison-Wesley (2001)
4. Environmental Information Policy and Utilization of Information Technology: Towards a Shift in Environmental Policy Paradigm. Development Bank of Japan Research Report No. 25. Development Bank of Japan, Economic and Industrial Research Department (2002)
5. Graeme, A.: The Shrinking Game: Feature. 3D world. Volume 2003, Issue 43. October 2003. Future publishing (2003)
6. Haklay, M.: From Environmental Information Systems to Environmental Informatics – Evolution and Meaning. Centre for Advanced Spatial Analysis, Working Paper 7 (1999)
7. Jipping, M.: Symbian OS Communications Programming. John Wiley & Sons (2002)
8. Järvinen. Pertti.: On research methods. Opinpaja, Tampere (2001)
9. Katz, R. H. Adaptation and mobility in wireless information systems. IEEE Personal Communications Magazine 1 (1994) 6-17.
10. Maguire, D.: Mobile Geographic Services Come of Age: ESRI Dives into Wireless Markets. GeoInformatics 4, March (2001) 6–9
11. Muchow, J.: Core J2ME. Technology & MIDP. The Sun Microsystems Press (2002)
12. Nokia Corporation.: Location API for Java 2 Micro Edition. Version 1.0. 2003. JSR-179 Expert Group. Java Community Process (2003)
13. Nunamaker, J. Jr., Chen, M., Purdin, T. D. M.: Systems Development in Information Systems Research. Journal of Management Information Systems, Volume 7 Number 3 Winter (1991) 89-106
14. Riekert, W.-F.: Management of Data and Services for Environmental Applications. In: P. Stancikova and I. Dahlberg (eds.): Environmental Knowledge Organization and Information Management. Proceedings of the First European ISKO Conference. Bratislava, Slovakia, September 14-16, 1994. Indeks Verlag, Frankfurt/Main (1994)
15. Smith, J., Kealy, A. and Williamson I.P.: Location Based Services - The Underlying Technology. The Third International Symposium on Mobile Mapping Technology, January 3-5, Cairo, Egypt (2001)
16. Sommerville, I.: Software Engineering (5th ed.). Addison-Wesley (1996)
17. Spinney, J., Mabrouk, M.: Using J2ME to Deliver Spatial Information to Mobile Clients. Presented at JavaOne 2002, March 25-29, San Francisco, California (2002)
18. Tasker, M.: Professional Symbian Programming. Mobile solutions on the EPOC platform. Wrox Press (2003)
19. Uhlirz, S.: Cartographic Concepts for UMTS-Location based Services. In: Proceedings of the 3rd Workshop on Mobile Mapping Technology, January 3-5, Cairo, Egypt (2001)
20. Varshney, U.: Addressing Location Issues in Mobile Commerce. Local Computer Networks. Proceedings of the 26th Annual IEEE Conference on Local Computer Networks, November 14-16, 2001, Tampa, Florida, USA (2001) 184–192

21. Veijalainen, J., Virrantaus, K., Markkula, J., Vagan T. Developing GIS-Supported Location-Based Services. In Ozsu, T., Schek, H.-J. and Tanaka K. (eds.): Proceedings of the 2nd International Conference on WEB Information Systems Engineering (WISE 2001), 3-6 Dec. 2001, Kyoto. Kyoto University (2001) 423-432
22. Visser, U., Stuckenschmidt, H., Wache, H., Vgele, T.: Using Environmental Information Efficiently: Sharing Data and Knowledge from Heterogeneous Sources. Environmental Information Systems in Industry and Public Administration. C. Rautenstrauch, S. Patig. Hershey, PA, IDEA Group (2001) 41-74
23. Yin, H.: Location Based Service - T-109.551 Research Seminar on Telecommunications Business II, March 26th 2002, Helsinki University of Technology (2002)

Appendix: Symbian Based Phones, Available or Announced

Manufactured by	Model	Symbian	UI Platform	J2ME	CLDC	Personal Java	WAP
BenQ	P30	7.0	UIQ 2.1	MIDP 2.0	1.0		2.0
Ericsson	R380	EPOC 5.1(u)	Quartz	-			
Fujitsu	F2051	6.1	Fujitsu	i-Java, DoJa	1.0		
Fujitsu	F2102V	6.1	Fujitsu	i-Java, DoJa	1.0		
Motorola	A920	7.0	UIQ 2.0	MIDP 1.0.3		1.1.1a	2.0
Nokia	9210	6.0	Crystal	MIDP 1.0	1.0	1.1.1	1.1
Nokia	9290	6.0	Crystal	MIDP 1.0	1.0	1.1.1	1.1
Nokia	7650	6.1	Series 60/0.9	MIDP 1.0	1.0		1.2.1
Nokia	3650	6.1	Series 60/0.9	MIDP 1.0	1.0		2.0
Nokia	N-Gage	6.1	Series 60/0.9	MIDP 1.0	1.0		2.0
Nokia	6600	7.0s	Series 60/2.0	MIDP 2.0	1.0		2.0
Samsung	SGH-D700	6.1s	Series 60/1.1	MIDP 1.0	1.0		2.0
Sendo	X	6.1	Series 60	MIDP 1.0	1.0		2.0
Siemens	SX1	6.1	Series 60	MIDP 1.0	1.0		2.0
Sony Ericsson	P800	7.0	UIQ 2.0	MIDP 1.0	1.0	1.1.1	2.0
Sony Ericsson	P900	7.0s	UIQ 2.1	MIDP 2.0	1.0	1.1.1	2.0

From http://213.190.42.244/whisper/space/Symbian+Phones, checked Oct. 28th, 2003.

Selecting CMMI Appraisal Classes Based on Maturity and Openness

Stig Larsson[1] and Fredrik Ekdahl[2]

[1] Mälardalen University, Department of Computer Engineering, Västerås, Sweden
stig.larsson@mdh.se
[2] ABB, Corporate Research, Västerås, Sweden
fredrik.ekdahl@se.abb.com

Abstract. Over the last eight years, different approaches have been used to diagnose the performance in ABB organizations developing software. The efforts build to a large degree on methods from the Software Engineering Institute (SEI). In this paper we examine the experiences from five organizations through a description of the pathways that we have observed in the maturity development. We also propose a way to classify organizations based on two organizational characteristics, maturity and openness. Based on this classification, a simple method for the selection of how to collect performance data from the organizations is described.

1 Introduction

Considerable effort has been put into transforming ABB into an organization recognized for its software development excellence. This is in line with a strategic redirection of operations towards primarily the software intensive process automation market. Since the mid-nineties, several performance improvement initiatives have been run on a national level. Tangible results are evident in many of the participating organizations. Today, there is a global program for performance improvement in product development that coordinates improvement activities throughout ABB.

Using structured process improvement methods is a well-documented path towards increased maturity in product development organizations. In this paper, we adhere to the Software Engineering Institute's [1] definition of maturity as an organization's ability to consistently follow and improve its processes. In ABB, we have over the last eight years tried a number of approaches in different parts of the organization. Our focus has been on software development units, primarily developing software intensive products. Due to organizational dynamics and management short-term focus, some of the initiatives have been disrupted or slowed down. Also, the expected development towards higher maturity and accompanying results in quality and development speed has not been received. As organizations probably will continue to be dynamic and the focus on short-term results occasionally will re-appear, we need to better understand how performance improvement can be achieved in spite of these and similar circumstances.

Based on our experiences in ABB we introduce the concept of evolutionary paths in maturity and use five short case studies from organizations within ABB to illustrate

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 457–470, 2004.

how organizations develop over time in terms of maturity. As a structure for the illustration of how these organizations have evolved, we use a simple model that allows a longitudinal perspective. The model defines four different types of organizations, each exhibiting a few unique characteristics. Based on our experiences within ABB, we also propose strategies for approaching each of the four types of organizations. Future work will include more comprehensive evaluations of different approaches for the different organizational types. This will eventually lead to the possibility to provide strong guidance for all types of organizations relative to diagnostic strategies and to improve the maturity in software product development.

In ABB, the Capability Maturity Model Integration(CMMI® [1][2]) is used as the preferred process reference model and that is reflected in this paper. However, we would like to point out that the proposed classification and the corresponding strategies are valid independently of the reference model used.

The CMMI (and its predecessor the Software Capability Maturity Model, SW-CMM) was developed to answer the need for more structured and long lasting improvement of software product development organizations. Both the SW-CMM and the CMMI are derived from extensive industry experience.

The CMMI consists of five maturity levels, each representing an evolutionary stage that organizations pass through as they increase in maturity. Each maturity level consists of a set of carefully selected Process Areas of relevance to the specific evolutionary stage. This way, the levels provide an implicit prioritization of which processes to address during each evolutionary stage. Each Process Area consists of a set of Goals and a set of corresponding Practices.

The CMMI is a process reference model and it does not contain any explicit support for how to actually achieve improvement. Therefore the SEI developed the IDEAL model [3], which in detail describes how to use the CMMI (or in fact the SW-CMM) to professionally improve the maturity of an organization.

The IDEAL Model consists of five phases; Initiate, Diagnose, Establish, Act and Leverage, each serving a specific purpose in an ongoing improvement effort.

The purpose of the Diagnose phase is to baseline current level of maturity against a selected reference model, such as the CMMI. This includes planning, execution and follow-up of appropriate appraisal activities. For other reference models the terms audits or assessments are often used, which corresponds to the term appraisal used here. Findings from the appraisal activities are used as a basis for identifying appropriate improvement actions that are then the focus of the Establish, Act and Leverage phases.

The remainder of this paper is organized as follows. Section two describes how improvements in product development maturity can be based on appraisals. We also define the characteristics selected for classification of organizations. Section three contains the five case studies that illustrate the evolutionary pathways that we then use as a basis for the different diagnostic strategies detailed in section four. The paper ends with some conclusions and a brief look into possibilities for future work.

2 Appraisal-Driven Improvement

There is strong support in the literature that conducting diagnostics activities, i.e. systematically identifying strengths and weaknesses in an organization, contributes to

the advancement of organizational excellence [5]. Diagnostic activities come in different forms, including for example appraisals, audits, assessments and reviews. Common to all are that the results can be used as the foundation for future improvement activities.

In ABB, the preferred diagnostics methodology is CMMI appraisals. An appraisal is an examination of one or more processes that an organization does to and for itself for the purposes of process improvement. It is conducted by a trained team of professionals using an appraisal reference model as the basis for determining strengths and weaknesses [1]. The Appraisal Requirements for CMMI (ARC 1.1) [4] defines three classes of appraisals (Class A, B and C). All three classes are used in ABB and they all display different strengths and weaknesses [6]. Class A appraisals are the most comprehensive, but require substantial resources and may be considered very intrusive by the organization being appraised. Class B appraisals are less comprehensive and consequently less intrusive, but still require considerable resources. Finally, Class C appraisals are the least comprehensive, but again require fewer resources and are less intrusive. The comprehensiveness of the appraisals of course influences the reliability and validity of the appraisal results. Table 1 summarizes the characteristics of the different classes of appraisals.

Table 1. Appraisal class characteristics

Class of appraisal	**A**	**B**	**C**
Size of appraisal team	8-10	3-4	1-2
Appraisal time	10 days	3-4 days	1-2 days
Minimum # of data collection methods	3	2	1
On-site interview required	Yes	Yes	No
Cost	High	Medium	Low
Intrusiveness	High	Medium	Low
Validity	High	High	Low
Reliability	High	Medium	Low

As appraisal findings are fundamental to the subsequent activities it is of utmost importance that they show high reliability and validity. In this context, reliability represents the ability to produce findings that are relevant irrespective of variations in sources of data and validity represents the ability to pinpoint the most relevant findings.

There are several organizational characteristics that improve the chances of successful improvement activities. [7] provides an overview of characteristics of particular importance for software process improvement based on case studies and experience reports from 56 different organizations.

Similarly, there are several organizational characteristics that contribute to truly effective appraisal activities. [8] contains a good overview of what is needed from an organization to fully benefit from an appraisal. Among the more important can be mentioned:

Strong management commitment: Improvement does not happen overnight. It is important the senior management is willing and patient enough to visibly stand behind a genuine improvement effort based on the appraisal results. This also includes allowing and withstanding the scrutiny that appraisal activities entail.
Resources for Performance Improvement: Financial as well as human resources are necessary to cover the appraisal costs and of course also to fund subsequent improvement activities.
Improvement Infrastructure: In order to effectively make use of appraisal results, a certain degree of organizational structure must be available. It is for example common to establish an engineering process group that is responsible for coordinating the improvement activities. Also a process champion that reports directly to the management team on improvement progress is very valuable.

Organizational characteristics such as these are fairly easily identified when observing an organization from the outside. Consequently, they can be used as indicators of the readiness of an organization to work effectively with appraisal-driven and professional improvement.

In our contacts with development units within ABB we have identified two characteristics that distinguish the organizations from each other relative to the effectiveness of different classes of appraisals. The first characteristic is *Maturity,* i.e. the ability of an organization to follow and improve its processes. We have found that the maturity of an organization influences the possibilities to conduct appraisals effectively, as well as the ability to appreciate and benefit from the results from an appraisal. The concept of maturity according to the SEI's definition can be considered fairly objective as there are established techniques and considerable experience in the field of maturity evaluation.

The second characteristic is *Openness,* which basically captures the inclination of an organization to embrace external help. In this paper, openness is an aggregate that represents the willingness of an organization to accept the costs of an appraisal, to accept the inconvenience of external examination and to make a genuine effort to improve. This definition of openness is influenced by the list of organizational characteristics identified in [8]. A proposed scale, ranging from 1 to 5, for the openness of an organization:

5. Management and organization are openly requesting assistance in performance improvement, including appraisals, improvement planning and subject matter expertise.
4. Management and organization accept assistance in performance improvement, including appraisals and improvement planning.
3. Management accepts assistance in conducting appraisals.
2. Management accepts discussing performance improvement.
1. No access to management or organization.

We acknowledge the dangers of using a characteristic with such obvious subjectivity. However, we still argue its relevance for guiding appraisal activities, as will be illustrated in the case studies and in our recommendations.

There are of course many organizational characteristics beside maturity and openness that also influence the potential benefit and success of appraisal activities. For example, the size or complexity of he organization, the current lifecycle phase of

key projects, or current financial status. We do not claim to be in any way exhaustive in our search for suitable organizational characteristics. Instead, we have settled with a set of characteristics that appears good enough for our purposes.

3 Case Studies

The case studies include five different ABB organizations. The size of the organizations, in number of software developers, range from 60 to 120 and all five organizations develop software that is used in industrial environments. We have refrained from revealing actual maturity levels in the case studies, primarily because they add no value to the discussion or the conclusions of this paper, but also as they are considered confidential and proprietary to the individual organizations. Note that the overall goal of the case organizations is not to achieve a certain maturity level, but to improve performance.

3.1 Research Method

Each organization has been examined over a period of several years with different data collection methods. *Appraisal and assessments* have been made using the SW-CMM or the CMMI as a reference model. Typically, class B appraisals or equivalent assessments have been made. *Support work* ranges from leading workshops and providing training to assisting the organizations in developing and institutionalizing new processes. Through moderating *peer reviews* where software development managers meet to review each other's improvement activities, additional information has been collected. Finally, *networks* of software development managers have been organized. Both authors of this paper have been directly involved in all of the above activities. This allows us to estimate maturity and openness. Maturity estimates have been based on data available from appraisals, assessments and audits. Openness has been estimated based on the scale presented above.

The long-term observations have given us the possibility to describe a maturity development path for each organization. The extended time for the observations as well as the use of the diversity of the data collection methods should increases the reliability of the collected data.

Through the methods used in this study, we conclude that this is a hermeneutic research endeavor. Thus, we need to acknowledge the influence of our research on the units and the results. We claim that although we influence the different pathways, this influence does not change the validity of the observations or the conclusions regarding the classification of the different organizations.

We acknowledge the relatively unstructured data collection method used when determining the openness. This is also true for some of the maturity observations. The data collection is thus partly subjective. As a result, the possibility to replicate this particular study is small. However, we consider our observations as a good starting point for conducting a more stringent investigation.

In addition, through the long-term observations and extensive experience from the organizations, we still claim that the observations are reliable enough to qualitatively determine the openness and maturity for the organizations. In addition to this, the use of external observers doing peer review of our research adds validity.

3.2 Organization A

Organization A develops real time control systems and tools for adaptation of the system for different applications. The unit has developed software for almost 30 years, but has been exposed to repeated changes in the organizational structure over the last 15 years.

Figure 1 shows the development of the maturity and openness over the last eight years.

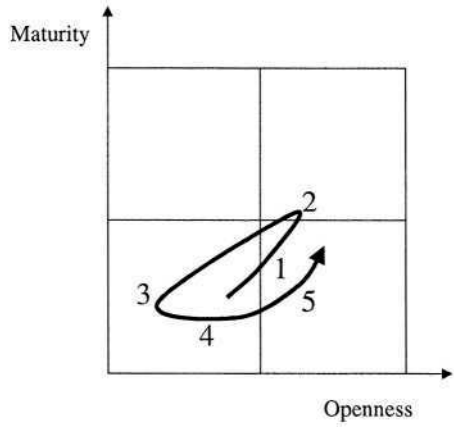


Fig. 1. Organization A maturity pathway

The following development has been observed:

1. *Use of CMM to improve performance.* The decision to base the improvement efforts on SW-CMM was firmly committed with the management. During this period, the performance improved as well as the maturity. Also, the need to involve external experts for specific areas was acknowledged.
2. *Major change in commitment.* Due to acquisitions and new development management, the commitment to increase maturity as a means to further increase performance was lost. Consequently, the openness diminished.
3. *Organizational changes.* As a result of the insufficient results, clarification of global responsibilities where made, again giving the responsibility to the local management to drive improvement.
4. *High pressure to deliver.* Improvement efforts were in this period not prioritized. However, the openness increased as a result of local management commitment to improvement efforts.
5. *Initial results from process improvement activities.* Through the commitment and openness, we can now observe initial results in maturity.

3.3 Organization B

With the experience of software development spanning over more than 20 years, this organization has established the basic routines for product development. The real time part of the system is primarily operating as an independent product, but is more and more connected to a larger system. The unit is also expanding the software development to PC based products. This expansion has partly been made through acquisitions.

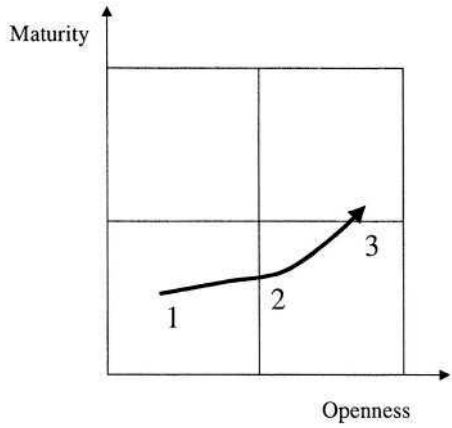


Fig. 2. Organization B maturity pathway

The development for the last five years for organization B is shown in figure 2:

1. *Yearly improvement plans introduced.* The efforts to improve were based on targets for the organizations. However, no diagnostic activity was performed.
2. *Increased pressure to Improve timeliness and quality.* The pressure lead to increased acceptance to involve external resources in finding improvement areas.
3. *Introduction of CMMI.* Recently the organization has decided to use CMMI as a tool for identifying weakness and initiating improvements in projects.

3.4 Organization C

Through several reorganizations and mergers, this organization has managed to maintain focus on the products that are entirely software based. The product development is primarily directed towards the evolution of the product platform. The organization is also cooperating with similar units around the world. However, there is currently no requirement that the products from different units should be integrated.

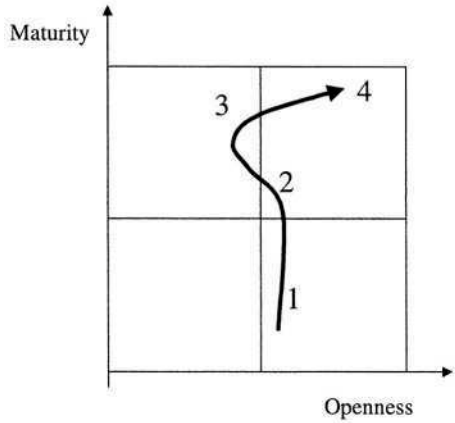


Fig. 3. Organization C maturity pathway

The pathway shown in figure 3 describes the development for the last three years:

1. *Strategic decision to use SW-CMM.* The decision was a response to market requirements and included the intention to use SW-CMM as a diagnostic tool as well as a roadmap for improvements. External expertise was requested from start.
2. *Initial results achieved.* As results where observed, the organization started to work more in isolation.
3. *Acceptance that external assistance is beneficial.* As the pace of the improvements slowed down, a more open attitude could be observed.
4. *Preparation for class A appraisal in progress.* The current status is that a class A appraisal is planned.

3.5 Organization D

The organization develops products with a tight integration of hardware and software. The real time requirements on the system are tough and also one discriminating factor in the marketplace. The development of software has gradually grown over the last 15 years.

As shown in figure 4, the organization has developed in the last five years as follows:

1. *SW-CMM used for improvement.* Through a Class A type of assessment, the organization started an improvement effort.
2. *Internal improvements.* As the organization was maturing, the strategy was to decrease external involvement in the efforts to diagnose and improve performance.

3. *Organizational changes.* The organization was transferred to belong to a different part of ABB. This resulted in a change in senior management with less interest for improvement efforts.
4. *Change agent changes.* In addition to organizational changes, the unit has had several changes in the staff responsible for processes and improvement efforts.

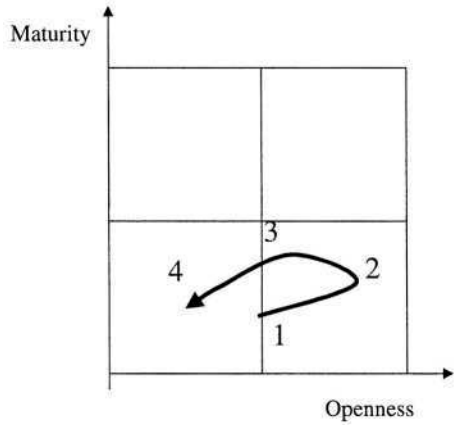


Fig. 4. Organization D maturity pathway

3.6 Organization E

The organization develops real time products with a tight integration of hardware and software. The reliability requirements on the products are high. The dependence on the software part of the product has steadily grown over the last 25 years.

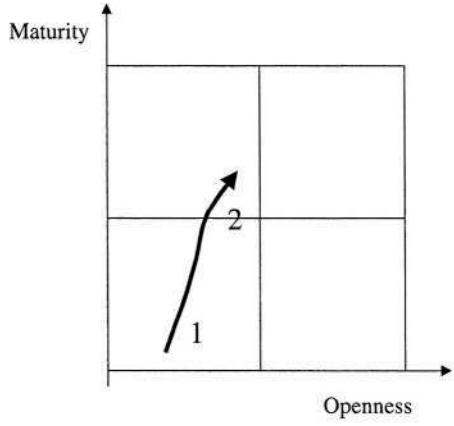


Fig. 5. Organization E maturity pathway

Figure 5 shows how the organization has developed over the last eight years:

1. *Increased demands on performance.* As the products started to include more and more software, the structure and complexity grew. This led to the needs to improve. The organization started to gradually improve through internal projects.
2. *External support.* As the organization matured, an appreciation of external assistance started to grow.

3.7 Longitudinal Perspective

The development of maturity over time may also add to the understanding of how to choose a diagnostic strategy. Figure 6 shows this development for the development organizations in the case studies.

We draw three additional conclusions from our experiences; to build up maturity takes considerable time, to build up the confidence for external support also takes time and finally, both maturity and confidence can easily and quickly be lost.

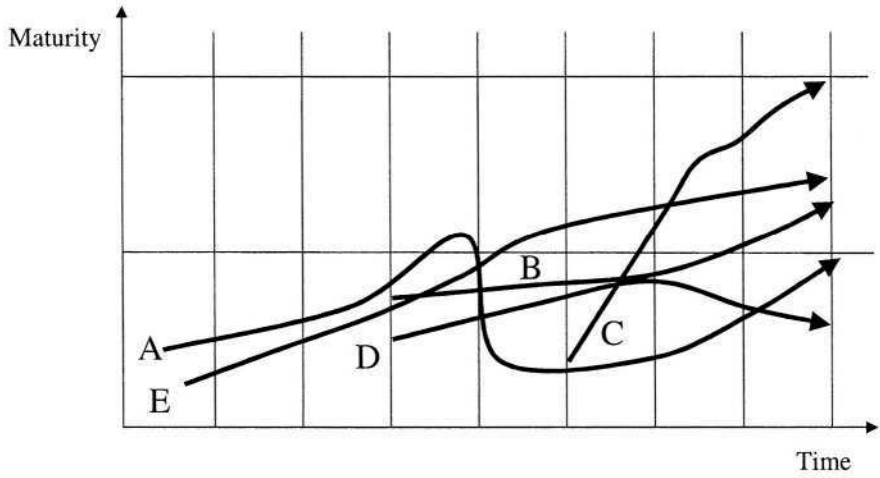


Fig. 6. Longitudinal maturity development for organizations A through E

4 Diagnostic Strategies

Diagnostic activities are key to the overall success of an improvement activity. Consequently, choosing an appropriate diagnostic strategy largely determines how effective an improvement effort will be. In an attempt to capture the experiences we have made in ABB and to allow some amount of generalization we propose classifying organizations based on their openness and maturity. In addition we propose a primary diagnostics strategy for each type of organizations.

4.1 Organizational Classification

From the observed case studies, we have identified four types of organizations as shown in Figure 7. Type 1 organizations are immature organizations that are unwilling to let external experts help in improvements. Typically these organizations think they are more mature than is the case.

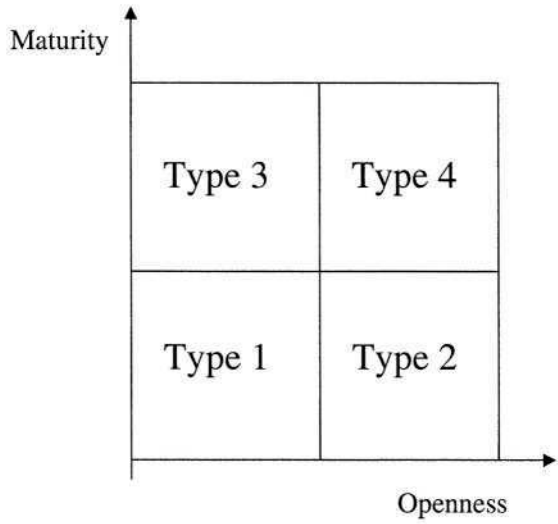


Fig. 7. Organization classification based on openness and maturity

In Type 2 organizations, where openness is found but the maturity is low, an acceptance that the organization is immature is often found. There is often awareness in this type of organization that external assistance is needed.

Type 3 organizations very often have a tradition of internal process improvement that has led to a mature status. The lack of openness in these organizations can have several different reasons, but as it may slow down or even stop the improvement activities, efforts should be made to overcome it.

Finally, Type 4 organizations take full advantage of the external expertise and use that to maintain their maturity.

4.2 Recommended Diagnostic Activities

Organizational openness and maturity are both relatively easy to observe. They are also highly indicative of the kind of external support and organization is ready for. Consequently, the openness and maturity of an organization can be used to guide decision making in the development of an improvement strategy. This is especially true when choosing appropriate diagnostic activities.

Figure 8 illustrates the recommended diagnostic methods, expressed in different classes of CMMI appraisals, for each of the four types of organizations. Our

recommendation is based on a combination of the observations made in the case studies and other organizations within ABB and externally.

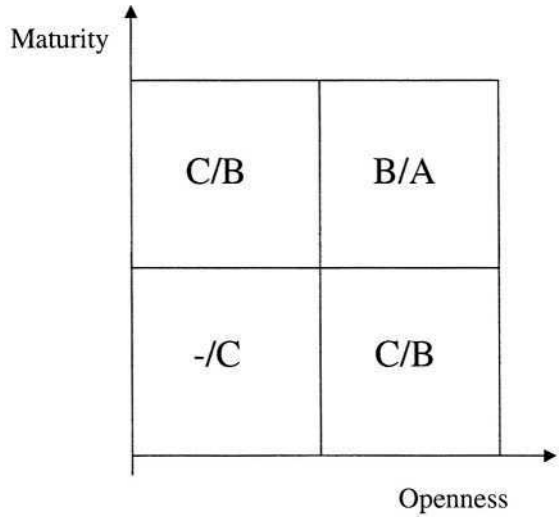


Fig. 8. Selection of CMMI Appraisal Class (A through C) based on organization characteristics

The basic observation is that organizations with low maturity need experience in process improvement before exhaustive appraisals give the expected benefits. In addition, organizations that are sensitive to intrusive appraisal methods need confirmation that external assistance in finding improvement opportunities is useful.

For organizations with low maturity that are sensitive to external appraisals (type 1 organizations), very basic processes may have to be put in place before any appraisal is helpful. As the organization is immature, the area to improve should be easy to find through initial discussions with the organization. It is very important that the efforts give quick payback, as this will encourage the organization to continue the efforts. As the organization gains experience and the first appraisal is conducted, the unit will get an understanding of what the result of process improvement activities can be. The organization will also start to appreciate the view from external sources. This type of organizations typically selects Class C appraisals. Since a class C appraisal only covers a part of the organization, the selection of projects is important. Our experience is that central projects should be selected. Based on the status of these projects, the process areas to be examined are decided.

If an immature organization is open to external assistance in finding improvement opportunities (type 2 organizations) the appraisal can be extended to cover a larger part of the organization. By increasing scope and through using additional data collection methods, more reliable results will be available. A Class B appraisal supports this.

Also mature organizations may need results to accept comprehensive appraisals (type 3 organizations). Reviews and audits made by external organizations are very often considered as an inspection or a test that needs to be passed. Frequent reviews can have the side effect that the organizations become sensitive to external

interference in the process improvement work. This means that for less open organizations, less intrusive appraisal methods should be selected to build confidence. It may be necessary to start with Class C appraisals or to use a Class B appraisal on a limited set of process areas in a limited part of the organization.

When an organization is both open and mature (type 4 organizations), the use of Class A appraisals can be enhanced with frequent Class B appraisals to verify the direction of the performance improvements in the organization.

As the organizations move along a pathway, the strategy for the performance improvement needs to be adapted. We propose that the method described increases the probability for long term sustainable improvements, and can assist management in determining the appropriate level of activity in each moment.

5 Summary and Conclusions

Through the use of different diagnostic methods, we have captured and described the evolutionary paths of four organizations. To describe the development we use a mapping based on two organizational characteristics, maturity and openness. These have been selected as our experience indicates that both affect the possibilities to effectively use the results from appraisals. We acknowledge that there are several other organizational characteristics that are of equal importance for successful performance improvement. Among these are for example, organizational size, project pressure, management commitment and economical success. However, the experiences from the cases show that for the selection of the most appropriate diagnostic strategy the maturity and openness characteristics are outstanding. Based on the defined characteristics, we have identified four types of organizations, and this classification enables us to propose what class of diagnostic method should be used for a specific organization. The principle is that lower maturity organizations benefit from less intrusive appraisals, as the benefits of improving the processes need to be shown to create acceptance for external involvement in diagnostics and improvement activities. Also, if an organization is not open, the need to confirm that external assistance is beneficial.

6 Future Work

Looking into the future there are several ways to further develop the claims and recommendations made in this paper.

We would want to continue monitoring the evolutionary paths of the organizations in the case studies to get a better understanding of how organizations evolve over an extended period of time. This would in the long run enable us to identify patterns of behavior, and consequently allow prediction in some sense of how an organization is going to evolve. We would also want to extend the study to include additional organizations, as this would increase the reliability and validity of our claims.

It would also be interesting to make a comparison between the observed patterns of behavior and what can be called an “ideal” path. Often, reference models, such as the CMMI, expect and require organizations to evolve along extreme paths that are not achievable in reality. Whether or not this is detrimental for the way improvement

efforts are planned an executed remains to be investigated. It is reasonable to claim that having a better understanding for more realistic evolutionary paths will help improve the way improvement efforts are set up.

It would also be interesting to further investigate the openness characteristic of organizations. In this paper, we have let openness represents an aggregate of a set of organizational traits. However, more work is needed to better understand the organizational aspects that influence the ability to improve professionally. Future study would allow development and verification of a more complete set of organizational characteristics. These characteristics could then be used as readiness indicators, much as openness is used in this paper, when developing an improvement strategy and before undertaking an improvement effort.

In turn, a more complete set of organizational characteristics, would allow identification of additional types of organizations, which would add more perspective to the analysis.

Finally, more work is needed to verify the recommendations made in this paper as to the choice of diagnostic method for different types of organizations. In the long run, it will also be possible to extend the recommendations to include not only diagnostic activities, but also the remaining phases of the IDEAL model. This would entail developing comprehensive strategies for planning and executing professional performance activities in organizations based on their unique characteristics.

References

[1] CMMI® Product Development Team, CMMI for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing Version 1.1 (CMMI-SE/SW/IPPD/SS, V1.1), Staged Representation. Technical Report CMU/SEI-2002-TR-012, Pittsburgh, PA (2002)

[2] CMMI® Product Development Team, CMMI for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing Version 1.1 (CMMI-SE/SW/IPPD/SS, V1.1), Continuous Representation. Technical Report CMU/SEI-2002-TR-011, Pittsburgh, PA (2002)

[3] McFeeley, R., IDEALSM. A User's Guide for Software Process Improvement, Handbook CMU/SEI-96-HB-001, Pittsburgh, PA (1996)

[4] CMMI® Product Development Team, ARC, V1. 1; Appraisal Requirements for CMMI, Version 1.1. Technical Report CMU/SEI-2001-TR-034, Pittsburgh, PA (2001)

[5] Kitson, D. H., and Humphrey, W. S., The Role of Assessment in Software Process Improvement, Software Engineering Institute, CMU/SEI-89-TR-3, Pittsburgh, PA (1989).

[6] Minnich, I., CMMI Appraisal Methodologies: Choosing What Is Right for You, Crosstalk, Feb 2002, http://www.stsc.hill.af.mil/crosstalk/2002/02/minnich.html

[7] Steltzer, D., and Mellis, W., Sucess Factors of Organization Change in Software Process Improvement, Software Process Improvement and Practice, 4, (1998)

[8] Kasse, T., Action Focused Assessments for Software Process Improvement. Artech House Inc., Norwood, MA (2002)

Combining Capability Assessment and Value Engineering: A BOOTSTRAP Example

Pasi Ojala

Department of Information Processing Science, University of Oulu, P.O.Box 3000, FIN-90401 Oulu, Finland
pasi.ojala@oulu.fi

Abstract. Process improvement is a challenging task for software engineering. As Kuvaja [7]. has stated it:" It is difficult to find a unique way to identify a common improvement path suitable to all kinds of organizations." The BOOTSTRAP method gives an assessor tools to evaluate processes. As a method it evaluates processes with low capability and a high effect on an organization's goals the most important, and with high capability and a low effect on the organization the least important. It takes into account the organization's needs and goals, capability profiles of its processes and industry as the main drivers of process improvement. Value Engineering (VE) is a systematic method to improve the value and optimize the life cycle cost of a function or a facility. VE generates cost improvements without sacrificing the capability levels needed. By combining these two processes, process improvement work can be tailored to take into consideration, not only the capabilities of software processes but also the values of the same processes. This article discusses how to enhance the BOOTSTRAP assessment method to include new value characteristics and phases. Same principles can be applied also in other capability based assessment methods (for example CMM, CMMI or SPICE).

1 Introduction

Software has become more and more important in recent years. The number of companies producing software has grown constantly and we need software in our everyday lives. Therefore, the importance of good quality software has increased and it affects many people and most companies today. The processes producing software are important and from the customers' point of view they should produce cheap, reliable and usable help for everyday life. From the industry's point of view they should be cost effective and reliable. The paradigm of the software process proponents is that the quality of the software development process has a close relationship with the quality of the resulting software [4]. Krasner [6] points out that: "In a mature software organization, the following holds:

- Quality is defined and therefore predictable
- Costs and schedules are predictable and normally met
- Processes are defined and under statistical control"

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 471–484, 2004.

Because software process improvement and development is expensive to companies, capability based improvement only, is not enough to start process improvement work. To be effective, software process improvement and development methods also need to take into account costs created in processes. The goal of this study is to help to create an enhanced assessment method, which measures capability and value of processes. This is seen as important because if a process is not valuable to a company and its capability is low, the company should not worry about the situation [11]. On the other hand, if the capability of process is high and the value is high, the situation is in control, because the company is creating value by using a good process [11]. Problems arise when the company is creating quality with high capability processes, which do not create value, or processes are valuable but their capability is low (Figure 1.). These situations are not cost effective and profitable enough. In these situations the company either wastes resources in keeping up high capability processes which create a lot of costs and low value, or does not probably know that by increasing capability could probably increase also value significantly.

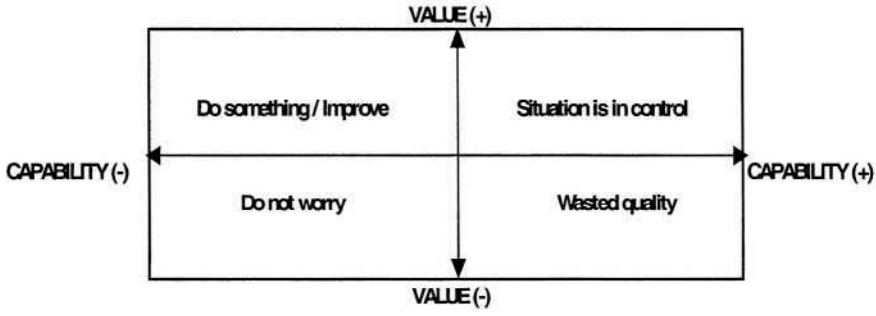


Fig. 1. Capability and value based process evaluation.

In this study value, cost and worth concepts are adopted from Value Engineering (VE) (synonymous with the terms Value Management and Value Analysis) which is a professionally applied, function-oriented, systematic team approach used to analyze and improve value in a product, design, system or service and process – a methodology for solving problems and/or reducing costs while improving performance/quality requirements. By enhancing value characteristics, VE is understood to increase customer satisfaction and value to investments.

Lawrence Miles describes Value Analysis as "a disciplined action system, attuned to one specific need: accomplishing the functions that the customer needs and wants". [11] Crum [2] calls Value Analysis (in the broadest sense) "a disciplined procedure directed towards the achievement of necessary functions for minimum cost, without detriment to quality, reliability, performance and delivery. It also means the application of the VA techniques to existing products." Crum describes Value Engineering as "the application of the VA techniques to existing products [2]."

VE can be understood also as a systematic method to improve the value and optimize the life cycle cost of a function or a facility [1, 3]. However, VE can be used also in improving the value and optimizing the life cycle cost of a process and a base- or a management practice [11].

In this study Value (V) is a measure usually in currency, effort or exchange or on a comparative scale, which reflects the desire to obtain or retain an item, service or ideal. Cost (C) is the price paid or to be paid. It can be divided into elements and to some extent functions (or processes). Worth (W) is defined as the least cost to perform the required function (or process) or the cost of the least cost functional equivalent. Using a formula V=W/C.

Dell' Isola [3] has also described value by using a formula. In his formula:

$$\text{Value} = \frac{\text{Function} + \text{Quality}}{\text{Cost}}$$

Where:

Function	=	The specific work that a design/item must perform.
Quality	=	The owner's or user's needs, desires, and expectations.
Cost	=	The life cycle cost of the product.

Therefore, he says that: Value = The most cost-effective way to reliably accomplish a function that will meet the user's needs, desires, and expectations.

Practically, in his formula Function + Quality is the same as worth.

2 BOOTSTRAP and Value Engineering Processes

The objective of BOOTSTRAP is to characterize the current practice, identify strengths and weaknesses and the ability of the process to control or avoid significant causes of poor quality, cost and schedule performance [8]. In this paper it is used as an example of capability assessment method to be enhanced by VE.

2.1 The BOOTSTRAP Assessment Process

Three different types of BOOTSTRAP assessments exist. Self assessment (Boot Check) is a starting point for process improvement. It is especially useful for small and medium -size companies). The assessment scope in the second type includes a full assessment. It is performed by two external assessors and it evaluates the capability of the software producing unit (SPU) and projects in terms of the organization, methodology and technology. This assessment includes all main processes. The third assessment scope is a focused assessment. It evaluates the capability of a selected set of processes. These processes are in the SPU or at the project level, (about assessment methods see [7])

All three types of BOOTSTRAP assessment cover or can cover SPU and project assessments. SPU assessment of the processes focuses on processes that are in the scope of the SPU. It also evaluates project processes based on managers' interviews. Project assessment includes assessment of the project processes based on interviews and evidence of practice capability.

The assessment process is performed in three main phases: preparation, assessment execution and action plan derivation (figure 2). The preparation phase has six steps:

Pre-Assessment briefing, Initialization, Assessment team identification, Collecting supporting material, Planning and scheduling and Defining confidentiality.

The second phase focuses on executing the assessment process. It includes five different steps: Opening briefing, SPU assessment, Project assessment, Evaluation and Feedback sessions.

The third phase, improvement planning and execution, focuses on documenting and presentation. It includes three different steps: Assessment report preparation, Final report preparation and On-site final meeting.

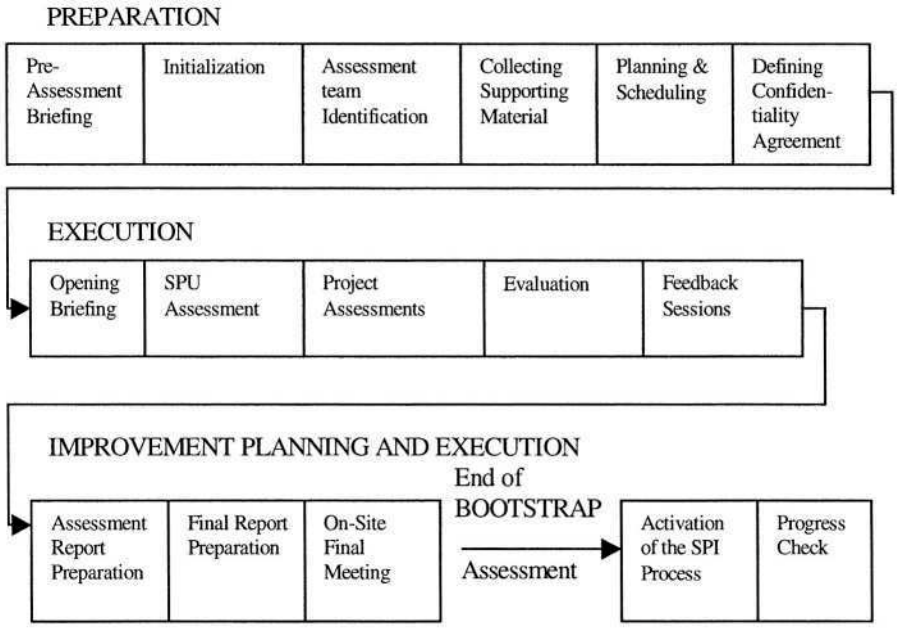


Fig. 2. BOOTSTRAP assessment process

2.2 Value Engineering as a Process

VE process consists of three main phases that are: pre-study, value study and post-study. The VE job plan is a systematic plan to make sure that the VE analyzing team understands customer requirements and develops a cost-effective solution.

Even though there are several job plans in the literature they are basically alike. The process of VE is the same. Park [12] has said: " No matter how many steps there are, the process is always the same, analysis, creativity, evaluation and development. In the following chapters the VE phases are categorized in three main classes, pre-study (tasks before value study), value study and post-study. Figure 3 shows the phases used in this study.

Pre-study activities should always be considered carefully before starting the VE process. The success of a VE study depends largely on preparation and coordination. In a properly established Value Engineering situation, orientation would include the

selection of appropriate areas to be studied and the appropriate team to accomplish the study.

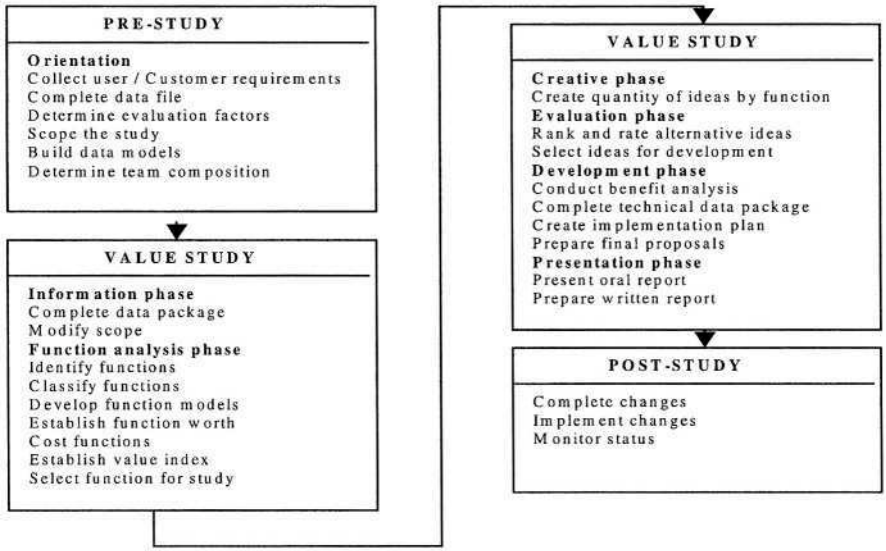


Fig. 3. VE process phases

A key point in organizing the VE effort is the use of the job plan or value study. The information phase is a fact-finding phase. The purpose is to accumulate all the factual information available in regard to the proposed area of study. The function analysis phase is the heart of the Value Methodology. Mudge [10] has formulated that function analysis is based on two major parts: define a function and evaluate the function relationships. He emphasizes that: "within defining the functions lies the keystone of the Value Engineering systematic approach". He continues that the evaluation of function relationships, is accomplished by taking the above technique and the information and data secured in the information phase and establishing a relationship between them. In the creative phase the VE team puts forward suggestions to answer the functions that have been selected for study. The VE team evaluates the ideas generated in the creativity phase using one of a number of techniques, many of which depend upon some form of weighted vote. This stage forms a crude filter for reducing the ideas generated to a manageable number for further study [5]. The accepted ideas, selected during the evaluating phase are investigated in detail in the development phase for their technical feasibility and economic viability. The refined ideas supported by drawings, calculations and costs, are presented by the VE team to the body who commissioned the VE exercise.

The purpose of the post-study is to assure the implementation of the approved value study change recommendations. Implementation tasks are made by the VE team, the organization's own personnel or together.

Practically, successful VE depends highly, upon which kind of techniques are used in the Value Analysis process [3, 9]. Right techniques identify unnecessary cost,

remove obstacles, and provide a course of action that will ensure the development of value alternatives of merit. These techniques can be divided into key techniques, techniques to test value, costing techniques and different kinds of screening and selection techniques [3, 9].

3 Towards a VE-Enhanced Assessment Method

In the previous chapters this study presented a discussion of what are the basics of BOOTSTRAP and VE. This chapter combines capability and value based methodologies together and outlines the enhanced BOOTSTRAP assessment methodology. Same principles can be applied also in combining VE and other capability based assessment methods like CMM, CMMI or SPICE. However, each combination needs specific definition work for example in defining relationships for existing processes and VE processes which is seen out of scope of this study.

3.1 Assessment Types

Using VE in the assessment process does not necessarily create new types of assessment. In some aspects their content changes partially. Self assessment (Boot Check) can be the starting point for process improvement or assessment and it can be done two ways. The company can carry out a complete self assessment or it can do a guided self assessment. The second type of assessment can be a full assessment. It can evaluate the capability of the software producing unit (SPU) and projects in terms of organization, methodology and technology. This assessment can include all main processes. The use of VE brings new steps into full assessment and it can be used to find out which processes give the biggest value [11]. The third assessment type can be focused assessment. It can evaluate the capability of a selected set of processes. These processes can be at the SPU or project level. Using VE brings new techniques to the selection part of focused assessment [11]. By doing first VE analysis for processes it is easier for an assessor to pick up a process with poor value to be assessed also from the capability point of view.

3.2 VE Enhanced Assessment Methodology

VE study bases on functions which assessor has formed himself using a verb and a noun and therefore method has been flexible in analyzing different physical product parts, which has to be defined individually. Later on cost accounting tools have been used strongly to assign costs as well as possible to defined functions. From the methodological point of view there does not seem to be any reason why this could not be done also to SW components thinking that they are SW parts. By doing so company would be able to classify its SW product into components and their subcomponents and finally also calculate value, worth and cost for each of them.

In literature there has been presented comments on that capability -based assessment methods are often too big and difficult to handle. They include a lot of processes and practices and they are not necessarily always related to existing process

descriptions used in a company. Sometimes company might be interested in to assess capabilities for those practices which are in their process descriptions not to those practices which are defined in the reference model like BOOTSTRAP or CMMi. However, assessing capabilities for these practices is time-consuming job, because there is no clear reference model for all situations which have been defined to companies process descriptions. From VE point of view value, cost and worth can be assessed easily also for processes and practices defined in company's own process description. In this situation there is just no reference model and functions are defined as processes and subfunctions as practices.

However, if capability assessment is done using a reference model arises question that why value assessment could not be done using the same model. By calculating also values, costs and worth to all processes and their practices could company get much more detailed data of its processes. It is clear that if company knows capabilities, cost, worth and value for all processes it can make better decisions in developing them too. Not all the processes necessarily give the same value if same amount of money is used to develop them.

From capability assessment point of view VE processes can be defined as own process cluster in existing reference model. In this enhancement the new processes have own practices, relationships to other processes and defined capability levels. On the other words in this assessment the focus would be to solve how capable company is to collect, analyze, develop and improve its value. In this study this possibility is examined further using BOOTSTRAP as a reference model.

3.3 Capability Assessment for VE Processes Using BOOTSTRAP

The BOOTSTRAP process model has three dimensions. The process dimension is called the BOOTSTRAP process model. The process dimension and capability dimension form together with the technology support dimension, the full BOOTSTRAP model [7].

In the VE enhanced BOOTSTRAP process model VE processes are included in the model as a new fourth dimension. This is seen justified because value analyzing and improving processes (VE processes) are independent combination with own methodological background. They are also directly related to all other processes and they do not seem to belong to existing process categories. They use own tools and the scope of VE study is also different basing on value.

However, because cost, worth, value and capability can be calculated to all processes in the enhanced process model it seems justified that VE processes can be included to process model. [11]. The enhanced BOOTSTRAP process model is shown in figure 4.

BOOTSTRAP assessment evaluates processes by measuring process capabilities. In the Bootstrap model process capability is the ability of each process to achieve its goals in the context of the assessed organization [7]. The enhanced BOOTSTRAP assessment evaluates capabilities also for VE processes. It includes also calculations for cost, worth and value for all processes (about value, cost & worth calculations see [11].). The enhanced method includes the same six levels as basic BOOTSTRAP method (these are compliant also with ISO 15504):

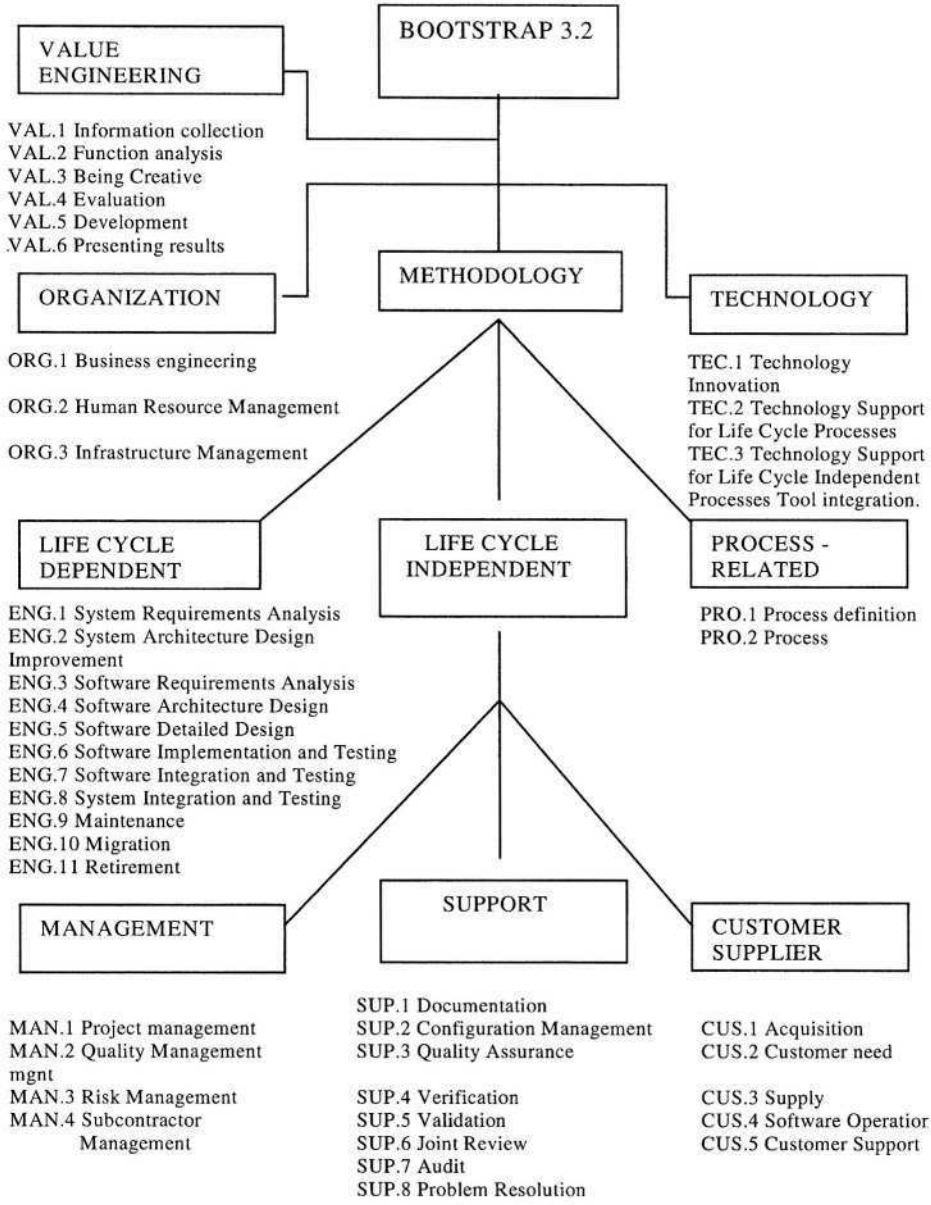


Fig. 4. The VE enhanced BOOTSTRAP process model

Level 0:	Incomplete process: The process does not fulfill its purpose.
Level 1:	Performed process: The implemented process achieves its defined purpose by utilizing a set of practices (base practices) that are initiated and followed and that produce identifiable work products.
Level 2:	Managed process: The performed process delivers work products of acceptable quality within defined time scales and resource needs.
Level 3:	Established process: The process is performed according to a standard process definition that has been suitably tailored to the needs of the process instance. The process is performed with qualified and available resources.
Level 4:	Predictable process: The execution of the established process is supported by process goals and measures which are used to ensure that the implementation of the process contributes to the achievement of the business goals.
Level 5:	Optimizing process: The predictable process optimizes its capability to meet current and future needs and achieves repeatability in meeting defined business goals.

In the VE enhanced assessment methodology the capability levels are divided into quartiles so that the evaluation can show how close or far the process is from the next level. The basic idea is to show and evaluate the processes by using the categorized six levels with the quartile prezision.

The final results of the enhanced assessment can be presented as a figure with quartile levels, or in the case of a SPICE rating with normal 0-5 level rating. As the assessment includes two levels, there are two figures. The SPU level presents results of organizational processes and the project level shows results of the individual projects.

The SPU level profile shows the capability of organizational processes and it reflects the management's point of view. The project level profile shows the process capability of the individual projects. Comparison between SPU and project profiles also provides valuable information to support improvement planning [7].

In the assessment process the assessors also collect information, which is not directly included in the rating. This information is used to explain the overall results with examples. Considering VE processes this information is mainly cost related information which is used later on for calculating cost, worth and value for processes and their basepractices and management practices [11].

3.4 Assessment Process

The enhanced assessment process is performed in four main phases: preparation, assessment execution, action plan derivation and making VE study (figure 4). The preparation phase has six (or seven) steps: [11]

1. Pre-assessment briefing
2. Initialization
3. Function analysis (in case of assessor is using value based focusing)
4. Assessment team identification
5. Collecting supporting material

6. Planning and scheduling
7. Defining confidentiality

The pre-assessment briefing justifies the importance of taking part in enhanced assessment. The new process developed also gives in this phase reasons why the VE process is taken along to the assessment work. Initialization concentrates on selecting project and processes to be assessed or decides that function analysis is used for value based focusing [11]. In the enhanced initialization phase the assessor can use the VE techniques to make a decision why some processes are more important to be assessed than others.

The assessment team identification selects the assessment team and interviewees. This step should evaluate also whether assessors are capable of taking responsibility of the VE parts of assessment or not. In many cases it would be worthwhile to use VE specialists or persons who know VE. Collecting supporting materials gives the possibility to see previous assessment results, quality manuals, etc. This step, in the enhanced assessment process also collects new information depending on which selection techniques are used in selecting processes or in focusing assessment work.

Planning and scheduling schedules time, makes resource definitions and interview planning as before. Defining the confidentiality agreement focuses on determining who is allowed to see the assessment and what kind of results. This phase does not change in the enhanced assessment process. In the company, SPU and project level confidentiality can differ. This should also include determination of the VE proposals confidentiality level.

If the company is interested in focusing BOOTSRAP assessment on areas where value creation is the biggest, the assessor could use value-based focusing. Value-based focusing can select the processes, which give the biggest value for the BOOTSTRAP assessment process. In the case of value-based focusing the VE function analysis is placed after the initialization phase, because focusing should be done before assessment team nomination [11].

The second phase focuses on executing the assessment process. It includes five steps: [11]

1. Opening briefing
2. SPU assessment
3. Project assessment
4. Evaluation
5. Feedback sessions

The purpose of the opening briefing is to make contact with people to be interviewed and explain them how the assessment will be done. If possible, in this step top-level management could show commitment to this assessment by taking a part in the briefing. SPU assessment includes manager interviews and evaluation of SPU documentation. Project assessment includes interviews and evaluation of practice capability evidence. In evaluation, assessors create preliminary assessment profiles and in feedback sessions, assessment findings are presented to interviewees, feedback is collected and errors are corrected. In this phase it is possible that an assessor uses VE techniques to get some tasks done more easily. For example creative techniques can be used in opening briefing to create an open atmosphere for assessing work.

The third phase, improvement planning and execution, focuses on documenting and presentation. It includes two different steps: [11]

1. Assessment report preparation
2. Final assessment report

The assessment report preparation is included in the enhanced assessment process, which mainly contains the facts of assessment results and calculated capability levels. Possible improvement actions are presented only after the fourth phase. In the final report all assessment results, strengths and weaknesses are documented.

The fourth phase, making the VE study, concentrates on finding out which functions or processes create the biggest value. In case of calculating values to the processes, VE functions are considered to be processes and secondary functions are considered to be practices (base-, or management practices). In case of calculating values also for product parts, VE functions are assigned to be product parts. For example, it is possible to calculate capabilities and values for VE enhanced BOOTSTRAP processes and their practices [11]. On the other hand, to get more product-related point of view it is also possible to calculate value for the product parts in question. In this case, improvement work has two dimensions, which are SW process improvement and product development.

Fourth phase includes all seven steps of the VE process [11]. It is only partially needed, if the assessment is based on value based focusing, where all process values are already calculated after the initialization phase. In this case the needed phases are the creative, evaluation, development and presentation phase [11].

1. Orientation
2. Information phase
3. Function analysis phase
4. Creative phase
5. Evaluation phase
6. Development phase
7. Presentation phase

Orientation is mainly needed for preparing the data file, determining evaluation factors and focusing the VE part of the study. In this step the assessor should build data models and determine team composition. Orientation can be considered as a pre-study.

During the information phase a data package is completed and the scope of the VE study is modified. The function analysis is the most important part of the VE process [3, 9]. It concentrates on: Identifying functions, Classifying functions, Developing function models, Establishing function worth, Costing functions, Establishing value index and Selecting functions for study.

In this study and in earlier research it has been shown to be possible to calculate values, worth and costs directly for processes [11]. From VE point of view, this shortens VE process, because there is necessarily no need to identify and classify functions and form a function model any more. Practically, in this case all functions have a correspondent process from the process model in question. More detailed cost, worth and value information can be calculated by defining secondary functions as certain base- or management practices [11].

The creative phase creates a quantity of ideas by function and evaluation phase ranks and rates alternative ideas and finally selects ideas for development. The development phase conducts benefit analysis, completes a technical data package, creates an implementation plan and prepares a final VE proposal for presentation. The development phase also prepares software process improvement plan. In this phase it would be recommended to take into account also the main findings of the rated processes and their capability levels reported in the phase final assessment report preparation. This should be done because, when company has a list of rated processes and their capability levels it would be worthwhile to compare this information to the VE proposals improvement activities, before giving presentation (about comparison see [11]).

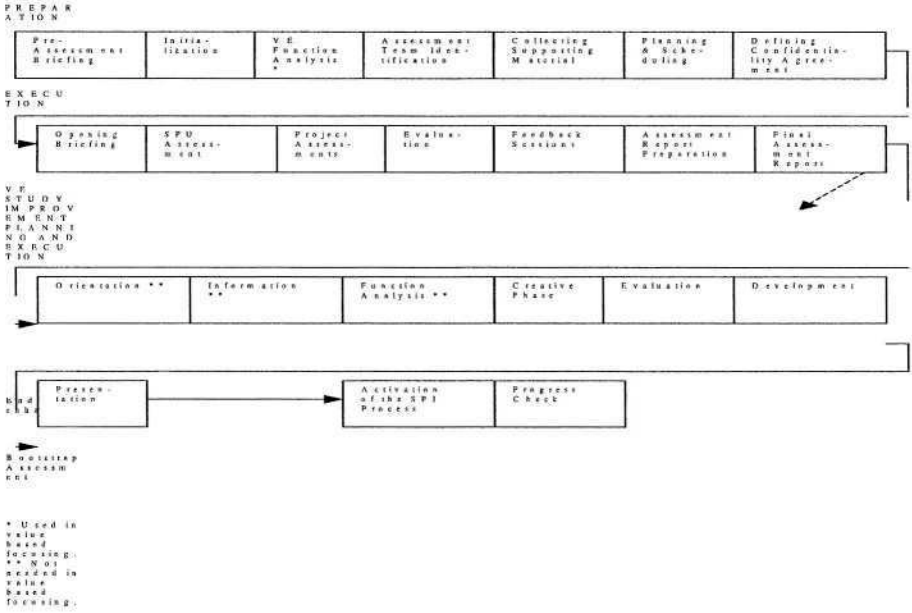


Fig. 5. The enhanced BOOTSTRAP assessment process

After the VE enhanced BOOTSTRAP assessment process all changes decided need to be completed and implemented. After changes it is also important to monitor the status of developed new situation. The VE enhanced BOOTSTRAP assessment process (as well as new process model), which includes some characteristics and techniques of VE, is the main idea and thrust of this article. The VE enhanced BOOTSTRAP assessment process with the new fourth phase is presented in the figure 4 (see also [11]).

4 First Assessment

The first assessment using enhanced BOOTSTRAP assessment method was done in summer 2003. It was a full assessment and it included capability level calculations for

all processes described in enhanced process model (figure 2). Cost, worth and value were calculated as an ad hoc basis for certain processes.

Experiences of the first assessment showed that the enhanced BOOTSTRAP assessment model works in practice. Capability levels can be calculated for VE processes as well as cost, worth and value to examined processes. However, in calculating cost, worth and value practical problems might arise if a company does not have sophisticated cost accounting systems, which can assign costs to processes and their practices.

In the final meeting a new enhancement opened up a many sided discussion. Managers vs. designers and technical vs. economical personnel seemed to find a topic which they all were able to discuss. This topic was value. They started to discuss process by process about their possibilities to create more value, cut costs and develop better worth to customer.

The enhanced BOOTSTRAP assessment model still needs more development work. The relationship between Value Engineering processes and existing BOOTSTRAP processes need more definition work as well as capability levels definition for new processes.

5 Conclusion

In broadest sense this article wanted to open up a discussion of concept of value for SPI. By calculating cost, worth and value for processes including their practices and finally, also to their improvements or improvement proposals, it is obvious that SPI has stronger value also for those who see that it is a waste of money or it has a little value. This is because by doing so it can clearly show in which areas the improvement efforts are worth doing.

The basic statement of this article claimed that capability and value are both as important in measuring and improving software processes. Therefore, combining capability assessment method and VE creates more efficient tools to everyday assessment and software process development work. As an example, this article presented how to combine typical capability based assessment method, like BOOTSTRAP, and VE.

Firstly, this article presented the basics of BOOTSTRAP assessment process, assessment types and VE process. The purpose of this was to reveal the possibilities to combine these different processes and methods, no matter whether they have the same kind of aims and characteristics or not. Both methodologies respected same basics. BOOTSTRAP respected companies' business goals and the VE process identified areas where unnecessary costs were possible to be removed while assuring that quality, reliability, capability, and other critical factors were met or exceeded the customer's expectations. So, basically VE respected the same basic idea of customer's expectations and satisfaction as BOOTSTRAP was doing, when it emphasized the company's business goals as an assessment's cornerstone.

Secondly, this article outlined VE enhanced assessment types, method and process. The capability based assessment types were seen possible to be used also in VE enhanced assessment. The VE enhanced assessment method was defined to include enhancements like 1) value assessment for products, 2) value assessment for processes without reference model, 3) combined capability and value assessment using reference

model and 4) capability assessment for enhanced process model including VE processes.

Thirdly, the fourth possibility was examined further and VE processes like information collection, function analysis, being creative, evaluation, development and presenting results were included to reference model. The capability scoring principles to VE enhanced assessment method were suggested to be same as they are in normal capability based assessment methods. The defined VE enhanced assessment process was tailored to include three parts: (1) preparation, (2) execution and (3) VE study, improvement planning and execution.

Fourthly, this article shortly presented the results of the first industrial assessment. According to this assessment, VE enhanced assessment method and process were seen usable and worth for further development.

Further writings in this area should present more empirical evidence of industrial assessments and a discussion about advantages and disadvantages of suggested VE enhanced assessment process and method. Practically, it would be interesting to combine this enhanced idea of assessment also to other capability based assessment methods like CMM, CMMi or SPICE. More research should be made also in evaluating VE techniques and selecting the most suitable ones to enhanced assessment work.

References

1. Cooper, R. & Slagmulder, R.: Target Costing and Value Engineering. Productivity Press, Portland Oregon (1997)
2. Crum, L.W.: Value Engineering: The organized search for value. Longman, London (1971)
3. Dell'Isola, A.: Value Engineering. Practical Applications...for Design, Construction, Maintenance & Operations. R.S.Means Company Inc. Kingston MA (1997)
4. Humprey, W.S.: Managing the Software Process. Reading, Addison Wesley, MA (1989)
5. Kelly, J. and Male, S.: Value Management in Design and Construction. E & FN Spon, U.S (1993)
6. Krasner, H.: The Payoff for Software Process Improvement: What it is and How to get it. Elements of Software process assessment and improvement. K. E1. Emam & N.H. Madhavji (eds). Los Alamitos, Calif. IEEE: Computer Society (1999)
7. Kuvaja, P.: BOOTSRAP 3.0 – A spice Conformant Software Process Assessment Methodology. Software Quality Journal, 8. Kluwer Academic Publishers, Netherlands (1999) 7-19
8. Kuvaja, P., Bicego, A., and Dorling, A.: SPICE: The software process assessment model, Measuring Training Based Process Improvement. September 11-12. Proc. ESI_ISCN '95 Conference, Vienna Austria (1995)
9. Miles, L.D.: Techniques of Value analysis and Engineering. McCraw-Hill, New York (1972)
10. Mudge, A. E.: Value Engineering.McGraw-Hill, New York (1971)
11. Ojala, P.: Enhancing Software Process Improvement Using Value Engineering. University of Oulu (2001)
12. Park, R.: Chrysler Corporation, Cost, Function, Value Analysis, Value Control Manual. St. Lucie Press, New York (1978)

Assessing the State of Software Documentation Practices[1]

Marcello Visconti[1] and Curtis R. Cook[2]

[1] Departamento de Informática, Universidad Técnica Federico Santa María, Valparaíso, Chile
visconti@inf.utfsm.cl
[2] Computer Science Department, Oregon State University, Corvallis, Oregon, USA
cook@cs.orst.edu

Abstract. A system documentation process maturity model and assessment procedure were developed and used to assess 91 projects at 41 different companies over a seven year period. During this time the original version evolved into a total of four versions based on feedback from industry and the experience gained from the assessments. This paper reports the overall results obtained from the assessments which strongly suggest that the practice of documentation is not getting a *passing grade* in the software industry. The results show a clear maturity gap between documentation practices concerned with defining policy and practices concerned with adherence to those policies. The results further illustrate the need to recognize the importance of improving the documentation process, and to transform the good intentions into explicit policies and actions.

Keywords: system documentation processes, maturity model, key practices, degree of satisfaction, assessment results

1 Introduction

The purpose of this paper is to report the results after using a system documentation process maturity model and assessment procedure to assess 41 companies involving 91 projects and over 370 software professionals in a period of seven years. The essence of the model remained unchanged throughout its first three versions, but industry feedback as well as the experience gained during the assessments led to modifications of key practices and question scoring scheme which are explained elsewhere [11]. Then, a fourth version changed the model structure to one without the maturity levels as an indicator of process maturity that focused on the key practices. This evolution led to the formulation of a process maturity meta-model, further explained in [13].

In this paper we describe the general state of the practice of the system documentation process based on the assessments results. To facilitate comparisons

[1] This research has been funded in part by the National Commission on Scientific and Technological Research of the Government of Chile, CONICYT, through research project FONDECYT #1990845.

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 485–496, 2004.

between the results from the different assessments using the distinct versions, only the key practice profiles are used. The maturity levels computed for the first three versions are shown for illustration purposes only. Maturity levels were not part of the fourth version.

The idea behind the Documentation Process Maturity Model is very simple: most defects discovered during software testing are documentation defects (requirements and design defects - defects in documentation that were introduced before any code was written). Empirical studies have shown that poor quality, out of date, or missing documentation is a major cause of defects in software development and maintenance [1], [2], [8], [9]. Thus, documentation is a key component in software quality and improving the documentation process will have considerable impact on improving the quality of software. Our solution scheme has been to design a maturity model that provides the basis for the assessment of current documentation process and guides the identification of key practices and challenges to improve the process [10]. The focus is on the documentation used in software development and maintenance and does not consider end-user documentation. Our approach has been influenced by the Capability Maturity Model (CMM[sm][2]) developed by Carnegie Mellon University's Software Engineering Institute (SEI) [4], [6], [7].

This paper is organized as follows: next section presents the Documentation Process Maturity Model and discusses its foundations and related research that led to its formulation. The following section highlights the major changes performed from the first three versions to the fourth in terms of model structure, and gives a summary of the model evolution. Then, the results obtained from the assessments conducted with each version are shown. The final section discusses the state of the practice of software documentation based on the assessment results, and gives our conclusions.

2 Documentation Process Maturity Model – Framework and Evolution

The Documentation Process Maturity Model (DPMM) is a description of process maturity, capability and practices that characterize an organization that generates high quality documentation. (Recall that documentation refers to the system documentation generated as part of the software development process. It does not include end-user documentation.) A four level software system documentation process maturity model and assessment procedure have been developed. The model represents an ideal process and the assessment determines where the organization stands relative to the model. The model and assessment procedure were influenced by CMM in that key practices, indicators, and challenges are defined for each of the four levels of the model; for the assessment procedure a questionnaire, that takes only 30 minutes to complete, is administered to each member of the project team. The tabulated questionnaire responses are used to generate an assessment report that gives the maturity level and a documentation process profile that indicates what practices the organization is doing well, what practices need improvement, and challenges to move to the next higher maturity level. More information about the context

[2] CMM[sm] is a service mark of Carnegie Mellon University.

framework that was the supporting basis to develop the maturity model and related assessment procedure is given in [10]. The model's overall structure is presented in Table 1.

Table 1. Documentation Process Maturity Model - Summary

	Level 1 Ad-hoc	**Level 2 Inconsistent**	**Level 3 Defined**	**Level 4 Controlled**
Keywords	Chaos, Variability	Standards Check-off list Inconsistency	Product assessment Process definition	Process assessment Measurement Control Feedback Improvement
Succinct Description	Documentation not a high priority	Documentation recognized as important and must be done.	Documentation recognized as important and must be done well.	Documentation recognized as important and must be done well consistently
Key Practices	Ad-hoc process Not important	Inconsistent application of standards	Documentation quality assessment Documentation usefulness assurance Process definition	Process quality assessment and measures
Key Indicators	Documentation missing or out of date	Standards established and use of check-off list	SQA-like practices	Data analysis and improvement mechanisms
Key Challenges	Establish documentation standards	Exercise quality control over content Assess documentation usefulness Specify process	Establish process measurement Incorporate control over process	Automate data collection and analysis Continually striving for optimization

A key to understanding DPMM is the succinct description of each of the four levels in the model, as shown in Table 1. The four succinct descriptions are: Level 1 - all the required documentation may not be done; Level 2 - all the required documentation is done; Level 3 - all the required documentation is done well and attention is paid to the usefulness of the documentation; and Level 4 - an optimizing loop where measurement of the process and usability provide feedback to continually improve the process.

It is important to notice that for the fourth and last version of the model we have dropped the maturity level idea and have only concentrated on key practices. The

reason is that in our experience maturity levels seemed to draw attention away from what is really the key issue in software process improvement: the key practices. We have extended this idea and have proposed a meta-model to identify key practices that also considers a product dimension (in terms of quality assurance and usability) when assessing a particular process during the diagnosis phase. This shows to be especially important when the process produces tangible deliverables, as is the case of the documentation process. Further details are available in [13].

DPMM has evolved over four versions. In the first three versions, each of the four levels has a number of key practices associated with it. Version 4 of DPMM only has a set of associated key practices and no maturity levels. In versions 3 and 4, a number of subpractices were associated with each key practice. Table 2 shows a summary to illustrate the evolution the model has undergone over the four versions. These changes are the result of documentation process assessments performed in the industry, assessing over 90 projects at more than 40 organizations. Full details of the evolution over the first three versions are available in [11].

Table 2. Summary - Evolution of DPMM

Version	Maturity Levels	Key Practices	Sub Practices	Questions	Assessed Companies	Assessed Projects
1	4	18	n/a	56	7	26
2	4	19	n/a	68	13	34
3	4	9	26	67	9	19
4	n/a	11	32	75	12	12

Table 3 shows a list of all key practices identified for each version, indicating which ones have remained unmodified, which have been dropped and which have been added. It is important to note that in Table 3 a key practice that was rewritten or modified appears as a dropped key practice and later as an added key practice. For example the key practice *Use of a check-off list of required documents* in version 1 was modified to *Mechanism to check that required documentation is done* in versions 2 and 3. In other cases, some dropped key practices became subpractices of later versions, so they were not actually dropped either. Subpractices are not shown in Table 3. Only one key practice remained unmodified for all four versions: *Process improvement feedback loop.*

To carry out the assessments we have used an assessment questionnaire, whose purpose is to determine where an organization's documentation process stands relative to the model. The assessment questions are derived directly from the model and its key practices. There are one or more questions for each key practice. The process maturity level (for the first three versions) and the degree of key practice satisfaction are determined from the questionnaire responses. For version 1 the key practices were rated as *Not Satisfied, Partially Satisfied,* or *Fully Satisfied.* This rating system seemed ambiguous and too coarse as there was a wide latitude in the degree of key practice satisfaction especially when a practice was classified as *Partially Satisfied.* For example, does *Partially Satisfied* mean the key practice is satisfied seldom or often or half of the time? *Not Satisfied* and *Fully Satisfied* were also ambiguous. Does *Not Satisfied* mean never or most of the time it is not satisfied? Does *Fully Satisfied* mean always or most of the time? To eliminate these

ambiguities, from version 2 on there was a change from three to five degrees of satisfaction as follows: *Very High, High, Medium, Low* and *Very Low*.

Each person on a project team completed the assessment questionnaire. The degree of satisfaction of a key practice was generally determined by the average of the responses to the questions associated with that key practice.

Table 3. Summary – Evolution of Key Practices of DPMM

Key Practices of DPMM	Version 1	Version 2	Version 3	Version 4
Consistent creation of basic software development documents	X	X		
Creation of basic software documents				X
Documentation generally recognized as important	X	X		
Written statement or policy about importance of documentation	X	X		
Management recognition of importance of documentation				X
Adequate time and resources for documentation	X			
Adequate time for documentation		X		
Existence of documentation policy or standards		X	X	X
Monitor implementation of policy or standards				X
Adherence to documentation standards	X			
Adherence to documentation policy or standards		X	X	
Use of check-off list of required documentation	X			
Mechanism to check that required documentation is done		X	X	
Use of simple documentation tools	X	X		
Accuracy and reliability of documentation	X	X		
Mechanisms to update documentation	X	X		
Existence of a defined process for creation of documents			X	X
Mechanisms to monitor quality of documentation	X	X	X	
Methods to assure quality of documentation				X
Methods to assess usefulness of documentation	X	X		
Assessment of usefulness of documentation			X	
Assessment of usability of documentation				X
Use of common sets of documentation tools	X	X		
Use of advanced documentation tools	X	X		
Documentation-related technology and training	X	X		
Documentation is traceable to previous documents		X		
Definition of software documentation quality and usability measures				X

Key Practices of DPMM	Version 1	Version 2	Version 3	Version 4
Measures of documentation process quality	X	X		
Measures of documentation process quality and usefulness			X	
Collection and analysis of documentation quality measures				X
Analysis of documentation usage and usefulness	X	X		
Analysis of documentation process quality and usefulness			X	
Collection and analysis of documentation usability measures				X
Process improvement feedback loop	X	X	X	X
Integrate CASE and documentation tools	X			

Finally, an assessment report is generated from the questionnaire responses. The report contains an executive summary with the maturity level (only for the first three versions), a documentation process maturity profile and an improvement action plan. Besides the maturity level, the executive summary lists the key practices that were not satisfied, those that need improvement, and those that were missing. The process maturity profile indicates the degree of satisfaction of each key practice. Specific actions to improve existing key practices or to address missing key practices to move to the next higher level are described in the improvement action plan. See [10] for an example of an assessment report. See [12] for a proposed framework to face the action planning required to move on after conducting the assessments.

3 Assessment Results

This section reports, for each version, the degree of key practice satisfaction from the assessments conducted for that version.

Using Version 1

The first version was released in March 1993. It was used to assess 26 projects at 7 companies in the time period March 1993 - September 1995, and Tables 4 and 5 show the results in terms of degree of satisfaction for each key practice and the overall results in terms of maturity levels.

Version 1 was used to assess 26 projects. Although only four out of the 26 projects classified defect data by development phase, there was one promising data point that supported the model. Three of the four projects were from the same organization. Two projects were assessed as a level 2 and one a level 1. The defect data indicated that for the one project at level 1, 41% of the defects found during integration testing were design and requirements defects whereas for the two projects at level 2 only 25% of the defects found were design and requirements defects [2]. This strongly suggests that defects are indeed detected earlier in projects with a higher documentation maturity level.

Table 4. Key Practices and Degrees of Satisfaction – DPMM Version 1

Level	Key Practices	Not	Partial	Fully
1	1. Consistent creation of basic software development documents	4%	77%	19%
	2. Documentation generally recognized as important	12%	8%	81%
2	3. Written statement or policy about importance of documentation	38%	12%	50%
	4. Adequate time and resources for documentation	27%	54%	19%
	5. Adherence to documentation standards	19%	69%	12%
	6. Use of check-off list of required documentation	31%	27%	42%
	7. Use of simple documentation tools	4%	8%	88%
3	8. Accuracy and reliability of documentation	0%	62%	38%
	9. Mechanisms to update documentation	65%	31%	4%
	10.Mechanisms to monitor quality of documentation	65%	27%	8%
	11.Methods to assess usefulness of documentation	77%	15%	8%
	12.Use of common sets of documentation tools	35%	23%	42%
	13.Use of advanced documentation tools	65%	23%	15%
	14.Documentation-related technology and training	100%	0%	0%
4	15.Measures of documentation process quality	100%	0%	0%
	16.Analysis of documentation usage and usefulness	96%	4%	0%
	17.Process improvement feedback loop	69%	23%	8%
	18.Integrate CASE and documentation tools	73%	12%	15%

Table 5. Breakdown by Maturity Level - DPMM Version 1

Level	%
1	62%
2	38%
3	0%
4	0%

The results from Table 4 show that most organizations are at level 1. The higher the maturity level the lower the degree of satisfaction for the key practices in that level. It's important to contrast degree of satisfaction of practice 2 *(Documentation generally recognized as important)* with that of practice 4 *(Adequate time and resources for documentation),* or between practices 3 *(Written statement or policy about importance of documentation)* and 5 *(Adherence to documentation standards).* Both cases stress the need to move from good intentions to concrete actions.

Using Version 2

The second version was released in September 1995. It was used to assess 34 projects at 13 companies in 3 countries in the time period September 1995 - September 1996. Tables 6 and 7 show the results in terms of degree of satisfaction for each key practice and the overall results in terms of maturity levels.

Table 6. Key Practices and Degrees of Satisfaction - DPMM Version 2

Level	Key Practices	VL	L	M	H	VH
1	1. Consistent creation of basic software development documents	0%	24%	53%	15%	9%
	2. Documentation generally recognized as important	29%	6%	3%	12%	50%
2	3. Written statement or policy about importance of documentation	26%	12%	12%	9%	41%
	4. Adequate time for documentation	18%	24%	26%	32%	0%
	5. Existence of documentation policy or standards	21%	21%	32%	18%	9%
	6. Adherence to documentation policy or standards	65%	26%	3%	6%	0%
	7. Mechanism to check that required documentation is done	15%	26%	24%	15%	21%
	8. Use of simple documentation tools	0%	0%	9%	32%	59%
3	9. Accuracy and reliability of documentation	0%	18%	56%	26%	0%
	10.Mechanisms to update documentation	65%	26%	9%	0%	0%
	11.Methods to monitor quality of documentation	59%	32%	6%	3%	0%
	12.Methods to assess usefulness of documentation	85%	12%	0%	3%	0%
	13.Use of common sets of documentation tools	9%	12%	21%	24%	35%
	14.Use of advanced documentation tools	53%	41%	3%	3%	0%
	15.Documentation-related technology and training	65%	26%	6%	3%	0%
	16.Documentation is traceable to previous documents	53%	29%	15%	3%	0%
4	17.Measures of documentation process quality	71%	26%	3%	0%	0%
	18.Analysis of documentation usage and usefulness	85%	12%	3%	0%	0%
	19.Process improvement feedback loop	56%	41%	0%	3%	0%

Table 7. Breakdown by Maturity Level - DPMM Version 2

Level	%
1	68%
2	29%
3	3%
4	0%

The results from using this version of the model are not substantially different from those obtained with the previous one. As the maturity levels increases the

satisfaction of related key practices gets lower; the apparent contradiction between intentions and actions remains, evidenced by the decreasing degrees of satisfaction of practices 2 *(Documentation generally recognized as important)* and 4 *(Adequate time for documentation),* and 3 *(Written statement or policy about importance of documentation),* 5 *(Existence of documentation policy or standards)* and 6 *(Adherence to documentation policy or standards).*

Using Version 3

The third version was released in September 1996. It was used to assess 19 projects at 9 companies in 2 countries in the time period September 1996 – December 1998. See Tables 8 and 9.

Table 8. Key Practices and Degrees of Satisfaction - DPMM Version 3

Level	Key Practices	VL	L	M	H	VH
1	No key practices – in this level there is only a set of basic practices	0%	6%	28%	50%	17%
2	2.1 Existence of documentation policy or standards	6%	11%	33%	17%	33%
	2.2 Mechanism to check that required documentation is done	0%	22%	17%	39%	22%
	2.3 Adherence to documentation policy or standards	33%	39%	0%	17%	11%
3	3.1 Existence of a defined process for creation of documents	17%	39%	17%	28%	0%
	3.2 Methods to assure quality of documentation	6%	56%	22%	17%	0%
	3.3 Assessment of usefulness of documentation	0%	33%	50%	17%	0%
4	4.1 Measures of documentation process quality and usefulness	39%	44%	11%	6%	0%
	4.2 Analysis of documentation process quality and usefulness	67%	28%	6%	0%	0%
	4.3 Process improvement feedback loop	50%	22%	22%	6%	0%

Table 9. Breakdown by Maturity Level - DPMM Version 3

Level	%
1	72%
2	28%
3	0%
4	0%

Again, the results from this version are very similar to those of the previous versions. The degrees of satisfaction of practices 2.1 and 2.3 illustrate the big gap between the existence of the policy and the adherence to it.

Using Version 4

The fourth version was released in December 1998. It has been used to assess 12 projects in the time period December 1998 – present. Table 10 shows the results in terms of degree of satisfaction for each key practice. No maturity levels are computed for this version.

Table 10. Key Practices and Degrees of Satisfaction - DPMM Version 4

Key Practices	VL	L	M	H	VH
1. Creation of basic software documents	0%	8%	25%	17%	50%
2. Management recognition of importance of documentation	0%	8%	25%	0%	67%
3. Existence of documentation policy or standards	17%	8%	25%	0%	67%
4. Monitor implementation of policy or standards	8%	8%	42%	25%	17%
5. Existence of a defined process for creation of documents	25%	0%	33%	33%	8%
6. Methods to assure quality of documentation	8%	25%	42%	25%	0%
7. Assessments of usability of documentation	8%	67%	17%	18%	0%
8. Definition of software documentation quality and usability measures	67%	17%	8%	8%	0%
9. Collection and analysis of documentation quality measures	42%	42%	17%	0%	0%
10.Collection and analysis of documentation usability measures	100%	0%	0%	0%	0%
11.Process improvement feedback loop	42%	33%	17%	0%	8%

The results are not very different from the results provided by the assessments using the previous three versions. Once again there is a considerable gap between the intentions and the actions, as clearly illustrated by the degrees of satisfaction of practices 2 *(Management recognition of importance of documentation),* 3 *(Existence of documentation policy or standards)* and 4 *(Monitor implementation of policy or standards).*

4 Summary and Conclusions

Software process maturity models are conceptual frameworks to improve software development and maintenance processes. They do not guarantee success in obtaining higher quality products or more efficient processes. To achieve these goals, there has to be a real commitment and actions to fulfill that commitment from the whole organization.

This paper has presented the results obtained from the assessments of 91 projects at 41 companies using a documentation process maturity model. Its purpose is to profile the key components of the system documentation process, and an assessment procedure whose goal is to determine how well an organization's documentation process matches that of the model. We developed a simple assessment questionnaire that takes a short time to complete and whose responses provide information for an assessment report that describes how the organization's process stands relative to the model and suggests changes to improve the process.

Our results show a clear picture of process immaturity and general software documentation process practice non-satisfaction. Not surprisingly, this is reflected in a steady decrease in key practice satisfaction expected in less mature organizations. The key practices are listed in a natural maturity order (e.g. practices of more mature organizations appear later in the list) that clearly show the trend. Assessed organizations mostly satisfy key practices linked to assuring the existence of policies or standards, but they fail for higher maturity key practices related to assuring the actual monitoring of compliance to these policies or standards. There is an acknowledgement of the problem but little action to back it up. The results also highlight the low satisfaction for key practices related to assuring the quality and usability of the actual documents produced. The main challenge is to make a serious effort to enhance the software documentation process, by adopting and carrying out explicit policies aimed at process improvement.

One crucial point that our experience has made very clear is the importance and fundamental role of the key practices. They drive the generation of the assessment questionnaire. When an organization's process is assessed relative to the model, the assessment is actually measuring the organization's degree of satisfaction of the key practices. That is, how well is the organization carrying out the key practices. In spite of this, the maturity level has been the most commonly used indicator of the organization's process maturity. The maturity level is such a broad indicator that it is of limited value as it provides incomplete information about which key practices are satisfied and which are not.

The results and analysis reported in this paper are part of a broad on-going research effort aimed at improving the whole software process.

Acknowledgments. We thank former students Patricio Antiman from Universidad Técnica Federico Santa María and Sheldon Dealy from Oregon State University for their help in improving the questionnaires, conducting the assessments and tabulating the results over the first three versions. We are also grateful to all the software professionals whose invaluable input and support helped carry out this research effort over this many years.

References

1. D. Card, F. McGarry and G. Page. Evaluating software engineering technologies. IEEE Transactions on Software Engineering, 13(7) (1987) 845-851
2. C. Cook and M. Visconti. Documentation is important. CrossTalk, 7(11) (1994) 26-30
3. C. Cook and M. Visconti. New and improved documentation process model, in *Proceedings of the 14th Pacific Northwest Software Quality Conference,* Portland, Oregon, October 1996 (PNSQC, Portland, 1996), pp. 364-380

4. W. Humphrey. Managing the software process (Addison-Wesley, Reading, 1989)
5. B. Lientz and E. Swanson. Problems in applications software maintenance. Communications of the ACM , 24(11) (1981) 763-769
6. M. Paulk, B. Curtis, M. Chrissis and C. Weber. The Capability Maturity Model guidelines for improving the software process (Addison-Wesley, Reading, 1995)
7. M. Paulk, B. Curtis, M. Chrissis and C. Weber. Capability Maturity Model, version 1.1. IEEE Software, 10(4) (1993) 18-27
8. J. Pence and S. Hon III. Building software quality into telecommunications network systems. Quality Progress, (October 1993) 95-97
9. H. Rombach and V. Basili. Quantitative assessment of maintenance: an industrial case study, in *Proceedings of the IEEE Conference on Software Maintenance,* Austin, Texas, September 1987 (IEEE, Washington, 1987), pp. 134-144
10. M. Visconti and C. Cook. A software system documentation process maturity approach to software quality, in Proceedings of the 11th Pacific Northwest Software Quality Conference, Portland, Oregon, October 1993 (PNSQC, Portland, 1993), pp. 257-271
11. M. Visconti and C. Cook. Evolution of a maturity model – critical evaluation and lessons learned. *Software Quality Journal* 7, 223-237 (1998)
12. C. Cook and M. Visconti. What to do after the assessment report?, in Proceedings of the 17th Pacific Northwest Software Quality Conference, Portland, Oregon, October 1999 (PNSQC, Portland, 1999), pp. 214-228
13. M. Visconti and C. Cook. A meta-model framework for software process modeling. Lecture Notes in Computer Science 2559, Springer-Verlag, 532-545 (2002).

Requirements Prioritization Challenges in Practice

Laura Lehtola, Marjo Kauppinen, and Sari Kujala

Helsinki University of Technology, Software Business and Engineering Institute,
P.O. Box 9210, FIN-02015 HUT, Finland
{Laura.Lehtola, Marjo.Kauppinen, Sari.Kujala}@hut.fi

Abstract. Requirements prioritization is recognized as an important activity in product development. In this paper, we describe the current state of requirements prioritization practices in two case companies and present the practical challenges involved. Our study showed that requirements prioritization is an ambiguous concept and current practices in the companies are informal. Requirements prioritization requires complex context-specific decision-making and must be performed iteratively in many phases during development work. Practitioners are seeking more systematic ways to prioritize requirements but they find it difficult to pay attention to all the relevant factors that have an effect on priorities and explicitly to draw different stakeholder views together. In addition, practitioners need more information about real customer preferences.

1 Introduction

Prioritizing requirements is an important activity in product development [1,2,3, 4]. When customer expectations are high, timelines short, and resources limited, the product must deliver the most essential functionality as early as possible [5] and the scope of each release must be limited [1], Many projects face the fact that not all the requirements can be implemented because of limited time and resource constraints. That means that it has to be decided which of the requirements can be removed from the next release. According to Wiegers [5] information about priorities is needed, not just so as to be able to ignore the least important requirements but also to help the project manager to resolve conflicts, plan for staged deliveries, and make the necessary trade-offs. Harwel et al. [6] describe a priority as being a characteristic of a requirement that can be used for different purposes, depending on program and company needs.

However, requirements prioritization is also recognized as a very challenging activity. For example, Lubars et al. [7] report that none of the companies in their study really knew how to assign and modify priorities or how to communicate those priorities effectively to project members. Furthermore, Karlsson et. al [2] argue that despite the recent rapid and welcome growth in requirements engineering (RE) research, managers still do not have simple, effective, and industrially proven techniques for prioritizing requirements.

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 497–508, 2004.

Some findings concerning the challenges in requirements prioritization can be found in the literature. Aurum et al. [8] describe the RE process, in essence, a complex communication and negotiation process involving many stakeholders. They argue that it includes a great deal of invisible decision-making [8]. For example, Wiegers [5] argues that customers might not want to prioritize their requirements, because they are afraid of having just the most important ones done and developers do not want to admit that they are not able to implement all the requirements. Political issues are discussed by other authors [9], too. The interdependencies between requirements are another topic that is discussed [10].

The articles written about requirements prioritization issues can be roughly divided into three categories. The first category involves those papers that describe the current state of RE processes in the industry (for example [7] [1]). These studies provide the information that requirements prioritization is an important and challenging issue. Other authors (for example [8] [11]) discuss decision making in the context of the whole RE process. These studies provide a basis for decision-making in the development context. A few authors introduce their own approach to requirements prioritization (for example [2] [4]) and evaluate it with a specific set of requirements, either with industrial partners or without them. On the basis of these studies, it seems that more information about the current state of requirements prioritization practices is needed.

The high-level goal of this study is to clarify the field of requirements prioritization. In this paper, we assess current requirements prioritization practices in industry by describing the state of the art in two organizations in which they had used different kinds of free-form requirements prioritization practices.

This paper is structured as follows: the case organizations and research methods used are described in Section 2; Section 3 provides an overview of the lessons learned from this study, and finally, Section 4 concludes the paper.

2 Research Design

The experience drawn on in this research comes from work with two industrial partners of the Qure[1] project. The research goal of the Qure project was to investigate how organizations can develop products that better satisfy user and customer needs. This research was part of a subproject investigating how user and customer needs and requirements should be prioritized. The research work was carried out in the product development units of two Finnish companies. The companies (introduced in Table 1) represent two different kinds of application domains.

In order to clarify the current practice in the requirements prioritization area we carried out a focus group study and two in-depth interviews. In addition, we observed real requirements prioritization work in different phases of product development.

[1] The Qure (Quality through Requirements) project was research project at the Helsinki University of Technology (HUT).

Table 1. Case companies and their application domains and product types

Company	Number of employees	Application domain	Product type
A	500	Information management systems for building public infra and energy distribution designers	Software systems
B	1100	Measurement systems for meteorology, environmental sciences and traffic safety	Interactive systems

The goal of the focus group session was to find out how and in which phases of the development work companies prioritize requirements in practice. We also clarified which factors have an effect on priorities and from which sources the practitioners gather the information on which they base their priority decisions. In addition, we inspected the problems the developers felt they had with their current requirements prioritization practices. In our case, the focus group consisted of four representatives from the two case companies. The participants and their relationship to requirements prioritization are introduced in Table 2.

Table 2. Focus group participants and their relation to requirements prioritization

Company	Description of participant's work and his relation to prioritization
A	Product manager. Main task is to elicit and prioritize requirements.
A	Leader of the R&D unit. Substantial experience of mutual prioritization of requirements.
B	Project manager. Collects information about markets and writes requirements documents. Substantial experience of mutual prioritization of requirements.
B	Product development process engineer. Main task is to implement a requirement management tool to the organization. Knows basics of requirements prioritization.

Rather than providing quantifiable responses to a specific question obtained from a large sampling of the population, focus group participants provide a flow of input and interaction related to the topic [12]. In our case, while we wanted to gain fresh insights regarding the issue [12] and the problem area was not strictly bounded, we thought that four people participating in a semi-formal discussion together would provide more intimate information of requirements prioritization than would be obtained by inter-views or questionnaires. For example, Templeton [13] supports our point of view by describing focus groups as small, temporary communities, formed for the purpose of the collaborative

enterprise of discovery. The assembly is based on some interest shared by the panel members, and the effort is reinforced because the panelists are paid for the work [13]. We did not pay cash for the work, but panel members had an incentive to participate because of benchmarking.

The discussions were semiformal. This meant that a researcher worked as the facilitator of the session by giving the participants five discussion topics (introduced in Table 3) and by leading the discussion. Each session (1/2 hour each) was started in such a way that participants wrote their thoughts and key words about the topic on post-it notes. After that, the post-it notes were gathered and organized on a white board using the affinity grouping technique [14]. The post-it notes were used as a basis for the discussions. All the discussions were recorded and analyzed later by reorganizing comments into topic tables according to similarity between them.

Table 3. Focus group discussion topics

Number	Topic
1	Current requirements prioritization practices in the companies
2	Problems that companies have with their current practices
3	Factors that have, or should have, an effect on priority decisions
4	Sources for priority information
5	Development phases in which requirements are prioritized

As mentioned earlier, we also carried out semi-structured, in-depth interviews by interviewing two project managers, one in each company. The goal of these was to have more detailed information about current practices and to understand better the project point of view in requirements prioritization. The questions covered prioritization practices used and challenges involved. We also focused on clarifying what kind of priority information they needed in the beginning of project. Both of the interviews were audio recorded. The two interviewees are presented in Table 4. In addition, we observed real prioritization work in one case company. The goal of the observation was to focus not only on what practitioners say they do but also on what they actually do.

Table 4. Interviewees

Company	Description of participant's work and his relation to prioritization
A	Project manager. Responsible for a modeling team. Prioritized requirements by negotiating informally.
B	Project manager. Responsible for releasing a second version of a product. Prioritized requirements by negotiating informally.

3 Results

Five main findings were identified that describe the current practice in requirements prioritization in the two case companies. The lessons learned from the companies, summarized in bullet points below and described in Sections 3.1 - 3.5, are:

- Requirements prioritization is an ambiguous concept
- Prioritization practices are informal and dependent on individuals
- Requirements are prioritized in many phases
- Developers do not know enough about customer preferences
- The priority of a requirement is based on many factors

3.1 Requirements Prioritization Is an Ambiguous Concept

Although it is essential that people have a common understanding about the terms they use and activities they perform in product development, the terms "requirements prioritization" and "priority" have several different meanings in practice. This causes confusion and misunderstandings among product development personnel. The terms are not uniformly defined in organizations, so in spoken language different activities with different purposes are referred to by the same terms. This happens without the awareness of the practitioners.

The activity called "requirements prioritization" had many meanings in the case companies. Occasionally, the term was used with the meaning "How do we decide which requirements are the most important ones for the company in the long run?"; sometimes it meant "How do we decide, which requirements we have to implement right away in the next product release?" or "How do we select the requirements that will be implemented first in this project?", or "Which of the requirements describe the system in high-level terms?".

A "priority" is an attribute of a requirement which should be the result of the activity called requirements prioritization. In the case companies, there were ambiguities in the usage of the term "priority", as well. In some cases the term was used as a quantity meaning "the importance of a requirement to the customer" and in other cases it described how soon the requirement would be implemented. In some cases these two scales, the importance scale and the time scale, were equal to each other. However, in release planning other things than importance to the customer, for example interdependencies of the requirements can have a greater effect on implementation decisions and their schedule.

Not only were there ambiguities with the usage of terms, but also with the usage of prioritization scales (introduced in Section 3.2). The categories high, medium and low were experienced as being ambiguous. For example, one of the interviewees mentioned that "We needed a lot of discussion about the meanings of each priority level with the project group when we set priorities. I was surprised how different the meanings we had in our minds were."

3.2 Prioritization Practices Are Informal and Dependent on Individuals

There are no commonly agreed ways to perform requirements prioritization in the companies. Requirements are prioritized mostly on the basis of experience of development personnel. The factors one should take into account when deciding priorities are not commonly explicated. Roughly speaking, individuals make prioritization decisions mostly on the basis of their tacit knowledge or feelings.

No explicit requirements prioritization methods were in use in the companies. The development personnel tried to make a rough guess which requirements were the most important ones to customers and users, how profitable requirements were to their own company, and how all this cohered with the strategy of the company, but there were no systematic practices for these analyses. Contracts made with customers and promises given to them in informal discussions played a major role when priorities were being set. An interesting point we found was that often the companies descend into a situation where they try to avoid the biggest "breach of contract" -payment. "First we promise to implement something and then we try to pin down how much it will cost if we do not implement it." complained one project manager.

Table 5. Requirements prioritization practices and related problems

Practices companies use	**Comments that describe related problems**
Assessing requirements value for customers and its development costs	"We try to judge costs in the early phases of development. We have no formal method for that."
Priority lists of local areas	"Our local areas have the same problem as we have. How to know what is truly important to customers?"
Prioritization scales	"I have no idea how we divide the requirements into categories. The process is very mutual."
Negotiation in project meetings	"We have a person who knows what it takes in the way of resources to implement the requirement and a person who knows how much effect it has on business. It is just a mutual discussion."

The participants mentioned that in practice there is no time to figure out all the relevant information as a basis for priority decisions. The development personnel had for example difficulties in analyzing all the raw requirements they gathered from customers. "There is no time to analyze thousands of wishes. Much of the work is done intuitively" said one of the interviewees. In most cases the requirements specification is written only by one person. That leads to situations

in which the writer of the specification thinks that there is no need to prioritize the requirements any more. One of the interviewees complained that "The writer does not want to drop anything. The first version is, in a way, prioritized in his head."

The practices companies currently use for prioritizing requirements are listed in Table 5. In the early phases of development the companies try to analyze the costs and value of the requirements. These analyzes are very informal and there is usually no documentation about these decisions and their rationale. In the other company the product managers also try to prioritize raw requirements according to priority lists drawn up locally, but they find it quite difficult to truly combine the information from different sources. In product development projects, requirements are mutually grouped into three categories using prioritization scales. This is usually done by product managers. Negotiation in project meetings is used especially in those situations where the project group is not able to implement all the requirements in the given time. These discussions are informal. The project group just makes the decision if there is or is not time to implement a requirement in this particular release.

3.3 Requirements Are Prioritized in Many Phases

Decisions about which requirements can be included in the next version of the product and which can be postponed are needed in many phases of product development. Requirements definition is a process during which priority decisions have to be made iteratively. Requirement priorities are needed, not only for making decisions as to which requirements to leave out, but also for analysis purposes after the release and in order to help the communication within the organization and with the customers.

Product management needs high-level information about customer preferences, markets and the company's own strategy and resources when they decide which requirements will constitute the basis of the product or release. This information is also needed to decide which of the raw requirements or user needs gathered should be evolved further. Participants felt that they need more systematic ways to work out this "high-level priority view" and common ways to link it to lower level-requirements.

Personnel working at the customer interface felt that they need information about priorities for communication purposes. They want to have a big picture of how they have managed to serve different market segments with their product. The other case organization wanted to know how much value they had produced for each customer segment with their current release. They felt that the priorities that customers give to their own raw requirements and change requests can be a key for managing customer satisfaction. In addition to this, they want to know better in advance how much value different requirements combinations would produce for different customers and market segments.

Product development projects need more, and better-documented, information about which requirements are important according to earlier phases of development. They also need to know the rationales for these decisions. This in-

formation is wanted in order to help them to decide which requirements they can leave out if there is no time to implement all that was planned. Participants mentioned that they needed priority information in order to make the kick-off of the implementation work easier. They would have liked to know which of the requirements constitute the basis of the product.

3.4 Developers Do Not Know Enough about Customer Preferences

The product development personnel would like to know why a requirement is important to users or customers. Usually they have no idea because people are working separately in the product development; product development personnel do not have direct contacts with users and customers. In addition to this, there are no common practices to communicate customer and user information through the product development process.

Particularly in small projects, contact with customers and users was felt to be too narrow. A great deal of important information was gathered from users by the help desk calls they made. In addition to this, product development staff communicated with vendors and gathered information in this manner. In the case companies, product managers created the first requirements specification on the grounds of discussions they had had with customers. They had an idea which requirements were important to customers and placed requirements into priority categories.

There were no generally agreed ways to transfer priority information to the project group and usually the original reason for requirements being considered important failed to reach as far as to the project manager and other project group. One participant complained that "Usually there is no clear explanation besides requirement or need, why it is important or wanted. A person who does not know anything about this particular requirement from the customer point of view does the prioritization."

3.5 The Priority of a Requirement Is Based on Many Factors

The requirement's importance to a customer is an important, but usually not the only, factor that has an effect to a requirement's priority. There are many difficulties in defining which factors should be taken into account when setting the priorities. Getting the right information for to use as the basis for prioritization decisions is not always easy.

Our study indicates that there are three main points of view which are more or less explicitly taken into account when setting priorities. These three points of view, introduced in Figure 1, seem to encapsulate the other factors. For a company it is a lifeline to profit, so issues like customer relationships, competitors, and the importance of the requirement's source for the company have to be taken into account. An other point of view is that of customers and users. The development organization must know which of the requirements are most important to them. A third point of view is implementation. The resources of the

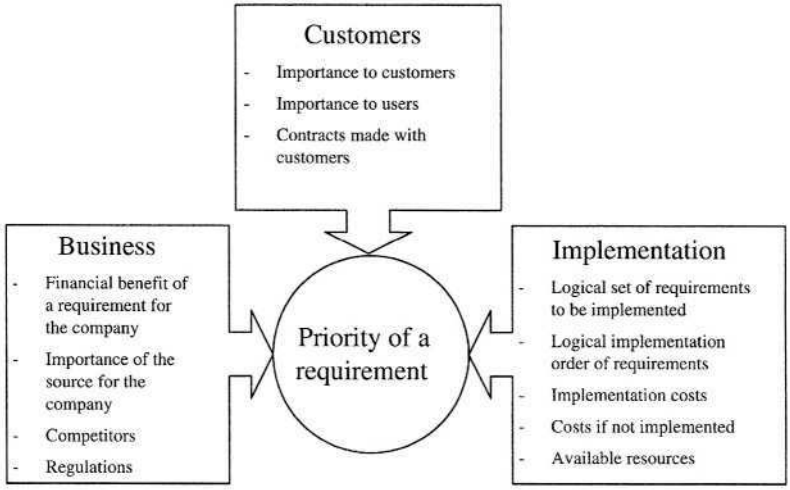


Fig. 1. Three points of views having an effect to requirement's priority

Table 6. Challenge, description and related comments

Challenge	Description	Related comments
How to know which are the relevant factors?	The priority of a requirement is based on many factors. It is not clear which points of views should be taken into account, when deciding priorities.	"Priorities are dynamic. There is different criteria for priorities in different phases."
How to get correct information?	Priority information can be skewed or even wrong. Companies lack ways to communicate priority information further.	"They (the local areas) have the same problem as we have. How to know what is truly important to customers?"
How to combine information from different sources?	Companies do not know how to combine different market and customer preferences systematically.	"There are requirements from this customer and that customer. There are Japanese requirements and requirements from the U.S.A" "Now we are in a situation where the customers who complain most get most."

company and the manufacturing situation, as well as the logical implementation order, have an effect on which requirements should be implemented first.

Developers can gain information about factors that have an effect on priority from many sources or stakeholders. The participants mentioned that one of the most used source for getting information concerning the basis of priority decisions is discussions with key customers and their representatives. In one of the companies the customer representatives in different geographical areas made their own priority lists, which were integrated into a world-wide priority list in the main company.

It is not always easy to decide which factors and stakeholders should be taken into account when deciding priorities, getting the right information about them, and combining information from different sources. These problems are examined more carefully in Table 6.

4 Conclusion

This work aimed to clarify the field of requirements prioritization. In this paper, we have described current requirements prioritization practices in the two case companies and highlighted the practical challenges involved. The main contribution of the paper is that it describes the nature of prioritization problem and why its solution is so hard. By addressing the prioritization challenges, researchers and practitioners may better be able to tackle them.

Our study indicates that requirements prioritization practices are informal and dependent on individuals. Individual practitioners prioritize requirements on the basis of their experience and tacit knowledge. Even the meaning of the term "requirements prioritization" depends on individuals. This causes misunderstandings. Lubars et al. [7] also found in their study that companies do not know how to set and modify priorities.

According to our research, having systematic requirements prioritization practices is a challenge because requirements prioritization requires a great deal of non-trivial decision making. Also, Yeh [15] reports that requirements prioritization is one of the most crucial and at the same time difficult tasks that faces the decision makers. Our study indicates that the priority of a requirement is based on many factors such as financial benefit of the requirement for the company, requirement's importance to users, and implementation costs. These factors can be grouped into three main points of views: business, customers, and implementation.

Requirements prioritization is usually described as a part of the requirements analysis phase [16]. Our study indicates that instead of being just one-off activity, requirements prioritization is needed in many phases of the development work. The importance of different viewpoints depends on the development phase. For example, in the early phases of development work business issues seem to have more influence on priorities than implementation issues. Other authors also discuss the importance of different viewpoints in requirements prioritization. For example, Moisiadis [17] argues that prioritizing requirements should involve rep-

resentatives from each group of stakeholders with a vested interest in the success of the development project.

In practice, it is difficult to get all the important information about factors that influence the priorities of requirements and explicitly draw different points of views together. Practitioners make decisions about priorities without explicitly being aware which factors they take into account and to what extent. Developers also feel that they do not have enough information about real customer preferences.

The lessons described in this study were gained from two Finnish organizations and therefore there might be issues that may not be appropriate to all organizations. However, as other authors have reported similar kinds of challenges [7] [15] we think that our findings are also valid for many other organizations. This study identifies important research challenges which should be investigated further. One of the research challenges in the future is to help organizations define which points of views have an effect on requirements priorities in different phases of development and in what extent. Another important challenge is to investigate how to combine different points of views in requirements prioritization.

Acknowledgments. The authors would like to thank the industrial partners of the Qure project for participating in the research and Dr. Jyrki Kontio, Dr. Pekka Abrahamsson, and Mr. Jarno Vähäniitty for their considerable effort in commenting on this paper. Additional thanks are due to Mr. Jouko Koski for his technical advice.

References

[1] Siddiqi, J., Shekaran, M.: Requirements engineering: The emerging wisdom. IEEE Software 2 (1996) 15–19

[2] Karlsson, J., Ryan, K.: A cost-value approach for prioritizing requirements. IEEE Software 14 (1997) 67–74

[3] Karlsson, J., Wohlin, C., Regnell, B.: An evaluation of methods for prioritizing software requirements. Information and Software Technology 39 (1998) 939–947

[4] Regnell, B., Höst, M., Natt och Dag, J., Beremark, P., Hjelm, T.: An industrial case study on distributed prioritisation in market-driven requirements engineering for packaged software. Requirements Engineering 6 (2001) 51–62

[5] Wiegers, K.E.: Software Requirements. Microsoft Press, Redmont, Washington (1999)

[6] Harwell, R., Aslaksen, E., Hooks, I., Mengot, R., Ptack, K.: What is a requirement? In: Proceedings of the Third International Symposium of the NCOSE. (1993) 17–24

[7] Lubars, M., Potts, C., Richter, C.: A review of the state of the practice in requirements modelling. In: Proceedings of IEEE Symposium on Requirements Engineering (RE'93), IEEE Computer Society Press (1993)

[8] Aurum, A., Wohlin, C.: The fundamental nature of requirements engineering activities as a decision-making process. Information and Software Technology 45 (2003) 945–954

[9] Andriole, S.: The politics of requirements management. IEEE Software 15 (1998) 82–84
[10] Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., Natt och Dag, J.: An industrial survey of requirements interdependencies in software product release planning. In: Proceedings of the Fifth IEEE International Symposium on Requirements Engineering (RE'01). (2001) 84–91
[11] Regnell, B., Paech, B., Aurum, A., Wohlin, C., Dutoit, A., Natt och Dag, J.: Requirements mean decisions! - research issues for understanding and supporting decision-making in requirements engineering. In: Proceedings of the First Swedish conference on Software Engineering Research and Practice (SERP'01), Ronneby, Sweden (2001) 49–52
[121 Edmunds, H.: The Focus Group Research Handbook. NTC Business Books in conjunction with the American Marketing Association, Lincolnwood (Chicago), Illinois (1999)
[13] Templeton, J.F.: The Focus Group - A Strategic Guide to Organizing, Conducting and Analyzing the Focus Group Interview. Revised edn. Mc Graw Hill, New York (1994)
[14] Oddo, F., ed.: Coach's Guide to the Memory Jogger II. GOAL/QPC, Methuen, MA (1995)
[15] Yeh, A.C.: Requirements engineering support technique (request) a market driven requirements management process. In: Proceedings of the Second Symposium of Quality Software Development Tools, New Orleans, USA (1992) 211–223
[16] Sommerville, I.: Software Engineering. 5 edn. Addison-Wesley, Wokingham, England (1996)
[17] Moisiadis, F.: The fundamentals of prioritising requirements. In: (Web) Proceedings of Systems Engineering/Test and Evaluation conference (SETE2002). (2002) http://www.seecforum.unisa.edu.au/Sete2002/ProceedingsDocs/.

A Requirement Elicitation Method in Collaborative Software Development Community

Masatoshi Shimakage and Atsuo Hazeyama

Graduate School of Mathematics and Informatics, Tokyo Gakugei University,
4-1-1 Nukuikita-machi, Koganei-shi, Tokyo 184-8501, Japan
hazeyama@u-gakugei.ac.jp

Abstract. We developed a collaborative software development community-ware whose goal is to provide practical tasks for the students in software engineering education in a university and to provide software for education purpose. We performed a pilot experimentation for the community-ware. The results show that software requesters made software change requests for not only functional aspects but also user interface. This paper proposes a user interface oriented requirement elicitation process. The process is that software requesters present functional requests and user interface design which met the requests. Then based on the requests and the user interface, developers and software requesters decide priority for each requirement in a collaborative manner.

1 Introduction

By permeating computer network like the Internet into our society, global software development [5] and/or open source software development [9] have been actively conducted. In several universities, global and collaborative software engineering courses have been done with company [1].

We aim at establishing a collaborative software engineering community, which elicits requirements from an unspecified number of people via the Internet, submits some of the tasks to a software engineering course, and opens the software students groups in the course have developed to the public. We developed a prototype community-ware for supporting such activities within the community. We have also conducted a preliminary experiment to validate the model and environment [12]. As the results, we found the following issues:

(1) some requesters have not only functional requirements but also user interface (UI) for the developed software. Whenever interim software is released, they ask for change requests for UI
(2) On account of characteristics of community-ware, it is necessary to suppose that several requesters insist on their own requirements and the requirements fall into inconsistency.

To solve these issues, this paper proposes a requirement elicitation method, which navigates stakeholders to attain agreement with UI specification as well as functional

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 509–522, 2004.

requirements between several anonymous requesters and developers on the community- ware.

This paper is organized as follows: Section 2 discusses two issues raised above in more detail. Section 3 proposes a user interface driven requirement elicitation method in collaborative software engineering community. Finally we conclude this paper.

2 Related Work

We discuss two issues raised in the previous section in more detail and show the direction of our study.

(1) Some requesters have not only functional requirements but also user interface (UI) for the developed software. Whenever interim software is released, they ask for change requests for UI
(2) On account of characteristics of community ware, it is necessary to suppose that several requesters insist on their own requirements and the requirements fall into inconsistency.

As for (1), Curtis et al. reported that as users get software, they tend to require higher functionality [2]. Our community is not its exception. We found the following from our preliminary experiment: requirement elicitation was done by text-based communication via a bulletin board system (BBS). After functional requirements were agreed on, the developers wrote programs without presenting design documents to the requesters, and released the system to the field for trial usage called "the Factory phase" [12]. A lot of change requests including modifications of UI specification were reported, so rework occurred. We recognized importance of design documents. On the other hand, UI design environment is required that developers, who have little domain knowledge and have tight schedule constraints, can design efficiently.

Issues which are similar to (2) have been dealt with as "Market Driven Requirement Engineering" ([4], [10], and so on). Most approaches provide informal requirement prioritization method. Karlsson and Ryan proposed "Cost-Value approach" for requirement prioritization [6]. It determined requirements prioritization by applying the AHP (Analytical Hierarchy Process) [11] method to the relative cost and value of candidate requirements of a target system. This approach has only one evaluation criterion from users, therefore it can not be applied to our collaborative software development community as it is because unspecified number of people participate in this community who have requirements independently. Although interdependency may exist among functional requirements of a target system, Karlsson and Ryan do not take this situation into account.

This paper proposes a UI oriented functional requirements elicitation method to overcome the issue (1). This paper also proposes a release planning based on the requirements that were elicited by the solution to (1), which takes unspecified number of requesters and interdependency among functional requirements into consideration.

3 A Proposal of Requirements Prioritization Method

This section proposes the requirement elicitation method to solve the problems as mentioned in the previous section. 3.1 shows an overview of the whole process, and the sub-processes – functional requirements elicitation, non-functional requirements elicitation, and release planning – are described in the subsequent sub-sections.

3.1 The Incremental Development Process Model

We adopt the incremental development process model [3] as the fundamental life cycle model. The incremental life cycle process model has characteristics that time interval to reflect user requirements on software is short and as the result, the gap with respect to recognition for software between developers and requesters decreases.

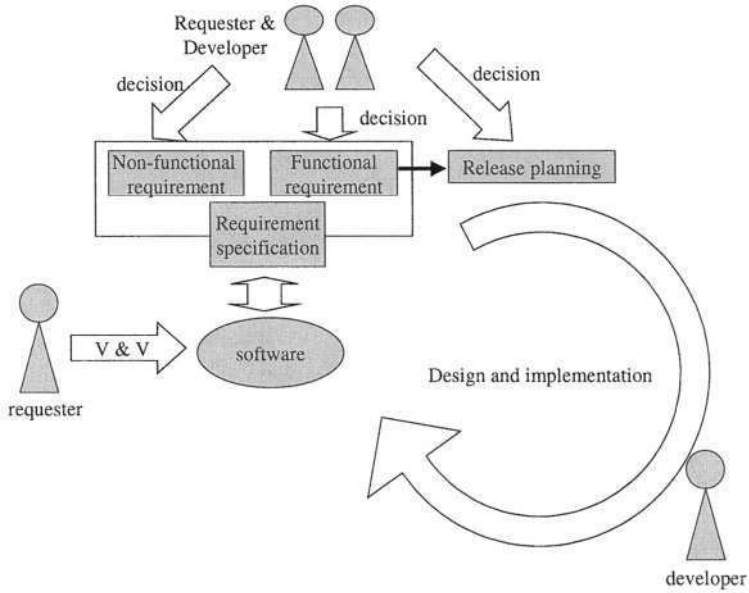


Fig. 1. The incremental development process model

As change requests occurred a lot for the software, which was developed in the trial experiment [12], we decided to adopt the process model.

Komiya et al. showed how software engineers proceeded to discuss with clients in order to elicit requirements [7]. Based on their result, we propose the following process. First of all, functional requirements elicitation is done, then non-functional requirements are decided, and finally a release plan is created.

1. Requesters and developers decide functional requirements by following the UI oriented functional requirement elicitation method (see section 3.2 in more detail)

2. Requesters and developers decide non-functional requirements (see section 3.3 in more detail)
3. Requesters and developers decide a release plan
4. Developers implement requirements with high priority based on the release plan and release the software
5. Requesters get the released software and test it

5.1 If requesters satisfy with the result of testing, developers start to implement requirements with the next level of high priority and release the software

5.2 If requesters do not satisfy with the result of testing, developers make modification till resolving the dissatisfaction

6. Step 4 and 5 are iterated until all functional requirements in the release plan are implemented.

3.2 UI Oriented Requirement Elicitation Method

The key concept is that the user presents UI specification as well as functional requirements, and that based on the materials requirement prioritization is done between the requesters, developers and other participants.

Prototyping is well known as an approach for UI specification. It follows the process that after discussions are done by developers and requesters, developers create a prototype and present it to the clients. If the clients are not satisfied with the prototype, modifications occur. As in our approach, on the other hand, requesters themselves present UI specification. It is therefore expected the system is constructed as the requester imagine. Furthermore as described later, because functional requirements are discussed in relation to UI that presents view to realize them in our community ware, we think users are more understandable. In other words, both requesters and developers create requirements specification, which includes UI specification in a collaborative manner. Requirements specification in general software development is created by developer side as the result of extracting from the client side. On the other hand, in our community-ware, requirement specification is placed between requesters and developers, and both can modify it in the course of the requirement elicitation process.

We describe our UI oriented functional requirement specification process in more detail in association with our tool:

1) Descriptions of image for software to be developed: Requesters describe requirements for software. This description may be vague ideas the requester has, and not clear requirement specification. For example, initial requirement description from a requester in our preliminary experiment is as follows: proposed software is one that generates several kinds of seating charts for a school class. “I want a system which generates several kinds of seating charts for a school class. A user can select generation methods such as alphabetic order, male/female separation, male/female mixture, at random, lots drawing. The user can check the result on the display, revise the result by manual operation, and print out the chart.”

This description with respect to image for software is the source information for functional requirements. This information affects the subsequent processes.

2) Usecase extraction: The initial vague requirement specification needs to be clarified so that developers can deal with it. At the same time, the specification must be able to be understood by requesters even though they are not specialists in software development. We focus on the usecase of the UML [13] to solve this problem.

The usecase represents interaction between a system and users who participate in the system in a simple manner. It is expected that developers can understand requesters' mental model by grasping what requesters want to do by using the software as usecases. We will show this step by an example. Developers and requesters extract usecases from the document described in Step 1. The following is an example of a seating chart generation system.

* show the seating chart
* print out the seating chart
* read a list of names of a class
* revise the generated seating chart manually
* change the layout of seating

The usecase can be added by any participants.

3) Actor extraction: Requesters/developers extract actors. Actor means the roles users play in the system. In the seating chart generation system, we can extract a role, "user".

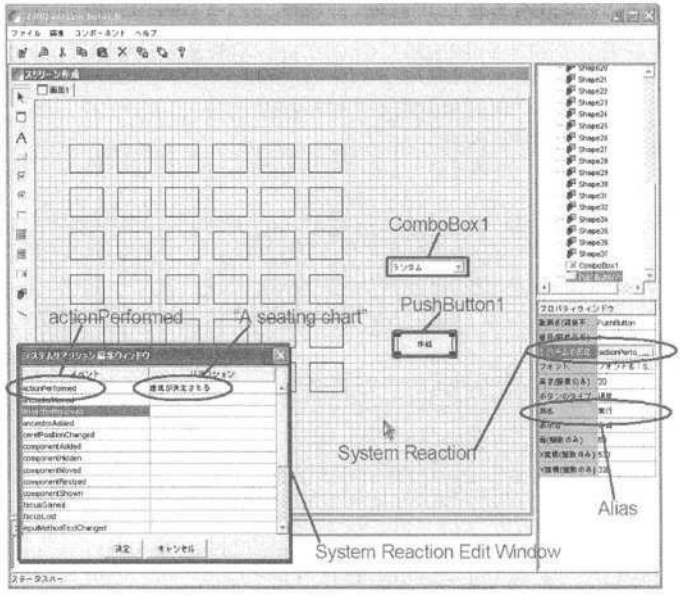


Fig. 2. UI image editor

4) UI design: Developers and requesters design screen images by using a UI image editor. This editor is an application software we developed for collaborative software

development community. Fig. 2 shows the screen image. Users create user interface of software by placing components such as a text box, a check box, and so on, on the frame window.

Once components placement has finished, then users describe the interactions of each component. This description enables the system to generate usecase descriptions semi-automatically (cf. step 5)).

We explain this by using Fig. 2. "Push button" in Fig. 2 has properties with respect to "alias" and "system reaction". "Alias" is represented as an alias of component name in the usecase description. We will define it as "execution". "System reaction" describes behavior of a system for an event of the component. In the example of the "push button", when an event "actionPerformed" occurs, behavior of the system is to describe "a seating chart is determined."

5) Write usecase description: Requesters and/or developers write usecase descriptions for extracted usecases. A usecase description is composed of scenario and its accompanying information. Although the structure for the usecase description is not specified by UML, we adopt five items, namely, usecase name, actor, pre-condition, main flow of scenario, and exceptional flow of scenario as the structure.

Pre-condition is a condition, which must be satisfied before executing this usecase. The pre-condition information is used at the release planning. The main flow is semi-automatically generated based on the information of UI design as described above.

First of all, a user selects the component an actor operates and the actions of the component. Then the system generates behavior of the action, which is specified as alias of the component. Furthermore if "system reaction" is defined, the system also generates behavior of the system corresponding to the behavior of the actor. The user performs this for all usecases.

We show an example main flow usecase description generation of "generate a seating chart" usecase as follows.

First of all, requesters/developers select "Combobox1" as the component the actor operates (see Fig. 3). Then they select "select" as the event. As the "Combobox1" is not defined "system reaction", the system generates the step "1. Select a seating chart generation method". We assume "seating chart generation method" is defined as "alias" of the "Combobox1". Next requesters/developers select "Pushbutton 1" as the component the actor operates and select "actionPerformed" as the event. We assume "Pushbutton 1" defines "generate a seating chart" as "system reaction". The system generates "2. The actor pushes execution button" and "3. The system generates a seating chart".

Following is an example of a usecase description of "generate a seating chart" usecase:

* usecase name: generate a seating chart
* actor: user
* pre-condition: read a list of names of a class
* main flow:
  1. The actor selects a "seating chart generation method"
  2. The actor notifies the system of the selection.
  3. The system generates the seating chart according to the selection method and displays the result.
  4. The actor notifies the system of the confirmation.
* exceptional flow: NONE

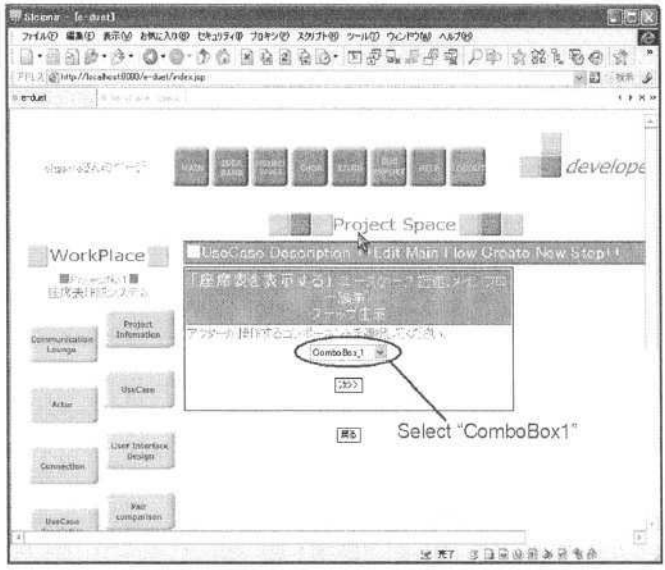


Fig. 3. Component selection

6) Input pair-wise comparison: The requesters input relative priority of a usecase with another one according to the evaluation criteria of value. The relative priority means the value of which a usecase is more valuable between two usecases and how much. This data is used in the release planning.

7) Input development duration: A developer inputs estimated duration for each usecase. The duration includes efforts for analysis, design, implementation and testing. We assume this data is determined according to developers' experience. This data is used in the release planning, too.

The above is the functional requirement elicitation process. A BBS is provided to support discussions during the process. Each usecase has its own BBS. We suppose the BBS is used to discuss with respect to the corresponding usecase.

A series of tasks are performed by requesters and developers in a collaborative manner through discussions. Another type of BBS is prepared to do meta discussions, for example, to add (delete) a usecase.

The system stores meta information per each usecase. This information and BBS messages play a role of context information for a functional requirements, therefore it is very important in requirement management [8]. Meta information includes the following items:

* usecase name
* author
* relative value compared with other usecases for a requester
* release ID this function is implemented

Fig. 4 shows the artifacts created in the functional requirements elicitation process and their relationships.

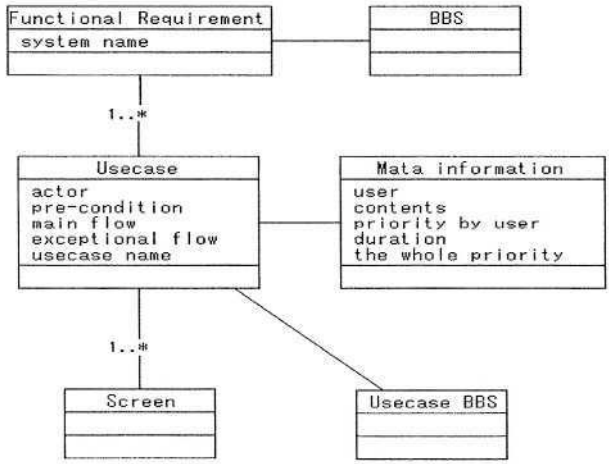


Fig. 4. Class diagram

3.3 Decision Process for Non-functional Requirements

After functional requirements are determined, we will move to determine non-functional requirements. This community-ware focuses on supporting to elicit software architecture and development conditions as non-functional requirements. Komiya et al. define budget, development duration, interface with other systems, software architecture, and user interface as non-functional requirements.

Our community-ware does not deal with budget because of education. Development duration will be obtained in the release planning. We have already described on UI. We integrate software architecture and interfaces with other systems into software architecture. Because of space limitation, we will report on this process elsewhere.

3.4 Release Planning

As mentioned before, we adopt the incremental development process model as a process model for collaborative software development. We propose priority assignment for each functional requirement and a grouping method of functions in release planning. We adopt the concept “Cost value approach” for decision of priority of functional requirements [6]. Overview of this approach is as follows: While requesters determine priority based on value of each function, developers do it based on development cost. The method suggests starting development from the functions with higher value from the requester side and with lower cost from the developer side.

Priority assignment process by AHP. To create a release plan, we have to assign

priority to functional requirements. As several requesters exist and each requester has his/her own priority for requirements in our collaborative development model, therefore priority for functional requirements among requesters must be coordinated. An evaluation criterion in the original cost-value approach by Karlsson and Ryan was a requester's value only [6]. On the other hand, we prepare the evaluation criteria for each requester. Namely, if requesters are three persons (person A, person B, and person C), the system prepares three evaluation criteria. At this time, weight for evaluation criteria must be determined. We decide the weight based on the degree of contribution for the requirement. By utilizing the degree of contribution, opinions of people who contributed the development a lot can be emphasized. The following is an equation to calculate the degree of contribution.

$$CV_i = cw\left(\frac{OC_i}{TC}\right) + (1-cw)\left(\frac{OR_i}{TR}\right)$$
$$\sum CV_i = 1$$

* CV_i is the degree of contribution by person *i.*
* TC is the total number of messages by all requesters in a BBS.
* OC_i is the total number of messages by person i in a BBS.
* TR is the total number of functional requirements.
* OR_i is the total number of functional requirements whose owner is person *i.*
* cw is a parameter of which has higher priority of messages in a BBS or proposal of functional requirements. cw is a real number between 0 and 1.

It is said that in AHP inconsistency may occur in pair-wise comparison when CI (The consistency index) of pair-wise comparison is more than 0.15. Therefore if the calculated CI is more than 0.15, then the system requires re-inputting pair-wise comparison. Cost-value approach used AHP in assigning priority to the developers' side. However we think it is more realistic and less effort for developers to estimate duration to implement each functional requirement than to estimate development costs. We therefore propose priority assignment method of a functional requirement from developer's side as follows:

$$DRP_j = \frac{1}{ET_j} \Big/ \frac{1}{TET}$$

DRP_j is a priority based on an estimated duration. ET_j is an estimated duration to implement a functional requirement, j. TET is the total sum ofl$/ET_j$.

Once priority for each functional requirement is determined, functional requirements are grouped based on their priority. Prioritization by AHP does not consider dependency relationships among functions at all. However dependency may exist among functions, which compose software. We propose a grouping method, which takes into account of dependency relationships among functions by using the

"pre-condition" of usecase descriptions.

We believe priority for functional requirements, which was assigned by the above mentioned method reflect on requirements of plural users and developers.

We show release planning process by illustrating an example. We introduce some variables as follows:

i: Variable which represents a requester
j: Variable which represents a functional requirement
RRP_{ij}: Variable which represents priority of a requester i for a functional requirement j
RRP_j: Variable which represents priority of all requesters for a functional requirement j

We assume the following five functional requirements were defined.

* show the seating chart (R1)
* print out the seating chart (R2)
* read a list of names of a class (R3)
* revise the generated seating chart manually (R4)
* change the layout of seating (R5)

We also assume three requesters A, B, and C participate in this theme and they contribute to the theme as shown in Table 1. Then TR becomes 5 and TC 70. We assume cw is 0.5. From these data, the degree of contribution for each requester is calculated as follows:

$$CV_A = 0.5 * (27/70) + 0.5 * (4/5) = 0.5926$$
$$CV_B = 0.5 * (36/70) + 0.5 * (1/5) = 0.3571$$
$$CV_C = 0.5 * (7/70) + 0.5 * (0/5) = 0.05$$

Table 1. Contribution of requesters

	No. messages on the BBS (OC)	No. functional requirements they created (OR)
Requester A	27	4
Requester B	36	1
Requester C	7	0

Each requester assigns the value of pair-wise comparison to five functional requirements according to his/her own evaluation criterion. Table 2 shows the result of pair-wise comparison by requester A.

Table 2. Result of pair-wise comparison by requester A

	R1	R2	R3	R4	R5
R1	1	3	1	3	5
R2	1/3	1	1/4	3	2
R3	1	4	1	6	5
R4	1/3	1/3	1/6	1	3
R5	1/5	1/2	1/5	1/3	1

Table 3. Result of pair-wise comparison among functional requirements by requesters

	R1	R2	R3	R4	R5
A	0.3187	0.1379	0.3962	0.0906	0.0566
B	0.3049	0.1362	0.2793	0.0692	0.2104
C	0.3502	0.2116	0.2822	0.0655	0.0904

We calculate priority from requesters by weighted sum using the degree of contribution and the data of Table 3. Table 4 shows the result. Priority of all requesters for R1 is calculated as follows:

$$RRP_{R1} = CV_A * 0.3187 + CV_B * 0.3049 + CV_c * 0.3502$$
$$= 0.3150$$

Table 4. Result of requirement prioritization by all requesters

	R1	R2	R3	R4	R5
RRP_j	0.3150	0.1409	0.3487	0.0817	0.1948

Developers assign estimated duration to each functional requirement based on their estimation. We assume Table 5 is the result.

Table 5. Estimated duration of each functional requirement by developers (unit: days)

	R1	R2	R3	R4	R5
ET_j	10	5	10	15	10

$$DRP_{R1} = (1/10) / \{(1/10 + (1/5) + (1/10) + (1/15) + (1/10))\}$$
$$= 0.1765$$

Table 6. Result of requirement prioritization by developers

	R1	R2	R3	R4	R5
DRP_j	0.1765	0.3529	0.1765	0.1176	0.1765

We could obtain the priority for all functional requirements from requesters and developers (Table 4 from requesters and Table 6 from developers). We normalize the results so that the total sum becomes 1. Table 7 shows the final result.

Table 7. Result of the final prioritization

Order of priority	Descriptions of a functional requirement	Priority
1	read a list of names of a class	0.2524
2	print out the seating chart	0.2373
3	show the seating chart	0.2361
4	change the layout of seating	0.1784
5	revise the generated seating chart manually	0.0958

Grouping Process. After priority of functional requirements is assigned, then grouping process of functional requirements is followed so that release schedule is met. One group is unit that is implemented in the one cycle of the incremental development process. Grouping is done from functional requirements with higher priority to those with lower priority. Priority assignment process by AHP does not consider dependency relationships among functional requirements. However when some software is composed of plural functions, they usually have dependency relationship. We therefore propose a grouping process of functional requirements, which take the dependency relationship among functional requirements into consideration. To attain this goal, we use pre-condition use cases as follows:

1) Functional requirements are grouped so that each group does not exceed the duration of one cycle and functional requirements with higher priority are implemented sooner (in this example we assume it is 21 days) according to the result of final prioritization. An example result is shown in Table 8.
2) As use case “print out the seating chart” in group 1 specify pre-condition use case “show the seating chart”, adjustment should be done so that the pre-condition belongs to the same group or it precedes. Table 9 shows the result.
3) If the result of 2) exceeds to the cycle time, adjustment should be done by moving some of the group to the next group. The step is iterated until all groups satisfy with the pre-condition relation and the cycle time. As the result of this example does not exceed to the cycle time, Table 9 can be an example of final result.

Table 8. Result of initial grouping

Group ID	Descriptions of functional requirement	Duration
Group 1	read a list of names of a class print out the seating chart	15 days
Group 2	show the seating chart	10 days
Group 3	change the layout of seating	10 days
Group 4	revise the generated seating chart manually	15 days

Table 9. Final result of grouping

Group ID	Descriptions of functional requirement	Duration
Group 1	read a list of names of a class	10 days
Group 2	print out the seating chart show the seating chart	15 days
Group 3	revise the generated seating chart manually	15 days
Group 4	Change the layout of seating	10 days

4 Conclusion

This paper has proposed a user interface oriented requirement elicitation method. Based on the functional requirements, which were obtained by the method, we have also proposed requirement prioritization method over showing an example. The method is an extension of "Cost-value approach" to our community model.

We plan to conduct an experiment to evaluate effectiveness of our approach and tool.

Acknowledgements. The study is supported by the Grant-in Aid for No. C (2) 14580209 from The Ministry of Education, Science, Sports and Culture of Japan.

References

1. Bruegge, B., Dutoit, A.H., Kobylinski, R., Teubner, G.: Transatlantic Project Courses in a University Environment. Proceedings of the Seventh Asia Pacific Software Engineering Conference. IEEE CS Press (2000) 30-37,.
2. Curtis, B., Krasner, H., Iscoe, N.: A Field Study of the Software Design Process for Large Systems. Communications of the ACM, Vol.31, No.11. (1988) 1268-1287
3. Davis, A. M.: A Strategy for Comparing Alternative Software Development Life Cycle Models. IEEE Transactions on Software Engineering, Vol. 14, No. 10. (1988) 1453-1461

4. Deifel, B.: A Process Model for Requirements Engineering of COTS. Proceedings of the DEXA. IEEE CS Press (1999) 316-320
5. Proceedings of the International Workshop on Global Software Development (2003) http://GSD2003.cs.uvic.ca.
6. Karlsson, J. Ryan, K.: A Cost-Value Approach for Prioritizing Requirements. IEEE Software, Vol. 14, No. 5 (1997) 67-74
7. Komiya, S., Kato, J., Nagata, M., Ohnishi, A., Saeki, M., Yamamoto, S., Horai, H.: A Realization Method of the System to Navigate Requirements Elicitation Work Driven by Interviews. IPSJ SIGNotes SE121-13 (1998) 99-106 (In Japanese).
8. Lang, M., Duggan, J.: A Tool to Support Collaborative Software Requirements Management. Requirements Engineering, No. 6, Springer-Verlag. (2001) 161-172
9. Raymond, E.: The Cathedral and Bazaar, http://www.tuxedo.org/~esr/writings/cathedral-bazaar.
10. Regnell, B., Beremark, P., Eklundh, O.: A Market-driven Requirements Engineering Process - Results from an Industrial Process Improvement Programme. Proceedings of Conference on European Industrial Requirements Engineering (CEIRE'98). (1998)
11. Saaty, T. L.: The Analytic Hierarchy Process. Procededings of the International Symposium of the Analytic Hierarchy Process. (1999)
12. Shimakage, M., Hazeyama, A., Ishide, T.: Collaborative Software Development Communityware and Its Prelimenary Applicatiuon. Proceedings of the First International Conference on Knowledge Economy and Development of Science and Technology. (2003)
13. http://www.omg.org/uml/.

Development of a Normative Package for Safety-Critical Software Using Formal Regulatory Requirements

Sergiy A. Vilkomir and Aditya K. Ghose

Decision Systems Lab, School of IT and Computer Science
University of Wollongong, NSW 2522, Australia
{*sergiy, aditya*}@ *uow.edu.au*

Abstract. Important tasks in requirement engineering are resolving requirements inconsistencies between regulators and developers of safety-critical computer systems, and the validation of regulatory requirements. This paper proposes a new approach to the regulatory process, including formulating requirements and elaborating methods for their assessment. We address the differences between prescriptive and nonprescriptive regulation, and suggest a middle approach. Also introduced is the notion of a normative package as the collection of documents to be used by a regulator and provided to a developer. It is argued that the normative package should include not only regulatory requirements but also methods of their assessment. We propose the use of formal regulatory requirements as a basis for development of software assessment methods. This approach is illustrated with examples of requirements for protecting computer control systems against unauthorized access, using the Z notation as the method of formalization.

1 Introduction

Given a diverse collection of stakeholders, there is considerable interest in the requirements engineering community in supporting the tasks of eliciting, analysing and negotiating requirements. Mainly the stakeholders are developers, users and customers of computer systems. But for safety-critical computer systems, there is another important stakeholder - a Safety Regulatory Authority, whose role has been studied in far less detail.

For the following reasons, issues such as handling inconsistency and negotiating requirements between regulators and developers pose special challenges:

- Regulatory goals and requirements have a general nature and are applied to large classes of systems.
- Regulatory requirements are often mandatory for developers and users.

Investigations of these tasks have important practical value. Theoretical and practical problems connected with regulatory activities have been considered mainly for individual industries such as the aircraft industry [11], nuclear energetics [9], railway transport [8], etc. In this paper we suggest some approaches which could be applied irrespective of the specific industry.

The object of this paper is to define phases of the regulatory process including formulating requirements and elaborating methods for their assessment. We consider these

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 523–537, 2004.

issues from the regulator's point of view and for the purpose of enhancing further collaborating between the regulator and the developer. Particularly, we suggest the approaches for formulating requirements, assessing requirements, and providing information to developers. Other goals are to:

- Assist developers in understanding safety goals and requirements.
- Eliminate future conflicts with developer as early as possible.
- Facilitate future safety assessment.
- And, on the whole, increase the safety level of the system and software.

Part of the issue of formulating regulatory requirements is to address the relevant standards. Problems during elaboration of software standards have been studied for groups of standards [10,7,6,5,21] as well as for separate ones, for example IEC 60880 [15] in [25], IEC 61508 [14] in [3], RTCA/DO-178B [23] in [4]. However, taking into account the poor quality of some standards (dealt with in Section 3.1), we consider that this direction is important, topical and requiring further investigation.

This paper is structured so that in Section 2 we consider the different participants of the regulatory process. The definitions and sources of regulatory requirements are also considered. Section 3 presents different approaches in the regulatory process. Differences between prescriptive and nonprescriptive regulation are addressed and a middle approach is suggested. Then we introduce the notion of a *normative package,* the collection of documents to be generated by a regulator and provided to a developer. We argue that the normative package should include not only regulatory requirements but also methods for their assessment in clear structural form. In Section 4 we propose using formal regulatory requirements for the development of software assessment methods. The key idea is to develop formal regulatory requirements from requirements in a natural language as a first step, and to translate the formal requirements thus generated to assessment checklists. Such checklists can be used to easily (and manually) ensure that eventual designs and implementations satisfy the regulatory requirements. We illustrate this approach considering an example of requirements for protection of computer control systems against unauthorized access, using the Z notation [24] as a method of formalization.

General conclusions and directions for future work are addressed in Section 5. Appendices A and B contain respectively the checklist and correspondence between the checklist and the Z schemas for an additional example involving requirements for functional redundancy. The formalization of this requirement has been developed previously in [29].

2 Main Notions

2.1 Regulator

A key participant in the safety regulatory process is a Regulatory Body (Regulatory Authority). The typical definition of a regulatory body is "an authority or a system of authorities designated by the government of a State as having legal authority for conducting the regulatory process" [12]. However, organization, structure and approaches of the

regulatory process (which is understood broadly and includes licensing, certification, safety assessment and similar activities) could vary substantially.

These variations are explained by:

- Different legal and political aspects in different countries.
- Particular features of key safety-critical industries.
- Differences in maturity of the same industry in various countries.

The last reason could be illustrated for example the nuclear power sector. Nuclear safety regulatory bodies have the same aims and tasks for all countries but it is natural that the organization and approaches in such countries as USA (104 nuclear reactors in operation), France (59 reactors) and Japan (54 reactors) are different in comparison with countries that have one or two reactors in operation.

Often a regulatory body does not carry out a safety assessment itself but entrusts it to another organizations. Three variants typically exist:

- Technical support organizations.
- Independent safety assessors (companies).
- Independent safety auditors (individuals).

A system of technical support organizations for regulatory bodies has been established in many countries, especially in the nuclear power sector. These independent organizations were established as permanent partners of a regulatory body and given specific scientific and technical tasks during licensing processes. Independent safety assessors, companies with a considerable experience in the safety analysis, could be recruited for assessment of specific systems and for applied research in safety as well. Collaboration with independent safety auditors is described, for example, in the UK Ministry of Defence 00-56 standard (section 5.3.4) [28].

In this paper we consider mainly methodological and technical issues of the regulatory process. That is why we consider the notion of regulator in a generalized sense, including all regulators' agents as already described. In the rest of this paper a regulator is an organization or an individual authorized to conduct safety assessments during regulatory process. In our case - safety assessment of software for safety-critical computer systems.

2.2 Regulatory Requirements

The sources of requirements for software, which are used during regulatory process, include the following:

- Documents of a regulatory body, for example rules, guides, internal standards.
- National standards, for example ANS standards in USA.
- International standards, for example standards of the International Electrotechnical Commission (IEC) and the International Organization for Standardization (ISO).
- Documents of international agencies such as the International Atomic Energy Agency (IAEA) and the European Space Agency (ESA).

Several standards or other sources of requirements could be used simultaneously for safety assessment of a specific system during a regulatory process. For example, the following documents could be used together for assessment of software of computer system at a nuclear power plant: IEEE Std 7-4.3.2-1993 [16], IEC 60880 [15], IAEA Safety Guide NS-G-1.1 [13]. It is possible that in some situations a regulator uses only a part of requirements from one source.

We consider a requirement for software as a regulatory requirement if a regulator took and documented a decision to use this requirement for software assessment. The regulator can decide to use a whole standard or only a part of requirements from the standard. The regulator can also decide to use a standard not for all systems but only for specific ones. The same requirement from a standard can be considered as a regulatory requirement in one situation (for one system), and not be considered as a regulatory one in another situation (for another system), depending on the regulator using a standard. So our understanding of a regulatory requirement is also quite general and is independent from the source of this requirement. But it is important that regulatory requirements should be documented and approved in the beginning of a regulatory process.

3 Normative Package for Safety-Critical Software

3.1 Different Approaches to the Regulatory Process

A key problem in the preliminary stage of the regulatory process is devising an appropriate strategy for formulating regulatory requirements. Looking at the large number of existent software standards, it would seem this problem should have been solved long ago, yet this is not the case. The reason is an inadequate quality of some standards. As mentioned in [19] and some other papers, there is widespread criticism of current safety standards. In particular, Fenton and Neil [10] claimed, "software standards are written in such a way that we could never determine whether they were effective or not". They suggested the following shortcomings:

- Overemphasis on process rather than product.
- Imprecise requirements.
- Non-consensus recommendations.
- Lack of clarity on risk and benefits.
- Standards are too big and poorly organized.

Emmet and Bloomfield criticized the typical standards formulation process, which was too long and resulted in out-of-date standards [7]. Emmerich et. al considered the problem of managing standards compliance using as examples some well-established software engineering standards which "are often incomplete and ambiguous" [6]. McDermid and Pumfrey claim that most software standards "are not standards" compared with standards in other industries and engineering disciplines [5].

To decide how to formulate software regulatory requirements, it is necessary to make a decision on a "philosophy" of a regulatory process. There are two contrary approaches to the regulatory process and correspondingly to development of standards: prescriptive and nonprescriptive.

The first one establishes strict mandatory requirements and is traditional for engineering (hardware) standards. But its use for software can be problematic. It can be inflexible and set up constraints on innovation [10]. The nonprescriptive approach assigns general safety goals and requirements for computer systems and software behaviour, but does not specify ways of achieving these goals. So a developer profits from the freedom to choose technical solutions but bears full responsibility for the safety of the resulting system. A developer is required to provide evidence that all safety goals and requirements are satisfied, and often is obliged to elaborate a special document (a Safety Case) for this purpose.

A general tendency to move towards nonprescriptive regulation can be seen in the aircraft industry. (see, for example, [21] where this approach is determined as "goal-based") and in nuclear industry (see [20] where it was called "performance-based regulation"). However, we should notice that although the pure nonprescriptive approach is convenient for a regulator, it could be unsuitable for a developer. The problem stands from the fact that the very general nature of the resulting formulation of regulatory requirements might lead to ambiguity. Thus, a developer may assume the implemented system to have satisfied the requirements, yet later finds the regulatory authority to be in disagreement. The inconsistency is detected at a late stage of a development and its resolution could be very expensive and laborious.

To avoid this problem, we suggest a "middle" approach. Although identifying a balance between "general enough" and "detailed enough" is not a simple task. A requirement could be considered as well formulated ("detailed enough") if [10]:

- It is clear what is required in order to conform to the requirement from a developer's point of view.
- It is possible to determine conformance to the requirement from a regulator's point of view.

To be nonprescriptive at the same time, we need to formulate requirements at a level of abstraction and generality that does not force commitment to specific hardware platforms, programming languages, methods of development, etc. There is no common recipe for this. Vilkomir and Bowen considered one possible technique using a formalization of regulatory requirements in the Z notation [29]. Some general issues of applying formal methods to standards have been considered in [1].

Regulatory requirements should be available for a developer prior to the start of the development process. But this alone is not sufficient. The next section introduces the notion of a *normative package* that, we argue, addresses the information needs of developers at the appropriate level of generality.

3.2 Structure of a Normative Package

We define a normative package as a collection of documents (standards, rules, guides, regulation, instructions, etc.) that contains information which will be used by a regulator during the regulatory process for a specific system. The list of these documents should be officially approved and be available for a developer.

The main aim of establishing the normative package is to provide the developer with all necessary information before or at the early stages of the development. Familiarization

of the developer with all "rules" of the regulatory process in advance helps to eliminate possible inconsistencies, misunderstandings and disagreements between the regulator and the developer as early as possible. It also facilitates further safety assessment. As well, it can exert real influence upon the increase of system or software safety because the developer takes into consideration all safety goals and requirements from the beginning of his activities.

The main contents of the normative package are formed by standards and other documents with regulatory requirements. But, in contrast to what is used in the standardization notion of standards profile [17], we claim that the normative package should contain documents with the following additional information: explanations, rationale or details of software requirements, and regulator's methods of safety assessment.

Requirements in standards are typically described in a brief and terse manner so supplement of additional explanations or details of requirements is a frequent practice. It could be provided in different ways:

- As a special appendix in the same standard. For example, IEC-60880 standard [15] contains Appendix A2 "Details of software requirements".
- As a separate standard. For example, UK Def Stan 00-55 contains two separate parts - Part 1 [26] with requirements for safety related software in defence equipment and Part 2 [27] with guidance on these requirements.
- As a regulator's guide which endorses and details the corresponding standard.

The last way is typical, for example, for US Nuclear Regulatory Commission (NRC). Thus, NRC Regulatory Guide 1.152 [22] clarifies requirements from IEEE Std 7-4.3.2-1993 [16], etc.

Of course, a part of regulatory requirements can be so simple and/or specific that there is no problem for their explanation and assessment. But another part of regulatory requirements can be more complicated and/or general and only these requirements will we consider further. Even if all such requirements are explained in detail and the regulator and the developer negotiated all possible contradictions in requirements' understanding, inconsistencies could arise during the process of software safety assessment.

That is why it is very important to have information in the normative package about regulator's methods of safety assessment, including methods of conformance assessment for each requirement. This information could be stated in internal regulator's documents. Some general approaches to software safety assessment have been considered in [30,31]. But for the normative package it is desirable to describe methods of assessment in the clear structural (procedural) form for each requirement. For example, in the form of checklists, questionnaires or procedures. It gives the developer the possibility of carrying out the self-assessment of his product and preventing possible future inconsistency.

To summarize, the idea of the normative package is not only that the regulator should have documented regulatory requirements, their explanations and methods of assessment. The idea is that all these documents should be combined in one package and provided (or be available) to the developer as an instruction for the further regulatory process.

The regulator may use the unified normative package for all computer systems and their software or, using the typical normative package as a basis, create the distinctive one for each specific system. The latter variant is used when the regulator has to apply

different approaches to safety assessment, for example because of differences of systems' safety levels. In this case, the normative package is created after receiving a request and sufficient information on the planned system from the developer. The regulator should firstly create a profile with requirements for the concrete system (some approaches addressed in [18]) and then combine the requirements and methods of their estimation.

The creation of a normative package demands considerable efforts. The most difficult problem is elaboration of methods of assessment for each requirement. A possible approach to solving this problem is considered now in Section 4.

4 Using Formal Regulatory Requirements as a Technic for Development of Software Assessment Methodology

4.1 General Approach

We have argued above for the need for a document detailing the software safety assessment method in addition to the more generally formulated regulatory requirements. We propose to use *checklists,* which describe the assessment process step-by-step. Checklists for regulatory requirements validation could be used in parallel with the analysis of safety arguments provided by the developer.

We propose using formal methods for the development of checklists. The idea is illustrated at Fig. 1.

The lower part of Fig. 1 shows the general organization of the regulatory process:

- The normative package (including checklists) is provided to the developer.
- The developer implements the proposed system knowing in advance all regulatory requirements and methods for their assessment (checklists).
- The regulator validates the implemented system using the checklists contained in the normative package.

The upper part of Fig. 1 shows internal activities of the regulator for elaborating checklists at the preparatory phase before the beginning of the regulatory process:

- The regulator formalizes the regulatory requirements.
- Using the formal requirements as a basis, the checklists are elaborated.

It is important to understand the motivation for deriving checklists from formally-specified regulatory requirements instead of directly obtaining them from the initial description of these requirements. The process of formalizing the initial informally-specified (and potentially imprecise) regulatory requirements forces the regulatory authority to add the necessary level of precision and detail required to generate meaning for assessment checklists.

The first step, the development of formal regulatory requirements, has been addressed in [29]. Extending this approach, we consider now how to use these formal requirements for elaboration of detailed methods of requirements assessment. The process of the checklists elaboration is in some sense the inverse of the process of formalization of requirements and corresponds to translations from formal requirements to informal

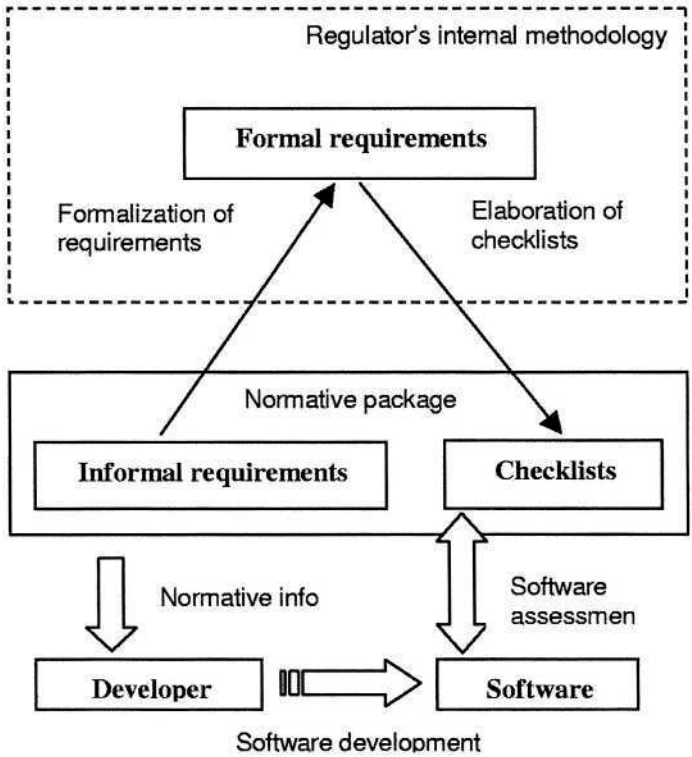


Fig. 1. Using formal methods for the development of checklists

(natural language) checklists. The specific formal method being used is not critical. Our subsequent discussion focuses on the Z notation, but the key elements of our proposal would hold equally well if a different formal method were used.

One of the advantages of formal requirements is that they can be verified mechanically. Using appropriate tools, it is possible to verify correctness, consistency and completeness of formal requirements. It also makes it possible to check the completeness of checklists created from these formal requirements. To achieve this, it is necessary to establish a one-to-one correspondence between items in the checklist and components of the formal specification. If the completeness of a formal requirement has been proved and one or several items of a checklist correspond to each object and each constraint of a formal requirement, then there are grounds to have increased confidence of the completeness of the checklist as well.

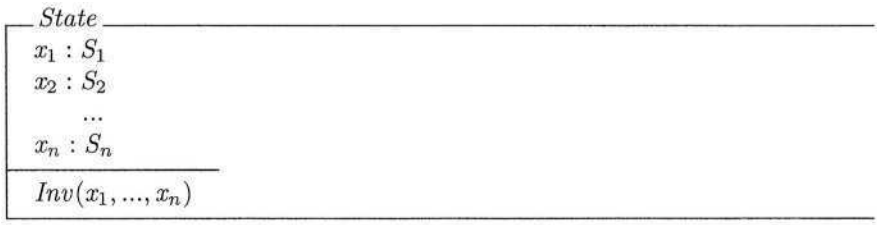
It is important to be able to systematically generate checklists from formal specifications. Given that formal requirements tend to be moderate-sized, it is feasible to discuss

methodologies for doing this manually. We present the outlines of such a methodology in the next section, applicable to formal specifications written in Z.

4.2 Methodology Outline

One of the possible formal methods for elaboration of formal regulatory requirements and correspondingly checklists is the Z notation [24] which is widely used for specifying and designing computer systems and software. Z includes the *schema* notation for structuring specifications so each regulatory requirement can be represented as one or several schemas.

Consider the generalized form of schemas [2]



where *State* is a name of the schema. The first part of the schema *(a signature)* contains *components* $x_1, ..., x_n$ which are the state variables with types $S_1, ..., S_n$ respectively. The second part of the schema (*a predicate*) contains the predicate *Inv (an invariant)* which describe properties that are always true.

When the checklist is elaborated on the basis of the Z schema, it should contain:

- An item for each component of the schema. These items should check whether the components are reflected in software or in software documentations.
- An item for each predicate of the schema. These items should check whether the constraints are satisfied.

An exception is a *normalized* form of the schema when the signature contains the most general types for components and the actual types are determined by constraints in the predicate part of the schema. In this situation, it is not necessary to have separate items in the checklist for a component and for a predicate with its actual type. It is sufficient to mention the actual type in the item, which ensures that the component is reflected. The example of the checklist for the requirements in the form of normalized schema is considered in Section 4.3.

The regulatory requirements could demand from the system or software to mandatory execute some operations (functions). To formalize such requirement in Z, an *operation* schema [2] is used:

Operation

$x_1 : S_1;\ ...;\ x_n : S_n$
$x'_1 : S_1;\ ...;\ x'_n : S_n$
$i_1? : T_1;\ ...;\ i_m? : T_m$
$o_1! : U_1;\ ...;\ o_p! : U_p$

$Pre(i_1?, ..., i_m?, x_1, ..., x_n)$
$Inv(x_1, ..., x_n)$
$Inv(x'_1, ..., x'_n)$
$Op(i_1?, ..., i_m?, x_1, ..., x_n, x'_1, ..., x'_n, o_1!, ..., o_p!)$

where x_j represents the state variable before the operation, x'_j represents the state variable after the operation, $i_j?$ is the input and $o_j!$ is the output. The predicate part of the operation schema contains invariants as well as a *precondition* (describes what must be true before the operation) and a *postcondition* (describes what must be true after the operation).

In this case, the aim of the checklist is to verify the realization of the operation. The methodology of checklist elaboration from the operation schema is similar to that for the state schema: the checklist should contain an item for each component (for each state, input and output variable) and an item for each predicate (for each invariant, precondition and postcondition).

It is advisable to formulate checklist items in a uniform appearance. This task is relatively easy for items corresponding with the signature part of the schema. These items check weather a component of the schema is "determined", or "established", or "assigned", etc. Items corresponding to the predicate part of the schema simply describe formal constraints in natural language. Examples of the checklist items for the non-trivial constraints are in Appendices A and B.

To verify completeness of a checklist, we perform an inverse mapping from checklist items to schema components and predicates. We annotate each Z schema component with an item number from the checklist. If every component has a corresponding number we may conclude that the checklist is complete.

4.3 An Example

An example of a formal regulatory requirement for protecting computer control systems from unauthorized access was first discussed in [29]. We use the same example here with a slightly different version of the Z schema describing the formal requirement. It is necessary to note that this simple example has been chosen not as the only possible variant of the requirement concerning unauthorized access but mainly to illustrate the proposed approach to formalization. On the basis of this example we elaborate the checklists and show the correspondence between items of the checklist and elements of the Z schema.

Various specialists (system engineers, safety engineers, programmers, technologists, operators, etc.) take part in the maintenance and use of a safety-critical computer control system. These specialists perform various tasks such as reading information, changing a program code, changing alarm setting, changing operational parameters, etc. In this case, every member of staff should have an access to fulfil only actions specified in his job description to avoid unpremeditated mistakes.

To formalize, consider the following given sets:

[*POST, ACTION, LEVEL*]

where *POST* contains possible posts of staff using a control system, *ACTION* contains possible actions of staff while using the system, and *LEVEL* contains possible levels of access. The Z schema below determines these notions for every specific system explicitly, namely the list of staff posts (set *posts*), the list of actions (set *actions*), and the list of levels of access (set *levels*). Furthermore, a level of access should be assigned for every member of staff (function *post_level*) and a list of authorized actions should be assigned for every level of access (function *level_action*).

Access

$$
\begin{aligned}
&posts : \mathbb{P}_1\ POST \\
&actions : \mathbb{P}_1\ ACTION \\
&levels : \mathbb{P}_1\ LEVEL \\
&post\_level : POST \twoheadrightarrow LEVEL \\
&level\_action : LEVEL \twoheadrightarrow \mathbb{P}_1\ ACTION \\
\hline
&\mathrm{dom}\ post\_level = posts \\
&\mathrm{ran}\ post\_level = levels \\
&\mathrm{dom}\ level\_action = levels \\
&\mathrm{ran}\ level\_action \subseteq \mathbb{P}_1\ actions
\end{aligned}
$$

Now let us create the checklist according the Z schema. There are five objects in this schema - three sets and two functions. So we include five items in the checklist, one item for each object. Because the predicate part of the Z schema only makes more precise ranges and domains of the functions and does not contains real restrictions, it is not necessary include separate items for this part. So, the checklist for the requirements for protection against unauthorized access could be as follows:

1. The list of posts of personnel is established with all posts having authorized access to a system.
2. The list of all possible actions (operation) during working with a system is established.
3. The scale of all different levels of access to a system is established.
4. The level of access is assigned for each post of personnel.
5. The unique subset of authorized actions is assigned for each level of access.

To verify completeness of the checklist, consider mapping between the checklist and the Z schema. For that, as it was considered in Section 4.2, parentheses with the number of the item of the checklist are put in the beginning of the every line of the Z schema.

Access

(1) $posts : \mathbb{P}_1\ POST$
(2) $actions : \mathbb{P}_1\ ACTION$
(3) $levels : \mathbb{P}_1\ LEVEL$
(4) $post\_level : POST \twoheadrightarrow LEVEL$
(5) $level\_action : LEVEL \rightarrowtail\!\!\!\!\!\!\rightarrow \mathbb{P}_1\ ACTION$

---

(4) $\mathrm{dom}\ post\_level = posts$
(4) $\mathrm{ran}\ post\_level = levels$
(5) $\mathrm{dom}\ level\_action = levels$
(5) $\mathrm{ran}\ level\_action \subseteq \mathbb{P}_1\ actions$

The mapping is simple for the given example but could demonstrate completeness for more complicated regulatory requirements. Such an example for the requirement for protection against common mode failures by using the functional redundancy has been considered previously. The description of the requirements can be found in [29] and here we only create the checklist (see Appendix A) and show the correspondence between the checklist and the formal requirement (see Appendix B).

5 Conclusion and Future Work

In this paper, we present an approach to the regulatory assessment of safety-critical systems and software in order to facilitate handling inconsistency between a regulator and a developer. We suggest the following:

- Using a middle (between prescriptive and nonprescriptive) regulatory approach.
- Providing all necessary information to the developer in the form of a unified normative package before or at the early stages of the development.
- Including in the normative package not only regulatory requirements but also detailed methods of their assessment.
- Using formal regulatory requirements as a basis for development of software assessment methods.

As an illustration of the proposed approach we consider eliciting the checklist from the specific formal requirements. As an example we use requirements in the Z notation for protection of computer control system against unauthorized access.

One of the possible directions for future work is determination of groups of requirements for which the suggested approach is the most effective. Another important direction is the choice of the most effective method of formalization and automation of translation from formal requirements to checklists.

References

1. Blyth, D., Boldyreff, C., Ruggles, C., Tetteh-Lartey, N.: The case for formal methods in standards. IEEE Software, Volume 7, Issue 5 (1990) 65–67

2. Bowen, J.P.: Formal Specification and Documentation Using Z: A Case Study Approach. International Thomson Computer Press (1996)
3. Brown, S.: Overview of IEC 61508. Design of electrical/electronic/programmable electronic safety-related systems. Computing & Control Engineering Journal, Vol. 11, Issue 1 (2000) 6–12
4. Cortellessa, V., Cukic, B., Mili, A., Shereshevsky, M., Sandhu, H., Del Gobbo, D., Napolitano, M.: Certifying Adaptive Flight Control Software. Proceedings of the ISACC2000 - The Software Risk Management Conference, Reston, VA, USA (2000)
5. McDermid, J.A., Pumfrey, D.J.: Software Safety: Why is there no Consensus? Proceedings of the 19th International System Safety Conference, Huntsville, AL, USA (2001)
6. Emmerich, W., Finkelstein, A., Montangero, C., Antonelli, S., Armitage, S., Stevens, R.: Managing standards compliance. IEEE Transactions on Software Engineering, Vol. 25, Issue 6 (1999) 836–851
7. Emmet, L., Bloomfield, R.: Viewpoints on Improving the Standards Making Process: Document Factory or Consensus Management? Proceedings of the Third International Software Engineering Standards Symposium (ISSES 97), Walnut Creek, California, USA (1997)
8. Eriksson L-H.: Specifying Railway Interlocking Requirements for Practical Use. Proceedings of the 15th International Conference on Computer Safety, Reliability and Security (SAFECOMP'96), Vienna, Austria (1996)
9. European Commission. Nuclear Safety and Environment. Common position of European nuclear regulators for the licensing of safety critical software for nuclear reactors. Report EUR 19265 (2000)
10. Fenton, N.E., Neil, M.: A strategy for improving safety related software engineering standards. IEEE Transactions on Software Engineering, Vol. 24, Issue 11 (1998) 1002–1013
11. Hayhurst, K.J., Holloway, C.M.: Challenges in software aspects of aerospace systems. Proceedings of 26th Annual NASA Goddard Software Engineering Workshop (IEEE/NASA SEW-26 2001), Greenbelt, MD, USA (2001) 7–13
12. IAEA Safety Standards Series No. GS-R-1. Legal and Governmental Infrastructure for Nuclear, Radiation, Radioactive Waste and Transport Safety: Safety Requirements. International Atomic Energy Agency, Vienna (2000)
13. IAEA Safety Standards Series No. NS-G-1.1. Software for Computer Based Systems Important to Safety in Nuclear Power Plants. Safety Guide. International Atomic Energy Agency, Vienna (2000)
14. IEC 61508. Functional safety of electrical/electronic/ programmable electronic safety-related systems. Part 3: Software requirements. International Electrotechnical Commission (1998)
15. IEC 60880. Software for computers in the safety systems of nuclear power stations. Edn.: 1.0, International Electrotechnical Commission (1986)
16. IEEE Std 7-4.3.2-1993. IEEE Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations (1994)
17. ISO/IEC TR 10000-1:1998. Information technology – Framework and taxonomy of International Standardized Profiles – Part 1: General principles and documentation framework. 4th edn. (1998)
18. Kharchenko, V.S., Shostak, I. V., Manzhos, Y.S.: The Intelligent System for Licensing Critical Software. Aerospace Engineering and Technologies, No. 4 (2002) 46–51 (in Russian)
19. Lutz, R.: Software Engineering for Safety: A Roadmap, Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000), Limerick, Ireland, ACM (2000)
20. NUREG BR-0303. Guidance for Performance-Based Regulation. Prepared by N.P. Kadambi, U.S. Nuclear Regulatory Commission, Washington, DC, USA (2002)
21. Penny, J., Eaton, A., Bishop, P.G., Bloomfield, A.E.: The Practicalities of Goal-Based Safety Regulation. Proceedings of the Ninth Safety-Critical Systems Symposium, Bristol, UK (2001) 35–48

22. Regulatory Guide 1.152. Criteria for Digital Computers in Safety Systems of Nuclear Power Plants. Revision 1, U.S. Nuclear Regulatory Commission, Washington, DC, USA (1996)
23. RTCA/DO-178B. Software Considerations in Airborne Systems and Equipment Certification. RTCA, Washington DC, USA (1992)
24. Spivey, J.M.: The Z Notation: A Reference Manual. Prentice Hall International Series in Computer Science, 2nd edn. (1992)
25. Thuy, N.N.Q., Ficheux-Vapne, F: IEC 880: feedback of experience and guidelines for future work. Proceedings of Second IEEE International Software Engineering Standards Symposium (ISESS'95) (1995) 117–126
26. UK Def Stan 00-55 (Part 1)/Issue 2. Requirements for Safety Related Software in Defence Equipment. Part 1: Requirements (1997)
27. UK Def Stan 00-55 (Part 2)/Issue 2. Requirements for Safety Related Software in Defence Equipment. Part 2: Guidance (1997)
28. UK Def Stan 00-56 (Part 1)/Issue 2. Safety Management Requirements for Defence Systems. Part 1: Requirements (1996)
29. Vilkomir, S.A., Bowen, J.P.: Establishing Formal Regulatory Requirements or Safety-Critical Software Certification. Proceedings of AQuIS 2002: 5th International Conference on Achieving Quality In Software and SPICE 2002: 2nd International Conference on Software Process Improvement and Capability Determination, Venice, Italy (2002) 7–18
Available: http://www.uow.edu.au/~sergiy/aquis2002.pdf
30. Vilkomir, S. A., Kharchenko, V.S.: An "Asymmetric" Approach to the Assessment of Safety-Critical Software During Certification and Licensing. Proceedings of ESCOM-SCOPE 2000 Conference, Munich, Germany (2000) 467–475
31. Vilkomir, S.A., Kharchenko, V.S.: Methodology of the review of software for safety important systems. Proceedings of ESREL'99 - The Tenth European Conference on Safety and Reliability, Vol. 1, Munich-Garching, Germany (1999) 593–596

A The Checklist for the Requirement for Functional Redundancy

1. The criteria are used to determine all emergency situations when the actuation of the system is required.
   a) The set of physical and/or technological parameters is assigned for each criterion.
   b) Each parameter has the identifier and the range of its variation in time is determined.
   c) According to a criterion, the system actuation takes place under a specific combination of parameters' values.
2. The set of main criteria is established which covers all possible emergency situations.
   a) Time of the admissible delay of the system actuation after the determination of an emergency situation is established for the each main criterion.
3. The set of additional criteria which give other ways to determine all emergency situations is established.
   a) Each main criterion has a corresponded additional criterion.
4. Sets of parameters for the each main criterion and the corresponding additional criterion are disjoint.
5. Time between the emergency situation appearance according the main criterion and the system actuation according the additional criterion is determined, and is less than time of the admissible delay.

B Correspondence between the Checklist and the Formal Requirement for Functional Redundancy

parameter

$$
\begin{array}{l}
(1b)\ ident : IDENTIFIER \\
(1b)\ areaparam : \mathbb{P}_1\, \mathbb{R} \\
(1b)\ valueparam : Time \rightarrow \mathbb{R} \\
\hline
(1b)\ \mathrm{ran}\ valueparam \subseteq areaparam
\end{array}
$$

criterion

$$
\begin{array}{l}
(1a)\ seqparam : \mathrm{seq}_1\ parameter \\
(1a)\ num : \mathbb{N}_1 \\
(1)\ statecrit : Time \twoheadrightarrow Command \\
(1c)\ areacrit : \mathbb{P}_1(\mathrm{seq}_1\, \mathbb{R}) \\
\hline
(1a)\ \#seqparam = num \\
(1c)\ \forall\, point : areacrit \bullet \#point = num \\
(1c)\ \forall\, point : areacrit;\ i : 1 \,..\, num \bullet point(i) \in (seqparam(i)).areaparam \\
(1c)\ \forall\, t_0 : Time \bullet statecrit(t_0) = act \Leftrightarrow \\
\quad \{i : 1 \,..\, num \bullet i \mapsto (seqparam(i)).valueparam(t_0)\} \in areacrit
\end{array}
$$

FunctionalRedundancy

$$
\begin{array}{l}
(2)\ setmaincrit : \mathbb{P}_1\ criterion \\
(3)\ setaddcrit : \mathbb{P}_1\ criterion \\
(2a)\ delay : criterion \twoheadrightarrow Time \\
(3a)\ addcrit : criterion \twoheadrightarrow criterion \\
\hline
(3a)\ \#setaddcrit \leq \#setmaincrit \\
(2a)\ \mathrm{dom}\ delay = \mathrm{dom}\ addcrit = setmaincrit \\
(3a)\ \mathrm{ran}\ addcrit = setaddcrit \\
(4)\ \forall\, maincrit : criterion \mid maincrit \in setmaincrit \bullet \\
\quad maincrit.seqparam \cap (addcrit(maincrit)).seqparam = \varnothing \\
(5)\ \forall\, maincrit : criterion;\ t_0 : Time \mid maincrit \in setmaincrit \wedge \\
\quad maincrit.statecrit(t_0) = act \wedge \\
\quad (\forall\, t : Time \mid t < t_0 \bullet maincrit.statecrit(t) = wait) \bullet \\
\quad\ \exists\, t_1 : Time \mid t_1 \geq t_0 \bullet \\
\quad\ (addcrit(maincrit)).statecrit(t_1) = act \wedge t_1 - t_0 \leq delay(maincrit)
\end{array}
$$

A Study of Developer Attitude to Component Reuse in Three IT Companies

Jingyue Li[1], Reidar Conradi[1,3], Parastoo Mohagheghi[1,2,3], Odd Are Sæhle[1], Øivind Wang[1], Erlend Naalsund[1], and Ole Anders Walseth[1]

[1] Department of Computer and Information Science,
Norwegian University of Science and Technology (NTNU),
NO-7491 Trondheim, Norway
{jingyue, conradi}@idi.ntnu.no
[2] Ericsson Norway-Grimstad, Postuttak, NO-4898 Grimstad, Norway
{parastoo.mohagheghi}@ericsson.com
[3] Simula Research Laboratory, P.O.BOX 134, NO-1325 Lysker, Norway

Abstract. The paper describes an empirical study to investigate the state of practice and challenges concerning some key factors in reusing of in-house built components. It also studies the relationship between the companies' reuse level and these factors. We have collected research questions and hypotheses from a literature review and designed a questionnaire. 26 developers from three Norwegian companies filled in the questionnaire based on their experience and attitudes to component reuse and component-based development. Most component-based software engineering articles deal with COTS components, while components in our study are in-house built. The results show that challenges are the same in component related requirements (re)negotiation, component documentation and quality attributes specification. The results also show that informal communications between developers are very helpful to supplement the limitation of component documentation, and therefore should be given more attention. The results confirm that component repositories are not a key factor to successful component reuse.

1 Introduction

Systematic reuse is generally recognized as a key technology for improving software productivity and quality [17]. With the maturity of component technologies, more and more companies have reused their software (program) in the form of components. Component reuse consists of two separate but related processes. The first deals with analysis of the application domain and development of domain-related components, i.e. *development-for-reuse.* The second process is concerned with assembling software system from prefabricated components, i.e. *development-with-reuse.* These two processes are tightly related, especially in reusing in-house built components. The number of components and the ratio of reused components to total components will determine the reuse benefits (e.g. improved productivity and quality) [11][23].

To investigate the current state of practice and challenges for *development-with-reuse* in the IT industry, and to investigate the relationship between companies' reuse level and some key factors in reusing of in-house components, an empirical study was

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 538–552, 2004.

performed as part of two Norwegian R&D projects. These projects were SPIKE (Software Process Improvement based on Knowledge and Experience) [29] and INCO (INcremental and COmponent-based development) [30]. From the literature review, we defined several research questions and hypotheses. A questionnaire was designed to investigate these questions. Developers from three Norwegian IT companies filled in the questionnaire based on their experience and attitudes to component reuse.

From the results of the survey, we found some new challenges in component reuse and component-based development based on in-house built components. The results support some commonly held beliefs and contradict others.

As the sample size of current research is still small, this study cannot provide statistically significant tests on hypotheses, and is therefore a pre-study. Later studies will be undertaken with refined hypotheses and on a larger sample.

The reminder of the paper is structured as follows: Section 2 presents some general concepts. Section 3 describes the research approach. Section 4 presents the survey results. Section 5 gives a detail discussion on the survey results. Conclusion and future research are presented in section 6.

2 Component Reuse and Component-Based Development

Software reuse can take many different forms, from ad-hoc to systematic [16]. In the broad definition of reuse, it includes reusing everything associated with software projects, such as procedures, knowledge, documentation, architecture, design and code. In our research, we focus on systematic reuse of software code. The code reuse literature has identified reuse practice and success factors through several case studies and surveys. A major reuse effort is the REBOOT (Reuse Based on Object-Oriented Techniques) consortium [25]. This effort was one of the early reuse programs that recognized the importance of not only the technical, but also the organizational aspects of reuse [18]. As more experience become available from industrial studies, non-technical factors, such as organization, processes, business drivers and human involvement, appeared to be at least as important as technological issues [15][19].

Following the success of the structured design and OO paradigms, component -based software development has emerged as the next revolution in software development [27]. More and more IT companies have started to reuse code by encapsulating it into components. Whitehead defines a component as: *A software component is a separable piece of executable software, which makes sense as a unit, and can interoperate with other components, within some supporting environment. The component is accessible only via its interface and is capable of use 'as-is', after any necessary installation and configuration procedures have been carried out* [28].

Component-based development is assumed to have many advantages. These include more effective management of complexity, reduced time to market, increased productivity, improved quality, a greater degree of consistency and a wider range of usability [4][13]. It also brings many challenges, because it involves various stakeholders and roles, such as component developers, application developers, and customers. Different stakeholders and roles have different concerns [3], and face different issues and risks [2][27].

Component-based development differs from traditional development, where the usual approach is for stakeholders to agree upon a set of requirements and then build a system that satisfies these requirements from scratch. Component-based development builds application by reusing existing components. Available components may not be able to satisfy all the requirements. Therefore, component-based projects must have flexibility in requirements, and must be ready to (re)negotiate the requirements with the customer. Moreover, components are intended to be used 'as-is'. If some additional functionality is required, 'glue-code' is needed to be built to meet the differences between the requirement and component functionality. Another important feature of component-based development is the strong focus on the quality attributes (such as reliability, performance, and security etc.) and related testing. A major effort has to be put into checking how components perform, how well they interact, and to make sure that they are indeed compatible. Components may be developed in-house, acquired as COTS (commercial-off-the-shelf) [3], or even as OSS (Open Source Software) [5]. Most current research on component-based software engineering focuses on COTS-based development. Because COTS users cannot access the source code and must rely on vendors to give technical support, COTS-based development is assumed to be more challenging. Therefore, there is little research on the challenges based on in-house built components.

3 Research Approach

The difference between development based on in-house built components and development based on COTS is that the former is related very tightly with *development-for-reuse.* Component reuse is generally an incremental procedure. The company will build some reusable components in the beginning. In case of successful reuse, more and more code will be encapsulated into reusable components. The more reusable components are developed, the more complex will the development process be, and more support is required from the organization [8]. Our motivation is to investigate the relationship between companies' reuse level and some key factors in component-based development so that company with low reuse level can make necessary software process improvements when moving to a higher reuse level.

3.1 Research Questions

To reuse in-house components successfully, developers must follow three basic steps [19]:

- Formulate the requirements in a way that supports retrieval of potentially useful reusable components.
- Understand the retrieved components.
- If the retrieved components are sufficiently 'close' to the needs at hand and are of sufficient quality, then adapt them.

From these steps, we selected several key factors. For step 1, we focus on the efficiency of component related requirements (re)negotiation and the value of component repository. For step 2, we study how knowledge about components can be transferred from a component provider to a component user. For step 3, our study focuses on definition and reasoning of quality attributes of components.

There is little research on the need for requirements (re)negotiation when components are built in-house. People assume that owning source code of in-house built components allows them to do any changes to meet the customers' requirements. However, components are intended to be used 'as-is', even it is built in-house. So, our first research question is:

RQ1. Does requirements (re)negotiation for in-house components really work as efficiently as people assume?

Crnkovic et al. have proposed that to successfully perform the component-based requirements (re)negotiation, a vast number of possible component candidates must be available, as well as tools for finding them [9]. Companies with a higher reuse level usually have more component candidates, more experience, and better experience than companies with a lower reuse level. So, our second research question is:

RQ2. Does the efficiency of component related requirements (re)negotiation increase with more in-house built components available?

To investigate this question, we formalized a null hypothesis H01 and an alternative hypothesis HA1 as follows:

H01. There is no relationship between the companies' reuse level and the efficiency of component related requirements (re)negotiation.

HA1 There is a positive relationship between the companies' reuse level and the efficiency of component related requirements (re)negotiation.

Concerning a component repository, Frakes claimed that it should not be given much attention, at least initially [12]. So, our third research question is:

RQ3. Does the value of component repository increase with more reusable components available?

To investigate this opinion more deeply, a null hypothesis H02 and an alternative hypothesis HA2 was proposed:

H02. There is no relationship between the companies' reuse level and the value of component repository.

HA2. There is a positive relationship between the companies' reuse level and the value of component repository.

A complete specification of a component should include its functional interface, quality characteristics, use cases, tests, etc. While current component-based technologies successfully manage functional interfaces, there is no satisfactory support for expressing quality parts of a component [9]. So, our fourth research question is:

RQ4. How can a component user acquire sufficient information about relevant components?

Berglund claimed that growing reusable software components will create a new problem, i.e. the *information-overload problem.* Therefore, learning which component

to use and how to use them become the central part of software development [1]. Companies with a higher reuse level usually have more reusable components than companies with lower reuse level. So, our fifth research question is:

RQ5. Does the difficulty of component documentation and component knowledge management increase with increasing reuse level?

To study this question, we formalize null hypothesis H03 and alternative hypothesis HA3:

H03. There is no relationship between the companies' reuse level and developers' satisfaction with component documentation.

HA3. There is a negative relationship between the companies' reuse level and developer' satisfaction with component documentation.

One key issue in component-based development is *trust,* i.e. we want to build trustworthy systems out of parts for which we have only partial knowledge [7]. Current component technologies allow systems builders to plug components together, but contribute little to ensure how well they will play together or to fulfill certain quality properties. So, the sixth research question is:

RQ6. Do developers trust the quality specification of their in-house built components? If the answer is no, how can they solve this problem?

3.2 The Questionnaire

The questionnaire included five parts. The questions in the first part were used to investigate the *reuse level* of the companies. The definition of *reuse level* in this study is the number of reused components vs. the number of total components in the organization. The other four parts were organized based on the four key factors. Each question in the questionnaire was used to study one or more research questions. The details of questions are showed in the following Table 1. The correspondences between the questions in the questionnaire and research questions are showed in Table 2. To increase the reliability of our survey, the questionnaire also included the definition of concepts used in the questionnaire, and the questions about the respondents' personal information.

3.3 Data Collection

The study was performed in three Norwegian IT companies. Data collection was carried out by NTNU PhD and MSc students. Mohagheghi, Naalsund, and Walseth performed the first survey in Ericsson in 2002. In 2003, Li, Sæhle and Wang performed the survey reusing the core parts of the questionnaire in two other companies (i.e. EDB Business Consulting and Mogul Technology). We selected those three companies because they have experience on component reuse and would like to cooperate with NTNU in this research. The respondents are developers in these three companies. They answered the questionnaires separately. The questionnaires were filled in either by hand or electronically (as a Word file). The MSc students provided support with possible problems in answering the questionnaire.

Table 1. Questions in the questionnaire

Resuse Level
Q1. What is the reuse level in your organization? Q2. To what extend do you feel affected by reuse in your work?
Component related requirements (re)negotiation
Q3. Are requirements often changed/ (re)negotiated in typical develop projects? Q4. Are requirements usually flexible in typical projects? Q5. Do the component related requirements (re)negotiation processes work efficiently in typical projects?
Value of component repository
Q6. Would the construction of a reuse repository be worthwhile?
Component understanding
Q7. Do you know the architecture of the components well? Q8. Do you know the interface of the components well? Q9. Do you know the design rules of the components well? Q10a. Is the existing design/code of reusable components sufficiently documented? Q10b. If the answer of Q10a is 'sometimes' or 'no', is this a problem? Q10c. If the answer of Q10a is 'sometimes' or 'no', what are the problems with the documentation? Q10d. If the answer of Q10a is 'sometimes' or 'no', how would you prefer the documentation? Q10e. What is your main source of information about reusable components during implementation? Q10f. How do you decide whether to reuse a component 'as-is', 'reuse with modification' or 'make a new one from scratch'?
Quality attributes specification of components
Q11. Are specifications for components' quality attributes well defined? Q12. Do you test components after modification for their quality attributes before integrating them with other components?

Table 2. Correspondence between Questions in the questionnaire and Research Questions

	RQ1	RQ2	RQ3	RQ4	RQ5	RQ6
Q1-Q2		X	X		X	
Q3-Q5	X	X				
Q6			X			
Q7-Q10f				X	X	
Q11-Q12						X

Below, we briefly characterize these three companies and respondents.

3.3.1 Companies

Ericsson Norway-Grimstad started a development project five years ago and has successfully developed two large-scale telecommunication systems based on the same architecture and many reusable components in cooperation with other Ericsson

organization. Their two main applications share more than 60% of ca. 1M lines of code [22].

EDB Business Consulting in Trondheim (now Fundator) is an IT-consultant firm which helps its customers to utilize new technology. It started to build reusable components from 2001. They have built some reusable components based on the Microsoft .Net in their eCportal framework (i.e. a web-application framework) 1.0 & 2.0. These components have been successfully reused in their new e-commence applications.

Mogul Technology (now Kantega) in Trondheim has large customers in the Norwegian finance- and bank sector. The main responsibilities are development and maintenance of the customers' Internet bank application. The application was originally a monolithic system. After several years in production, the customer itself took initiative to reengineer the old system to a component-based solution based on EJB component model in 2002. At the time of the survey, some components have been created and reused in their new Internet bank system.

3.3.2 Respondents

There were 200 developers at Ericsson in Grimstad, where we sent out 10 questionnaires to developers in one development team and got 9 filled-in questionnaires back. There were 20 developers in EDB Business Consulting in Trondheim, and we gathered 10 filled-in questionnaires back out of 10. We distributed 10 questionnaires to 22 developers at Mogul Technology in Trondheim and got 7 back. Those developers were selected because their work was related to component reuse, and they could assign effort to participate in the survey. This is non-probability sampling, which is based on convenience. Most participants in this survey have a solid IT background. 6 of 26 respondents have MSc degree in computer science and all others have a bachelor degree in computer science or telecommunication. More that 80% of them have more than 5 years of programming experience. The details of their position and their experience in the current organization are summarized in the following Table 3.

4 Survey Results

In this section, we summarize the result of the survey. All the following statistical analyses are based on valid answers, i.e. *Don't Know*-answers are excluded. The statistical analysis tool we used is SPSS Version 11.0.

4.1 Different Reuse Level in These Companies

First, we wanted to know the reuse level in those three companies. Q1 and Q2 were asked to get the answer based on developers' subjective opinion on this issue. The result of Q1 is showed in Fig. 1, and the result of Q2 is showed in Fig. 2. From Fig. 1 and Fig. 2, we can see that most developers in Ericsson think that the reuse level in their company is very high or high. Most developers in EDB regard the reuse level in

their company is high or medium. Most developers in Mogul think that the reuse level in their company is medium or little.

Table 3. Background of the respondents

Company	Position and working experience in the organization
Ericsson Norway-Grimstad	2 system architects, 7 designers. 1 person has 13 years of experience 7 persons have experience from 2-5 years, 1 person has 9 months of experience,
EDB Business Consulting in Trondheim	1 project manager, 5 developers and 4 IT consultants. 1 person has 17 years of experience 8 persons have experience from 3-8 years, 1 person has 2 years of experience.
Mogul Technology in Trondheim	6 developers and 1 maintainer (previous developer). 1 person has 10 years of experience, 6 persons have experience from 2-5 years.

4.2 Component Related Requirements (Re)negotiation

Questions Q3-Q5 were asked to investigate RQ1.We can see that no respondents to Q3 believe that the requirements were never changed/ (re)negotiated. Only 8% of respondents to Q4 think the requirements of their typical project are not flexible. However, only 48% of respondents to Q5 think component related requirements (re)negotiation works well. To study RQ2 and test the hypothesis H01, the correlation between the reuse level and response to Q5 is studied. We assign ordinal values to Ericsson, EDB and Mogul to represent their different reuse levels based on the responses to Q1 and Q2 (Ericsson = 3, EDB = 2, Mogul = 1). We also assign ordinal value to the answer of Q5 (Yes = 3, Sometimes = 2, No =1). The result of correlation between them using one-tailed *Spearman Rank Correlation Coefficient* analysis is .112, and the significance is .306. This shows that there is no significant statistical relationship between the reuse level and the efficiency of component related requirements (re)negotiation.

4.3 Value of Component Repository

From the answer of Q6, we found that 71% of respondents in Mogul and EDB regard constructing a component repository as worthwhile, against 57% in Ericsson. To study RQ3 and test hypothesis H02, the relationship between the answer of Q6 and the reuse level is studied. We use the same ordinal number mapping as previously. The result of correlation between them using one-tailed *Spearman Rank Correlation-Coefficient* analysis is -.124, and significance is .297, which shows that there is no obvious relationship between them.

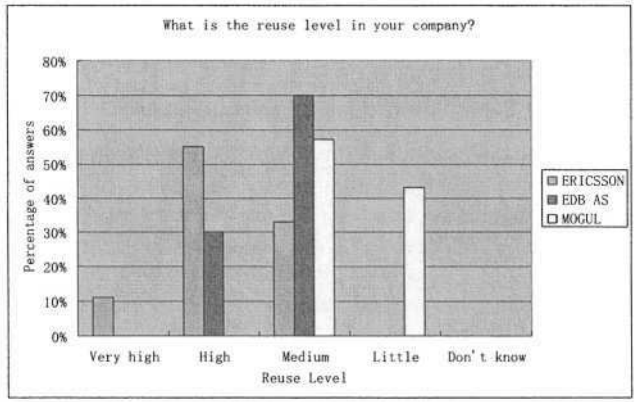


Fig. 1. Result of the question "What is the reuse level in your company?"

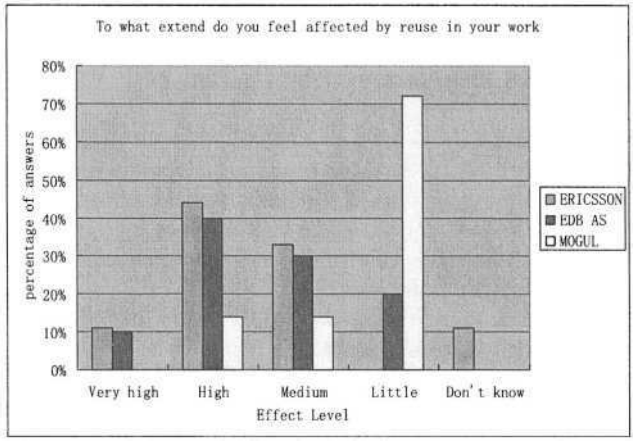


Fig. 2. Result of the question "To what extend do you feel affected by reuse in your work?"

4.4 Component Understanding

Questions Q7-Q10f were used to investigate RQ4. For Q7, Q8 and Q9, the results show that 67% of the respondents think the component structure is well understood, 61% say that the component interfaces are understood, and 63% regard the design rules of components are also well understood. But for the responses to question Q10a, no one thinks that the design/code of components is well documented, 73% think that they are sometimes well defined, and 27% believe that they are not well documented. Furthermore, the answers to questions Q10b and Q10c indicate that 86% believe that insufficient component documentation is a problem, e.g. documentation is not complete, not updated, and difficult to understand, etc. From responses to Q10d and Q10f, we can see that the preferable way of documentation is web pages. Some of the developers' knowledge of how to use components comes from informal

communication sources, for example, previous experience, suggestions from local experts, etc. To study RQ5 and test hypothesis H03, the association between reuse level and response to Q10a is studied. We use the same ordinal number mapping as previously. The result of correlation between them using one-tailed *Spearman Rank Correlation Coefficient*-analysis is -.469, and significance is .014, which shows that there is a weak negative relationship between them. It means that the higher the companies' reuse level, the less satisfied a developer is with the component documentation.

4.5 Quality Attributes of Components

Question Q11 and Q12 were used to investigate RQ6. From the responses to these questions, we see that 70% of the participants regard the design criteria for quality requirements are not well defined, and 87% will test the quality attributes of components after component modification, before integrating them into the system.

5 Discussions

Based on the result of the survey, we discuss our research questions and hypotheses, and discuss the limitations and threats to validity.

5.1 Component Related Requirements (Re)negotiation

Much research focus on how to improve the efficiency of component related requirements (re)negotiation in COTS-based development [20] [24] [26]. The main reason is that people think the challenges in requirements (re)negotiation are due to the lack of access to source code, to timely vendor supports, or to the lack of engineering expertise to modify the integrated components [26]. In our case, the components are mostly built in-house. The above constrains on COTS components are not considered as challenges with built in-house components. From the responses to question Q3-Q5, we found that although 92% think that requirements of their typical projects are flexible, less than half think the component related requirements (re)negotiation in their typical projects works well.

Since components are intended to be used 'as-is', it is possible that an in-house reusable component meeting all the requirements will not be found. So, even though the components are built in-house, requirements (re)negotiation is necessary. For research question RQ1, we do not want to claim that the requirements (re)negotiation based on in-house components is more difficult than COTS-based components. We just want to emphasize that requirements (re)negotiation based on in-house components is also important but not efficient.

From the test result on H01, we cannot find a statistically significant relationship between the reuse level and the efficiency of component related requirements (re)negotiation. So, we cannot reject null hypothesis H01. Our conclusion to RQ2 is that when IT companies change from a low reuse level to a higher reuse level, they

probably cannot expect that component-based requirements (re)negotiation becomes easier and more efficient.

5.2 Component Repository

Some researchers have claimed that repository is important, but not sufficient for successful reuse [18][21]. Our data confirms that developers are positive, but not strongly positive to the value of component repository. So, this result gives future support to the previous conclusion.

From the test result on H02, we can see that there is no statistically significant relationship between developers' positive attitude to a component repository and reuse level. So, we cannot reject null hypothesis H02. Our conclusion to RQ3 is that companies are not expected to invest in a repository to increase reuse.

5.3 Component Understanding

Transferring component knowledge from the component developer to the component user is critical for successful component reuse. The answers of Q7-Q9 show that most developers understand the components in detail. However, the answers of Q10a-Q10c show that no one believes that the components are well documented because the documents are either incomplete or not updated. So, our question is *"How can developers still understand the components without good documentation?"* From the answers to question Q10e and Q10f, we found that most developers got the knowledge of components from informal channels, such as previous experience and local experts. The most important feature of a component is the separation of its interface from its implementation. The component implementation is only visible through its interface. Moreover, current component documentation technologies cannot describe all the information the developer required, such as performance, reliability, and security etc. Therefore, informal knowledge transfer should be considered to supplement the insufficiency of formal component documentation and specification. This point was showed in other empirical studies as well [6][10]. For research question RQ4, we found that informal knowledge transfer is especially important in the component reuse. One possible solution is to have special interest groups or mailing lists for a components (or group of similar components) so that component users can share knowledge and experience of component usage.

From the test result on H03, we found a weak negative relationship between reuse level and developers' satisfaction with the documentation. We reject the null hypothesis H03 and accept the alternative hypothesis HA3, It means the higher the companies' reuse level, the less satisfied a developer is with components' documentation. Marcus et al. concluded that combine reuse education and training provided for staff with other reuse activity can lead to all the success of reuse [18]. Our conclusion to RQ5 implies that when a company moves from a low reuse level to high level, more effort should be spent on the component documentation and component knowledge management.

5.4 Quality Attributes of Components

Component-based development relies on the availability of high quality components to fill roles in a new intended system. When components are created or changed, we must ensure that they do not only fulfill the functional requirements, but also quality requirements. For research question RQ6, we found that most developers are not satisfied with the specification of components' quality attributes and therefore cannot use this information. Therefore, how can we model quality properties of both components and systems, and reason about them, particularly in the early stage of system development is still a key challenge in component-based development.

5.5 Threats to Validity

We now discuss the possible validity threats in this study. We use the definition given by Judd et al. [14].

Construct validity. In our case, the main construct issue applies to the variables chosen to characterize the data set. The independent variable, i.e. reuse level, is the most sensible one. The results of questions Q1 and Q2 give a qualitative and consistent value on this variable.

Internal validity. A major threat to this validity is that we have not assessed the reliability of our measurement. Most variables are measured on a subjective ordinal scale. An important issue for future studies is to ensure the reliability and validity of all measurement. In this survey, we gave clearly specified concepts in the questionnaire and provided support to possible misunderstanding. These methods partly increased the reliability.

External validity. The small sample size and lack of randomness in the choice of companies and respondents are threats to external validity. In general, most empirical studies in industry suffer from non-representative participation, since companies that voluntarily engage in systematic improvement activities must be assumed to be better-than-average.

Conclusion validity. This study is still a pre-study. Future studies will be implemented to give more statistically significant results.

6 Conclusion and Future Work

This study has investigated challenges related to four key factors for development based on in-house components, especially in development-with-reuse. These factors are component related requirements (re)negotiation, component repository, component understanding and components' quality attribute specification. Another contribution is that we compared three IT companies with different reuse levels to study the possible trend and challenges in these factors when more and more code will be encapsulated as reusable components inside a company.

- For *component-based requirements (re)negotiation,* the results of research questions RQ1 and RQ2 show that requirements (re)negotiation for in-house built components is important but not efficient. The efficiency will probably not increase with higher reuse level.
- For the *component repository,* the results of research question RQ3 confirm that a component repository is not a key factor for successful reuse. Furthermore, the potential value of a component repository will probably not increase with higher reuse levels.
- For *component understanding,* the results of research questions RQ4 and RQ5 show that most developers are not satisfied with the component documentation, and developers' satisfaction with component documentation will probably decrease with higher reuse level. The results also show that informal communication channels, which developers can get necessary information about the components through, should be given more attention
- For *components' quality attribute specification,* the result of research question RQ6 shows that developers still need to spend much effort on testing, as they cannot get relevant information from component specifications.

The main limitation of our survey is that it depends on the subjective attitudes of developers, and with few companies and participants involved. Later studies are planned to be undertaken with more precise quantitative methods and on more companies with more distinct reuse levels. Case studies will also be undertaken to follow the change of companies from lower reuse level to higher reuse level to future investigate our research questions.

Acknowledgements. This study is supported by the SPIKE and INCO projects. We thank the colleagues in these projects, and all the participants in the survey.

References

1. Erik Berglund: Writing for Adaptable Documentation. Proceedings of IEEE Professional Communication Society International Professional Communication Conference and Proceedings of the 18th ACM International Conference on Computer Documentation: Technology & Teamwork, Cambridge, Massachusetts, September (2000) 497–508
2. Pearl Brereton: Component-Based System: A Classification of Issues. IEEE Computer, November (2000), 33(11): 54 – 62
3. Alan W. Brown: The Current State of CBSE. IEEE Software, September/October (1998) 37–46
4. Alan W. Brown: Large-Scale Component-Based Development. Prentice Hall, (2000)
5. Alan. W. Brown and Grady. Booch: Reusing Open-source Software and Practices: The Impact of Open-source on Commercial Vendors. Proceedings: Seventh International Conference on Software Reuse, Lecture Notes in Computer Science, Vol. 2319. Springer, (2002) 123 – 136.
6. Reidar Conradi, Tore Dybå: An Empirical Study on the Utility of Formal Routines to Transfer Knowledge and Experience. Proceedings of European Software Engineering Conference, Vienna, September (2001) 268 – 276

7. Bill Councill and George T. Heineman: Component-Base Software Engineering and the Issue of Trust. Proceedings of the 22nd International Conference on Software Engineering, Limerick, Ireland, June (2000) 661 – 664
8. Ivica Crnkovic and Magnus Larsson: A Case Study: Demands on Component-based Development. Proceedings of the 22nd International Conference on Software Engineering, Limerick, Ireland, June (2000) 21 – 31 .
9. Ivica Crnkovic: Component-based Software Engineering - New Challenges in Software Development. Proceedings of 25th International Conference on Information Technology Interfaces, Cavtat, Croatia, June (2003) 9 – 18
10. Torgeir Dingsøyr, Emil Røyrvik: An Empirical Study of an Informal Knowledge Repository in a Medium-Sized Software Consulting Company. Proceedings of 25th International Conference on Software Engineering, Portland, Oregon, USA, May (2003) 84 – 92
11. W. B. Frakes: An Empirical Framework for Software Reuse Research. Proceedings of the Third Annual Reuse Workshop, Syracuse University, Syracuse, N.Y. (1990)
12. W. B. Frakes, C.J. Fox: Sixteen Questions about Software Reuse. Communication of the ACM, June (1995), 38(6): 75 – 87
13. Ivar, Jacobson, Martin Griss, Patrick Jonsson: Software Reuse-Architecture, Process and Organization for Business Success. Addison Wesley Professional, (1997)
14. C.M. Judd, E.R. Smith, L.H. Kidder: Research Methods in Social Relations. Sixth edition, Holt Rinehart and Winston, (1991)
15. Y. Kim and E.A. Stohr: Software Reuse: Survey and Research Directions. Journal of Management Information System, (1998), 14(4): 113 – 147
16. C. Kruger: Software Reuse. ACM Computing Surveys, (1992), 24(2): 131 – 183
17. N. Y. Lee, C. R. Litecky: An Empirical Study on Software Reuse with Special Attention to Ada. IEEE Transactions on Software Engineering, September (1997), 23(9): 537 – 549
18. Marcus A. Rothenberger, Kevin J. Dooley and Uday R. Kulkarni: Strategies for Software Reuse: A Principal Component Analysis of Reuse Practices. IEEE Transactions on Software Engineering, September (2003), 29(9): 825 – 837
19. H. Mili, F. Mili, A. Mili: Reusing Software: Issues and Research Directions. IEEE Transactions on Software Engineering, June (1995), 21(6): 528 – 561
20. M. Morisio, C.B. Seaman, A. T. Parra, V.R. Basili, S.E. Kraft, S.E. Condon: Investigating and Improving a COTS-Based Software Development Process. Proceeding of 22nd International Conference on Software Engineering, Limerick, Ireland, June (2000) 31 – 40
21. Maurizio Morisio, Michel Ezran, Colin Tully: Success and Failure Factors in Software Reuse. IEEE Transactions on Software Engineering, April (2002), 28(4): 340 – 357
22. Parastoo Mohagheghi and Reidar Conradi: Experiences with Certification of Reusable Components in the GSN Project in Ericsson, Norway. Proceedings of the 4th ICSE Workshop on Component-Based Software Engineering: Component Certification and System Prediction. Toronto, May (2001) 27 – 31
23. Jeffrey S. Poulin: Measuring Software Reuse-Principles, Practices, and Economic Models. Addison-Wesley, (1997)
24. Vijay Sai: COTS Acquisition Evaluation Process: The Preacher's Practice. Proceedings of 2nd International Conference on COTS-based software systems, Lecture Notes in Computer Science, Vol. 2580. Springer, 2003, Ottawa, Canada, February (2003) 196 – 206
25. Guttorm Sindre, Reidar Conradi, and Even-Andre Karlsson: The REBOOT Approach to Software Reuse. Journal of System Software, (1995), 30(3): 201 – 212

26. Vu N. Tran, Dar-Biau Liu: Application of CBSE to Projects with Evolving Requirements- A Lesson-learned. Proceeding of the 6th Asia-Pacific Software Engineering Conference (APSEC' 99) Takamatsu, Japan, December (1999) 28 – 37
27. Padmal Vitharana: Risks and Challenges of Component-based Software Development. Communications of the ACM, August (2003), 46(8): 67 – 72
28. Katharine Whitehead: Component-Based Development: Principles and Planning for Business Systems. Addison-Wesley, (2002)
29. http://www.idi.ntnu.no/grupper/su/spike.html
30. http://www.ifi.uio.no/~isu/INCO/

Managing COTS Components Using a Six Sigma-Based Process*

Alejandra Cechich[1] and Mario Piattini[2]

[1] Departamento de Ciencias de la Computación, Universidad Nacional del Comahue, Buenos Aires 1400, 8300 Neuquén, Argentina
acechich@uncoma.edu.ar

[2] Escuela Superior de Informática, Universidad de Castilla-La Mancha, Paseo de la Universidad 4, Ciudad Real, España
Mario.Piattini@uclm.es

Abstract. While the objectives of Six Sigma are to reduce variation and prevent defects, it is also a management philosophy that includes the need for fact-based decisions, customer focus, and teamwork. In attempting to build a COTS integrated system, selection of candidates typically pays attention to specify search criteria and goals to be met. Yet they often overlook some elements in the process such as fact-based decisions and teamwork, which might drive the process helping increase the probability of success. In this paper, we describe and illustrate a Six Sigma-based proposal for the process of selecting and integrating COTS components. Our approach also suggests some tools and measures to be applied during its specific phases.

1 Introduction

As Component-based Software Development (CBSD) starts to be effectively used, some software vendors have commenced to successfully sell and license commercial off-the-shelf (COTS) components. CBSD advocates the use of pre-fabricated pieces, perhaps developed at different times, by different people, and possibly with different uses in mind. The goal, once again, is the reduction of development times, costs, and efforts, while improving the flexibility, reliability, and maintainability of the final application due to the (re)use of software components already developed, tested and validated.

The particular nature of COTS components (black-box binary entities developed by third parties, and independently deployable in unforeseen contexts) imposes specific quality mechanisms to be put in place for their effective integration in the devel-

* This work is partially supported by the CyTED (Ciencia y Tecnología para el Desarrollo) project VII-J-RITOS2, the UNComa project 04/E048 (Modelado de Componentes Distribuidos Orientados a Objetos), and by the MAS project supported by the Dirección General de Investigación of the Ministerio de Ciencia y Tecnología (TIC 2003-02737-C02-02).

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 553–567, 2004

opment life cycle of software applications. Firstly, components need to be properly documented and offer measurement mechanisms in order to be assessed and evaluated for selection. Secondly, both the quality and some of the extra-functional properties of the final (composed) system heavily depend on the individual properties of its constituent components, and therefore some traceability between the quality attributes at these two levels is required. Finally, the use of third-party COTS components may also introduce risks, such as potential harmful side effects to the system, or quality degradation of the final product. COTS component testing and certification may offer partial solutions to these problems [5].

It is important but often difficult to determine the appropriate level of Component-Based Systems (CBS) quality. Fitness for use implies that the appropriate level of CBS quality is dependent on the context, and determining the quality is difficult when different users have different needs. Even the field of component quality has experienced significant advances during its relatively brief history, CBS quality has not yet advanced to the point where there are standard measurement methods, and few enterprises routinely measure COTS component quality. However, some efforts have started to define software metrics to guide quality and risk management in a CBS by identifying and quantifying various factors contributing to the overall quality [1,18].

On the other hand, Six Sigma is an approach to product and process improvement that has gained wide acceptance and has delivered large business benefit across many industries [12]. While the objectives of Six Sigma are to reduce variation and prevent defects, it is also a management philosophy that includes the need for fact-based decisions, customer focus, and teamwork. Six Sigma precepts can guide stakeholders through the selection and integration of COTS components and help increase the probability of success. In [19], the Six Sigma approach has been suggested selecting packaged software, however the evaluation mainly relies on the information provided by demos and additional documentation of the COTS software. Then, the lack of measures makes this process perfectible.

In this paper, we present a preliminary approach that can be used in the process of selecting and integrating COTS components in conjunction with Six Sigma precepts. The major goal of our work is to reduce the effort required to select COTS components by defining a framework based on customer focus, fact-based decisions, and teamwork. Particularly, when reasoning about provided and required services of components, we analyse them in terms of committed functionality. In Section 2 of the paper we introduce the main characteristics of our approach. Section 3 then presents the detailed phases by using a motivating example. Section 4 adds some remaining issues and finally, we conclude with an overview and future research directions.

2 A Six Sigma-Based Process: Motivation and Overview

Six Sigma is typically divided into five phases, creating what is referred to as DMAIC, which is an acronym for the following phases [19]:

1. *Define* the problem and identify what is important: Identify the problem and the customers; define and prioritise the customer's requirements; and define the current process.
2. *Measure* the current process: Confirm and quantify the problem; measure the various steps in the current process; revise and clarify the problem statement, if necessary; and define desired outcome.
3. *Analyse* what is wrong and potential solutions: Determine the root cause of the problem; and propose solutions.
4. *Improve* the process by implementing solutions: Prioritise solutions; and develop and implement highest benefit solutions.
5. *Control* the improved process by ensuring that the changes are sustained: Measure the improvements; communicate and celebrate successes; and ensure that process improvements are sustained.

Selection and integration of COTS components need to ensure, as in any Six Sigma project, that decisions are based on facts and that customer's requirements have been considered. However, in the continuing attempt to introduce CBSD, organisations have problems identifying the content, location, and use of diverse components. Component composition requires access to tangible and intangible assets. Tangible assets, which correspond to documented, explicit information about the component, can vary from different vendors although usually include services, target platform, information about vendors, and knowledge for adaptation. Intangible assets, which correspond to tacit and undocumented explicit information, consist of skills, experience, and knowledge of an organisation's people.

Six Sigma might help to put all these pieces together and define a measure-based procedure for selecting COTS components. As the selection process implies, the software is packaged with the goal that it will be used by many different systems. To introduce our approach, two kind of systems are referred to as follows:

- *The source system.* We refer to a source system as a set of pieces of software that running all together satisfy part of the user's requirements expressed as functional and non-functional properties, whose quality is able to be measured.
- *The Component-Based System (CBS).* We refer to a component-based system as a system that uses at least one component in conjunction with other components, legacy systems, and other pieces of software – including COTS components – to satisfy user's requirements. The concept of CBS is introduced to emphasize the fact that the output from the system satisfies the user's requirements by using the functionality supplied by at least one COTS component.

System satisfaction can refer both CBS and the source system – not necessarily both CBSs. In Figure 1 (a), the source system is stable in the sense that its outputs O_1 and O_2 satisfy the requirement R_1 (a previously stated and measurable requirement), although there is still a requirement (R_2) not provided by the system.

Let us introduce our case. Suppose that requirement R_2 in Figure 1 (a) can be satisfied by using the functionality provided by a COTS component. It could be the case that this COTS component alone can provide the entire functionality of R_2 – as Figure 1 (b) shows – by means of the output O_3 and using some inputs from the system. The point here is to determine whether the system – now a CBS system – remains stable. In other words, whether the system still preserves the previously quality attributes and the new attributes defined for R_2. Furthermore, how does the system preserve those attributes? Have the interfaces changed? Is quality affected? Have we introduced side effects that make the quality poor? It could also be the case that we need to introduce major changes to the system as Figure 1 shows.

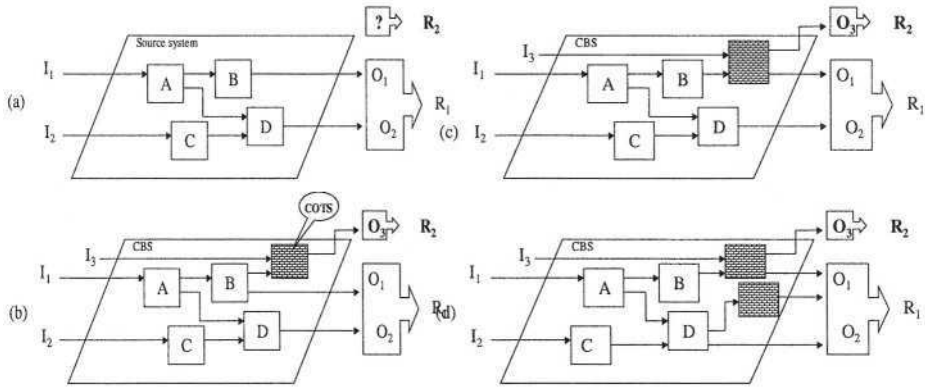


Fig. 1. Cases of an altered source system

In Figure 1 (c), the COTS component is more complex than in the case of Figure 1 (b), providing more functionality by replacing some outputs from the source system. As we can see, component B is not directly contributing to the requirement R_1 but supplying an input interface to the COTS component, that in turn contributes to reach the requirement R_1. In Figure 1 (d) a second COTS component is added providing that outputs from B have to be complemented with new outputs from D due to effects on the composition's functionality.

The process of selecting COTS components and detecting the degree of system satisfaction can be guided by a Six-Sigma approach. The five-phases of the COTS component selection process are shown in Figure 2. Deliverables are summarised in the figure to indicate the existence of a documentation suite composed of criteria for selection, and measures that identify suitability with respect to a source system. Tools in the figure refer to well-known tools for quality assessment as well as specific tools and measures for selecting COTS components.

Let's briefly introduce the phases of Figure 2, when applying the Six Sigma approach to our COTS selection case:

1. *Define* system's stability [6] and stakeholder's preferences. The Define phase sets the goals and context of the project. This phase is concerned with identifying key stakeholders, and determining their needs and criteria for selection. Some of the activities are as follows:
   - Given a source system, we should specify the current degree of satisfaction/stability of the system under evaluation by defining goals, constraints, component context, domain as well as product's quality attributes.
   - Then, we should identify sub-domains and relationships, i.e. relations among pieces of software from an architectural point of view – legacy systems, COTS and standard components – and relations among sub-domains.

2. *Measure.* This phase is concerned with assisting the selection process by creating a complete description of the COTS candidates. Some of the activities are as follows:
   - Measure COTS components to determine their suitability with respect to the required services.
   - Assess the impact of the integration of COTS components on a stable system.

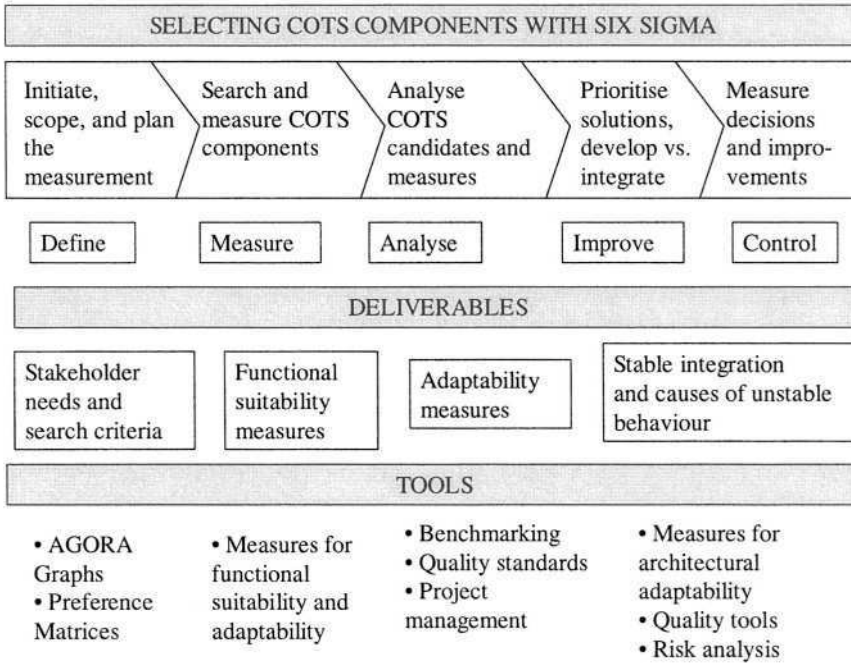


Fig. 2. Five-step measuring process

3. *Analyse.* In this phase, a COTS candidate or a set of COTS candidates is selected from several alternatives. Decisions are made based on previous definitions and measurements. Some of the activities are as follows:
   - Detect sub-domains affected by the recently incorporated COTS.
   - Identify dissatisfactions – functional dissatisfactions and accuracy dissatisfactions – according to the stability state established in phase 1.
   - Determine dissatisfaction's relative magnitudes referring the initial system's stability.
   - Decide whether perturbations affect the initial stability.

4. *Improve.* This phase is concerned with ensuring that the selected candidates can be integrated and supported within the required quality. For example, causes of low architectural adaptability should be determined, along with causes of functional dissatisfaction. Some of the activities are as follows:
   - Identify dissatisfaction causes (i.e., using Ishikawa diagrams).
   - Prioritise solutions deciding on integrating the COTS component/s.

5. *Control.* In this phase a successful COTS selection procedure should communicate its practices and suggests possible corrective/adaptive actions to sustain its success.

3 Moving into Selecting COTS Components

The following sections present our process by using an example of COTS component selection. Due to brevity reasons, only some of the activities involved in each phase are further detailed here.

3.1 Moving into the Define Phase

To ensure that decisions are fact based, it is important that the "define" phase will be reinforced. Although it is always important to understand the current process and the problem to be solved, the importance is greater when selecting COTS components because requirements and services must be balanced as part of a negotiation procedure.

Firstly, COTS component's required functionality should be expressed to define search goals and criteria for evaluation. To do so, we have adapted the model in [1], which explores the evaluation of components using a specification-based testing strategy, and proposes a semantics distance measure that might be used as the basis for selecting a component from a set of candidates. This model supposes that there is an architectural definition of a system, whose behaviour has been depicted by scenarios or using an architecture description language (ADL).

Our proposal [7], has adapted the model as a basement for quality measurement. We express the semantics distance in terms of functional suitability measures, which

provide a better identification of the different COTS component functionalities. Then, a system can be extended or instantiated through the use of some component type. Due several instantiations might occur, an assumption is made about what characteristics the actual components must possess from the architecture's perspective. Thus, the specification of the architecture *A (SA)* defines a specification S_c for the abstract component type *C* (i.e. $SA \Rightarrow S_c$). Given a specification S_c for an abstract component type *C,* a candidate component *K* to be a concrete instance of C must conform to the interface and behaviour specified by S_c. Although the process of selecting a component *K* consists of evaluating interface and semantic mappings, in our work only semantic mappings are addressed. Mappings in S_c, which represent the different required functionalities, are established between input and output domains. We focus on incompatibilities derived from functional differences between the specification in terms of mappings of a component K_i (S_{Ki}) and the specification in terms of mappings of S_c.

We will illustrate the procedure by using an E-payment system as an example. We suppose the existence of some scenarios describing the two main stages of the system – *authorisation* and *capture.* Authorisation is the process of checking the customer's credit card. If the request is accepted, the customer's card limit is reduced temporarily by the amount of the transaction. Capture is when the card is actually debited.

The scenarios will provide an abstract specification of the mappings of S_c that might be composed of:

- Input domain: (AID) Auth_IData{#Card, Cardholder_Name, Exp-Date}; (CID) Capture_Idata{Bank_Acc, Amount}.
- Output domain: (AOD) Auth_Odata{ok-Auth}; (COD) Capture_Odata{ ok_capture, DB_update}.
- Mapping: {AID → AOD};{CID → COD}

Suppose we pre-select two components to be evaluated, namely K_1 and K_2 respectively. A typical situation for inconsistency in the functional mappings between S_{K1}, S_{K2} and S_c is illustrated in Figure 3, where dashed lines indicate (required) mappings with respect to S_c, and the solid lines are (offered) mappings with respect to S_{K1} (grey) and S_{K2} (black). Note that the input domain of the component K_1 does not include all the values that the specification S_c requires, i.e. the capture functionality is not provided. Besides, the input domain of the component K_2 includes more values than the required by S_c, although the mapping satisfies the required functionality. We should also note that there is another functionality provided by K_2, which might inject harmful effects to the final composition.

On the other hand, committing requirements for COTS component selection can be problematic. Traditionally, the requirements elicitation techniques have widely used a family of goal-oriented requirements analysis (GORA) methods, such as I* [13], and KAOS [16], as an approach to refine and decomposing the needs of customers into more concrete goals that should be achieved. Our proposal extends a version of a Goal-Oriented Requirements Analysis Method called AGORA in [15] considering additional features of COTS components.

Initial goals, as considered in AGORA graphs, are the needs of the customers that will be refined and decomposed into sub-goals one after another. Therefore, with the functional goals of the component specified by mappings in S_c, the next step is to refine the goals considering the perspectives of different stakeholders – reuse architect, certifier, business process coordinator, etc. Then, the computation of stakeholder's preference values for the refined goals will allow us to add preferences to mappings of S_c.

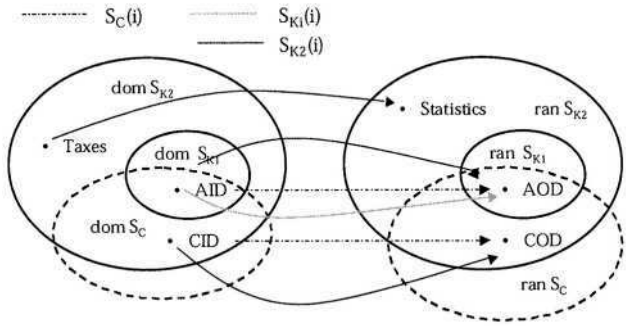


Fig. 3. Functional mappings of S_c and S_{K1}/S_{K2}

In our context of component-based systems, an AGORA graph describes the abstract specification of a required component (S_c) according to the scenario *S*. Figure 4 shows a snapshot of a possible AGORA graph for our E-payment case. Here, there are two types of conflicts on goals; one is the conflict between goals and the other one is the conflict on a goal between stakeholders. The first type of conflicts appears in Figure 4 between the goals "Prevent unauthorised debits" and "No authorisation", whose edge has a negative contribution value. The second type appears in the figure on the goal "Input AID data". The diagonal elements of the preference matrix show that the reuse architect gave the preference value 8 by himself, while the business process coordinator's preference is –5 given by himself (diagonal values on the matrix represent preferences given by a stakeholder on his own judgment).

When incorporating COTS components, goals should be balanced against COTS services. For example, using the main concepts of goal-oriented requirements engineering, the goal acquisition and specification process [2] includes the necessity of identify goals that help to distinguish between products (called core goals) from those that are provided by most available products (called peripheral goals). This characterisation would lead to dealing with a third type of conflicts on goals: conflicts between the abstract specification of a component and its possible instantiations. These conflicts should be resolved when the COTS component selection actually take place. Then, it is important to add some extra information on the graph, so the negotiation will be possible, i.e. the AGORA graph in Figure 4 has added the labels <core>/<peripheral> to facilitate the future negotiation. Besides the classification of goals as core and peripheral, the attributes desirability (level of importance for a goal

to be met), and modifiability (level in which a goal can be modified) are proposed as attributes for goal description when selecting COTS components [2].

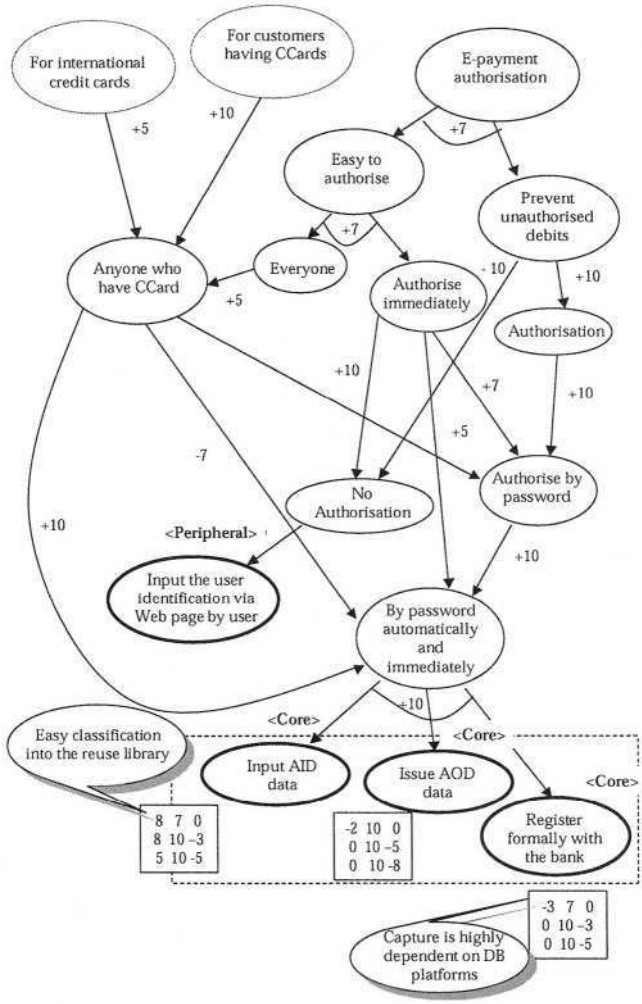


Fig. 4. A snapshot of the AGORA graph to balance stakeholder's preferences on required goals

By using an AGORA graph, we can estimate the quality of several properties of the adopted goals. Particularly, correctness is assumed as a quality factor that represents how many goals in a specification meet stakeholder's needs. Correctness in AGORA is strongly related to contribution values on the path of the adopted goal as well as on its stakeholder's preference value. Particularly, measures of modifiability and desirability of goals for selection may be derived from AGORA graphs.

Based on these measures, a weighted suite of required services can be more accurately specified. For example, we may suggest that desirability should be calculated and decisions should be made based on the following cases, where "agreement-threshold" is a indicated value between 0.6 and 0.7 and "core/peripheral" is the type of refined goal:

- Case 1: desirability(refined goal/s) < agreement-threshold ∧ core ⇒ Try other scenarios to get agreement
- Case 2: desirability(refined goal/s) < agreement-threshold ∧ peripheral ⇒ decision to discharge
- Case 3: desirability(refined goal/s) ≥ agreement-threshold ∧ peripheral ⇒ decision to retain
- Case 4: desirability(refined goal/s) ≥ agreement-threshold ∧ core ⇒ keep for the selection process

We should note that our proposal might be easily combined with other methods for weighting preferences such as the AHP method [17].

3.2 Moving into the Measure Phase

Suppose that before starting the selection procedure, a committed specification of the component (S_c) is defined as a reference to be used when comparing candidates.

Once committed goals have been achieved, we may proceed with the selection process by applying different measures on COTS candidates. Our particular suite of measures of functional suitability of COTS components has been classified into two different groups: component-level measures and solution-level measures. The first group of measures aims at detecting incompatibilities on a particular component *K*, which is a candidate to be analysed. However, it could be the case that we need to incorporate more than one component to satisfy the functionality required by the abstract specification S_c. In this case, the second group of measures evaluates the functional suitability of all components that constitute the candidate solution.

At the component-level, we have defined the following measures [7]:

- "Compatible Functionality" (CF_C) as the amount of functional mappings provided by S_K and required by S_C in the scenario *S*.
- "Missed Functionality" (MF_C) as the amount of functional mappings required by S_C in the scenario *S* and not provided by S_K.
- "Added Functionality" (AF_C) as the amount of functional mappings not required by S_C in the scenario *S* and provided by S_K.
- "Component Contribution"(CC_F) as the percentage in which a component contributes to get the functionality required by S_C in the scenario *S*.

For the complete and formal definition of the functional suitability measure suite, we refer the reader to [7].

Now, let's calculate the functional suitability measures on K_2 for the E-payment example. Considering the functional mappings provided by K_2 ({AID → AOD; CID → COD; Taxes → Statistics}), the component-level measure results are as follows:

$$CF_C(K_2) = 2;\ MF_C(K_2) = 0,\ \ AF_C(K_2) = 1;\ CC_F(K_2) = 1.$$

These values indicate that the component K_2 is a candidate to be accepted for more evaluation; i.e. the component is completely functionally suitable. But there is one added function that could inject harmful side effects into the final composition. Besides, there are another types of analysis the component should be exposed before being eligible as a solution – such as analysis of non-functional properties [8], analysis of vendor viability [3], and so forth.

During the mapping, it can be the case that the semantics of *K* are not sufficient for S_C. In this situation, *K* is somehow lacking with respect to the behavioural semantics of *C*. The possibility is that *K* has partial behavioural compatibility with *C*. In this case, *K* either has incompatible or missing behaviour with respect to some portion of S_C. To overcome this, a semantic adapter must be specified (and built) such that, when composed with S_K, the adapter yields a component that is compatible with *C* [1].

Adaptation required by the components should also be quantified. For example, measurement might be defined at three levels:

(1) *Adaptability size measures* will be basically in terms of the amount of adaptability needed by a component-based solution. For example, one of the adaptability's measures might measure the amount of functions that are added when implementing the adapter of the component *K* (extended functionality); or contribution measures might measure the completeness of the adapters' functionality. For example, we could measure the percentage in which a component adapter contributes to get the extended functionality required by S_c in the scenario *S*.

(2) *Complexity of adaptation* would be measured in terms of interactions with target components that are identified to determine all potential mismatches. Then, for each mismatch, potential resolution techniques might be considered from the proposal by [10]. Hence, a weighting factor would be assigned to each connection to describe the difficulty with which that solution could be implemented.

(3) *Architectural adaptability* might define calculations for measures of changes that affect system's stability [6] in terms of architectural adaptability. Adaptability of an architecture can be traced back to the requirements of the software system for which the architecture was developed. For example, the POMSAA (Process-Oriented Metrics for Software Architecture Adaptability) framework [9], achieves the need of tracing by adopting the NFR framework that is a process-oriented qualitative framework for representing and reasoning about NFRs (non-functional requirements) [8]. In the NFR Framework, the three tasks for adaptation become softgoals to be achieved by a design for the

software system. An adaptable component of a software system should satisfice these softgoals for adaptation. For example, the underlying architecture for the E-Payment [EPAY] part of a E-commerce [ECOM] system, and the metrics that the POMSAA calculates for the adaptability of the architecture are given in Figure 5. The NFR Framework requires decomposition for the NFRs of interest. Figure 5 also gives the maximum and minimum values (the numbers inside the clouds) for the metrics of different softgoals. We refer the reader to [9] for a more detailed description on the computation of the metrics.

3.3 Moving into the Analyse Phase

In this phase a COTS candidate or a set of COTS candidates is selected from several alternatives, and decisions are made based on previous definitions and measurements. Basement for decisions might include to detect sub-domains affected by the recently incorporated COTS and identify dissatisfactions – functional dissatisfactions and accuracy dissatisfactions – according to the stability state established in Phase 1.

Let's introduce a suggested analysis as an example. The measures calculated in the previous phase, such as Architectural Adaptability, are used as a basement for supporting the different judgments. From Figure 5, we can see that Semantic Adaptation [ECOM, EPAY] has been split into three main hierarchies for analysis – Interface, Protocol, and Data. Each branch of the hierarchy is analysed in terms of complexity and size, using the measures calculated in the previous phase. The semantic adaptation has a metric of -2, which is the minimum score a Semantic Adaptation can get (from previously defined thresholds).

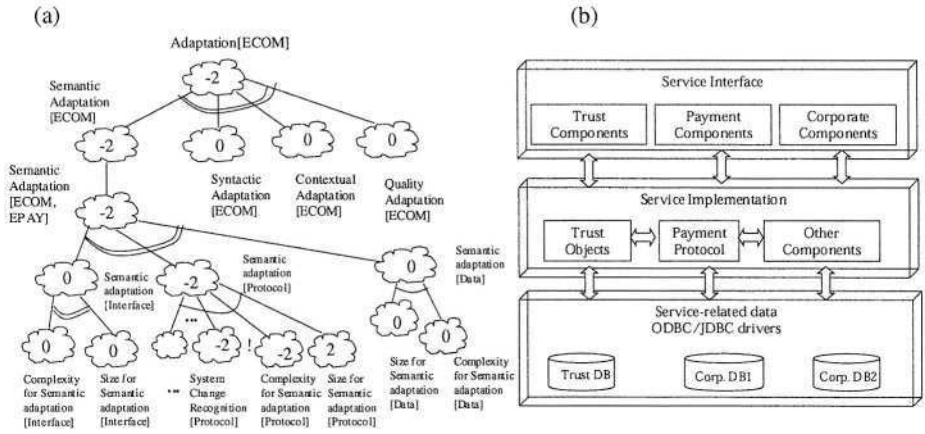


Fig. 5. An underlying architecture (b) and its architectural adaptability measures (a)

We suggest here integrating qualitative and quantitative measures for analysis considering that: (1) qualitative judgments are based on more precise values to be evalu-

ated, which reduce ambiguity of the evaluation process; (2) quantitative values provide a more objective scale for comparison among alternatives; and (3) design decisions might be more formally specified.

3.4 Moving into the Improvement and Control Phases

As we have indicated, this phase is concerned with ensuring that the selected candidates can be integrated and supported within the required quality. For example, causes of low architectural adaptability should be determined, along with causes of functional dissatisfaction.

Evaluation of results can reveal problems that might arise during integration leading to prioritise solutions deciding on integrating the COTS component/s or building the required functionality from scratch. Different architecture's uses may necessitate changes to the same components or connections. In such a case, we should detect uses that affect the architecture keeping designs with the fewest use conflicts. Detecting conflict's causes is a subjective process, involving all stakeholders in the system. The results should reflect the relative importance of the quality factors that the uses manifest.

As an additional concern, improvement must be examined in the context of the system's requirements and design. It includes some aspects relating to verify that the system's requirements h ave been met by integrating the COTS solution.

A successful COTS selection procedure should communicate its practices and suggests possible corrective/adaptive actions to sustain its success. After evaluating results and detecting causes of architectural dissatisfaction, stakeholders must develop metrics and a control plan to ensure that the system continues to deliver the planned improvements. Although these activities do not differ from a traditional system development project or, indeed, from a classic Six Sigma project, they are important and should not be neglected.

4 Additional Remarks

Phases depicted in the previous section are a broad guideline to the Six Sigma-based selection process. We have chosen some activities to exemplify some of the main tasks of each phase; however several equally important activities were not included in our discussion. The following are only two examples of them:

- Once requirements are categorized and weighted, a process to obtain product and vendor information should be carried out. Sources of vendor names include research services, Internet searches, networking, industry group publications and contacts, advertisements in IT journals, and so forth. Also, product information provided by vendors should be assessed. For example, a recent work [4] presents a survey trying to evaluate how much of the information required to assess COTS components is actually available from vendors' information. The main goal of the survey is to ana-

lyse the current gap between the required and provided information, so refinement of quality models can be guided to reduce the gap, yielding in more realistic attributes and metrics.

- Scenarios have been widely used during design as a method to compare design alternatives and to express the particular instances of each quality attribute important to the customer of a system. Scenarios differ widely in breadth and scope, and its appropriate selection is not straightforward. Our use of scenarios is a brief description of some anticipated or desired use of a system. We emphasise the use of scenarios appropriate to all roles involving a system. The architect role is one widely considered but we also have roles for the system composer, the reuse architect, and others, depending on the domain. It is important when analysing a system that all roles relevant to that system be considered since design decisions may be made to accommodate any of the roles. The process of choosing scenarios for analysis forces designers to consider the future uses of, and changes to, the system. It also forces to consider non-functional properties that should be properly measured during the COTS selection process.

5 Conclusion

Six Sigma translates customer's needs into separate tasks and defines the optimal specification for each, depending on how the tasks interact. Based on what the process reveals, the steps that follow can have a powerful effect on the quality of products and customer services as well as the professional development of employees.

In this paper, we have introduced a procedure for selecting and integrating COTS components based on a Six Sigma approach. Its phases were depicted by introducing some techniques and measures, which would help in establishing a formal basis for applying the approach. Besides, the presence of specific measures allows stakeholders to make fact-based decisions improving the analysis of COTS candidates. We have shown, by using a simple example, how this can be done.

However, our proposal needs further validation. To do so, some empirically cases are currently carried out on the domain of E-payment systems. Among others, we are analysing the possibility of dealing with the large volumes of data generated and with low quality information provided by COTS component's suppliers.

References

1. Alexander R. and Blackburn M. Component Assessment Using Specification-Based Analysis and Testing. *Technical Report SPC-98095-CMC, Software Productivity Consortium* (1999).
2. Alves C. and Filkenstein A. Challenges in COTS-Decision Making: A Goal-Driven Requirements Engineering Perspective. In *Proceedings of the Fourteenth International Conference on Software Engineering and Knowledge Engineering,* SEKE'02 (2002).

3. Ballurio K., B. Scalzo, and L. Rose. Risk Reduction in COTS Software Selection with BASIS. In *Proceedings of the First International Conference on COTS-Based Software Systems,* Springer Verlag LNCS 2255, pages 31–43, (2002).
4. Bertoa M., Troya J., and Vallecillo A. A Survey on the Quality Information Provided by Software Component Vendors. In *Proceedings of the 7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE),* (2003).
5. Cechich et al.: *Component-Based Software Quality: Methods and Techniques,* Springer-Verlag LNCS 2693 (2003)
6. Cechich A. and Piattini M.: Defining Stability for Component Integration Assessment. In *Proceedings of the 5th International Conference on Enterprise Information Systems,* ICEIS, pages 251-256, (2003)
7. Cechich A. and Piattini M. On the Measurement of COTS Functional Suitability. In *Proceedings of the 3rd International Conference on COTS-based Software Systems,* ICCBSS, (to appear February 2004)
8. Chung L., Nixon B., Yu E., and Mylopoulos J. *Non-Functional Requirements in Software Engineering.* Kluwer Academic Publisher, (2000).
9. Chung L. and N. Subramanian. Process-Oriented Metrics for Software Architecture Adaptability. In *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering (RE '01),* pages 310–312, (2001).
10. Deline R. A Catalog of Techniques for Resolving Packaging Mismatch. In *Proceedings of the Symposium on Software Reusability,* Los Angeles, CA, (1999).
11. De Feo J. and Bar-El Z. Creating Strategic change more efficiently with a new Design for Six Sigma process, *Journal of Change Management,* vol. 3(1), pages 60-80 (2002)
12. Gack A. and Robinson K. Integrating Improvement Initiatives: Connecting Six Sigma for Software, CMMI, Personal Software Process and Team Software Process. *Software Quality Journal* 5 (4), 5-13
13. I* homepage, http://www.cs.toronto.edu/km/istar
14. ISO International Standard ISO/IEC 9126. ISO/IEC 9126 - Information technology - Software product evaluation - Quality characteristics and guidelines for their use, (2001).
15. Kaiya H., Horai H., and Saeki M. AGORA: Attributed Goal-Oriented Requirements Analysis Method. In *Proceedings of the IEEE International Conference on Requirements Engineering,* RE'02 (2002).
16. KAOS homepage, http://www.info.ucl.ac.be/research/projects/AVL/ReqEng.html
17. Saaty T.L. *The Analytic Hierarchy Process.* McGraw-Hill (1990).
18. Sedigh-Ali S., A. Ghafoor, and R. Paul. Metrics-Based Framework for Decision Making in COTS-Based Software Systems. In *Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering* (HASE'02), pages 243–244, (2002).
19. Tayntor C. *Six Sigma Software Development.* Auerbach Publications (2002)

Using Dynamic Modeling and Simulation to Improve the COTS Software Process

Mercedes Ruiz[1], Isabel Ramos[2], and Miguel Toro[2]

[1] Department of Computer Languages and Systems
Escuela Superior de Ingeniería. University of Cádiz
C/ Chile, n°1. 11003 – Cádiz (Spain)
mercedes.ruiz@uca.es
[2] Escuela Técnica Superior de Ingeniería Informática. University of Seville
Avda. Reina Mercedes, s/n. 41012 – Seville (Spain)
{isabel.ramos, miguel.toro}@lsi.us.es

Abstract. In the last several years, software industry has undergone a significant transition to the use of existing component products in building systems. Nowadays, more and more solutions are built by integrating Commercial-Off-The-Shelf (COTS) products rather than building from scratch. This new approach for software development has specific features that add new factors that need to be taken into account to successfully face software development. In this paper, we present the first results of developing a dynamic simulation model to model and simulate the COTS-based software development process with the aim of helping to understand the specific features of this kind of software development, and design and evaluate software process improvements. An example of how to use these dynamic simulation models to study how the system integration starting point affects the main project variables is shown.

1 Introduction

In the software industry, demands for new services are outpacing our ability to develop and manage them. Current systems have stringent demands in scalability, reliability, and real-time interaction with other elements. In an attempt to meet the requirements of current systems, new paradigms, methods and processes have been developed by the software engineering community. One of them, the component-based software engineering (CBSE), promotes the building of new systems by incorporating pre-existing software with the aim of lowering overall development and maintenance costs, as well as involving less development time. The practice of 'buy, don't build' was initially introduced by Fred Brooks in 1987 [3]. Since then, the features and trends of new software systems have made nothing but support the benefits of building by integration and not from scratch.

Nowadays, creating, deploying, and offering a new software system often requires complex interactions between several disparate systems, some of which may be legacy ones. The problem is how to achieve this integration in a speedy, cost-

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 568–581, 2004.

effective, flexible manner. It is important to minimize costs for building and maintaining integration solutions. Over the last decades, there have been several approaches to solve the problem of integrating systems. Integration techniques can be classified into three generations attending to their evolution over time.

First-Generation integration techniques were the first to appear. This approach coincides with the first attempts of the software industry in software reutilization, and it is based on the concept of point-to-point integration. It becomes evident that this solution to the problem of integration was soon observed impractical because of two factors. First, the number of interfaces required grows exponentially with the number of components to integrate. Second, the impact of minor changes, such as that of adding a new component, is significant, turning maintenance into a nightmare.

Second-Generation integration techniques, try to solve the problem of the increasing number of interfaces by reducing them to a linear increase through the use of middleware. This solution requires interfacing each application to a 'data bus' using, for instance, the Common Object Request Broker Architecture (CORBA) [7] and XML [14] as the data format for the interchange of data through this so-called data bus.

Third-Generation integration techniques encourage the concept of extreme integration. This extreme integration is aimed to obtain a better understanding of the requirements, an easier maintenance, and therefore, an easier accommodation of changes in the system developed. Within this extreme integration approach, the concept of building software using certain components called Commercial-Off-The-Shelf (COTS) products has become a topic of research, development, and production. COTS refers to a particular type of software component which is purchased from and supported by a third party and that is not customized, or is only minimally customized. COTS software has the potential to save both time and money in the software development process and it is a common way of developing software nowadays. Some examples of the systems developed under this approach are: Internet tools and browsers, CASE tools, GUI generators, code generators, database management systems, Geographic Information Systems (GIS), office automation software, and operating systems, among others.

Each COTS software component used means less code that needs to be designed and implemented by the developers. However, the developer is faced with the problem of ensuring that the COTS product does perform the functionality that it claims to perform, that is not intentionally perform functionality to be harmful to the system, that it will not adversely affect the system, and that it can robustly respond to failures and anomalous inputs to prevent errors from propagating through the entire system. Furthermore, the use of COTS products in software development can require a considerable integration effort leading to the research of new models or means to better manage this effort.

Whereas much of the research effort in the area is mainly focussed on the development of methodologies to better document, search and evaluate components, or the improvement of the trading process, it is also necessary to invest some research effort to help in the understanding and improvement of the specific features of the COTS software process. In this paper, we propose the utilization of System Dynamics simulation models to help understand and improve the COTS-based software development process. The structure of the paper is as follows. Section 2 introduces the motivation and problem definition, the concepts of software process modeling and simulation using system dynamics models, and describes in depth the model

developed to analyze the phases of glueware development and system integration. An example of how these dynamic models can be used to simulate different scenarios and analyze its results is included in Section 3. Finally, Section 4 summarizes the paper and draws the conclusions and further works.

2 Developing the Simulation Model for COTS Process

2.1 Motivation and Problem Definition

Our main objective is to develop a simulation model to help understand and improve the COTS software process. In the building of this simulation model, several issues regarding modularity, abstraction and reusability have been taken into account. With the aim of obtaining a certain product after this research effort, our goal has been the development of the dynamic simulation modules that can be added to, or literally "plugged" into, an existing framework of dynamic modules that simulate the traditional software process. Following this approach, our intention is to build new dynamic models combining the existing dynamic modules that help in the design of improvement initiatives aimed to achieve higher maturity levels within organizations, with the new ones that have been developed. The dynamic model resulting from the collaboration of the previous modules (for the traditional software development) and the new ones (specific for the COTS software development) will be able to simulate the complete life cycle.

2.2 Simulation Approach

In the simulation domain there are multiple strategies to build models. Among them, there are two main approaches: continuous and discrete modeling. The continuous simulation technique is based on System Dynamics [1]. A continuous simulation model represents the interactions between key process factors, as a set of differential equations, where time is increased step by step. Frequently, the metaphor of a system of interconnected tanks filled with fluid is used to exemplify the ideas underlying this kind of modeling approach.

On the other hand, discrete modeling is based on the metaphor of a queuing network, where time advances when a discrete event occurs. When this happens, an associated action takes place which, in most occasions, can imply placing a new event in the queue. Time is always advanced to the next event, so it can be difficult to integrate continually changing variables.

Since the purpose of this study is to model and visualize process mechanisms, continuous modeling has been used. This technique also allows to include systems thinking and it is considered to be better than the discrete event model at showing qualitative relationships [13].

Traditionally, three important drawbacks have been claimed against the use of dynamic simulation models in the software industry: the level of education or expertise needed to develop and use the models, the effort required to be invested to model the organization processes, and the lack of data available to validate and populate the final models. [8]

Having specific education in order to be able to use simulation models is no more a requirement. Current simulation tools allow to develop user-friendly interfaces capable of hiding all the mathematical details of models, and enabling to carry out simulation games easily.

In an attempt to initiate a research effort aimed to ease the objections previously mentioned, we developed a Dynamic Integrated Framework for Software Process Improvement (DIFSPI) [10, 11, and 12]. This framework had been originally designed with the aim of creating both a conceptual and formal framework, and a working environment to help in the achievement of higher maturity levels according to CMM [6]. The main issues underlying this integrated framework follow. First, during the process of model building, the project manager may gain much new insight into those aspects of the development process that mostly influence the success of the project (time, cost and quality). Second, having the possibility of gaming with the DIFSPI, it allows project managers to better understand the underlying dynamics of the software process. As a consequence, several process improvement suggestions may easily be designed and, most importantly, analyzed using simulation of scenarios. Third, templates and guidelines for a metrics collection program may be almost automatically derived from the requirements of the dynamic modules. Fourth, the approach of abstraction and encapsulation followed to develop the dynamic modules makes it possible to easily instantiate a dynamic model using different dynamic modules which can be plugged in the final model. Finally, the combination of the dynamic approach with other techniques allows project managers to perform complete analysis and quantification of the effects and the benefits of different software process improvements. All these features combined in the framework intend to help organizations to design and execute more mature processes and, therefore, to increase their maturity level.

Although there are some significant applications of System Dynamics to model and simulate the traditional software process, little has been done regarding the modeling and simulation of the COTS-based systems in the sense of this study, except Kim's work [4]. However, concepts such as modularization, abstraction, encapsulation and reutilization that have been widely applied in the field of computer programming, and which have not been so commonly applied in the field of System Dynamics, are indeed strongly used within the framework proposed.

2.3 COTS Process Modeled

Like other software development processes, the process of building a COTS-based system starts with the definition of the system requirements. Once the system requirements have been gathered and reviewed, the processes of COTS identification, evaluation, and selection begin. COTS identification consists of Web searches, product literature surveys and reviews, identification of other reusable system components, and recommendations from external sources. As COTS components are identified, the evaluation and selection processes start. COTS evaluation steps include prototyping, vendor demonstrations, and in-depth review of literature such as manuals and user guides. Vendor training sites and availability are considered. Procurement issues surface such as development fees for added requirements, licensing and maintenance fees, and sustaining engineering support.

It can be said that there are as many COTS software development processes as organizations are applying the principles of integration to develop software. For this study, the suggested process resulting from the survey carried out by the Software Engineering Laboratory (SEL) has been used [5]. This process is targeted to COTS-based projects using several peer COTS or one COTS integrated with a considerable amount of new developed software.

According to the conclusions of this study, the main phases of the COTS software process are: requirements, design, coding and integration. Most phases encompass activities specific to COTS-based development. Figure 1 illustrates the proposed COTS process. It is important to notice that the horizontal line in the figure graphically separates the two tracks existing in COTS-based projects, that is, traditional activities and COTS-specific activities (highlighted in shadowed boxes).

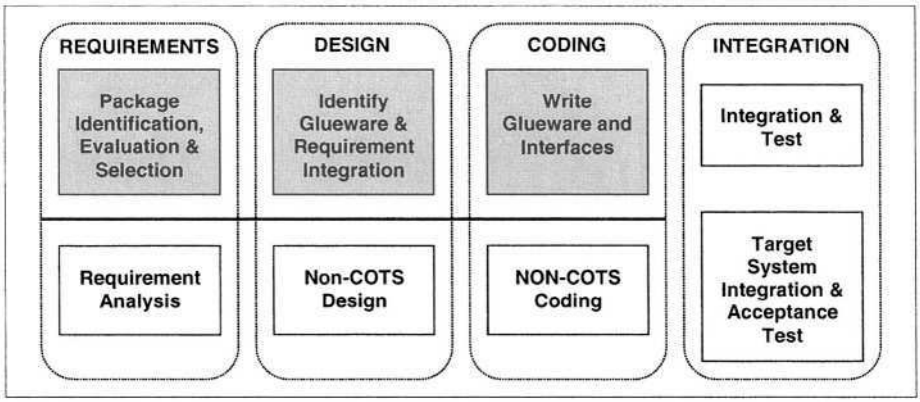


Fig. 1. Proposed COTS Process

It is important to notice that the parallelism between the COTS-specific activities and the traditional activities is strongly emphasized using this process. Hence, a specific effort to support the same parallelism between theses activities in the simulated process has been needed.

2.4 Development of the Simulation Modules

For the purpose of this study, new dynamic modules have been developed and added to the DIFSPI. These dynamic modules are aimed to model the structure, relationships, and behavior of the following processes:

- Glueware Development. Glueware is the new code needed to get a COTS component and integrate it into a larger system that can be the target system or another component that needs to be integrated in a later phase. This special kind of code is considered to be one of the following: 1) any code required to facilitate information or data exchange between a COTS component and the application, 2) any code needed to "hook" the COTS component into the application, even if it may not necessarily facilitate data exchange, and 3) any code needed to provide

functionality that was originally intended to be provided by the COTS component, and which must interact with the COTS component [2].

- Application Integration. The integration process varies a great deal from project to project, depending on which and how many COTS products are being used. At system integration and testing the COTS packages are treated as black boxes. The final integrated system is made up from the application system, the COTS packages, and the glueware that has been needed to be developed.

- Another non-process dynamic module has been integrated in the framework to model and facilitate the analysis of cost effects of different improvement initiatives. This new module gathers together all the parameters that have an influence on the COTS-based system development according to the COCOTS model [2].

The following sections describe in depth the components of these modules.

Glueware Development

This dynamic module represents the process of glueware development. The development of glueware can be achieved through the following phases:

- First, the requirements for the glueware to be developed must be elicitated and analyzed.
- Second, these requirements constitute the input to the design phase, and
- Finally, the designed product enters the phase of coding.

Figure 2 illustrates a simplified stock and flow diagram of this module.

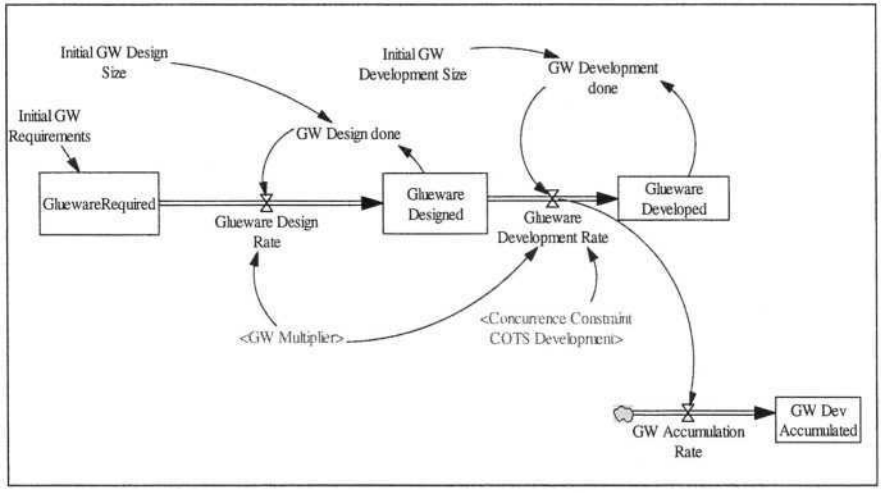


Fig. 2. Simplified stock and flow diagram for Glueware development

As it can be seen, each one of the phases previously mentioned is represented by a level variable. Each level variable is initially populated with the estimated size of the product that needs to be developed during that phase. The meaning of each of these level variables follows:

- `Glueware Required`: Glueware that needs to be developed, measured in number of tasks.
- `Glueware Designed`: Glueware that has been designed, in number of tasks.
- `Glueware Developed`: Glueware that has been coded, in number of tasks.
- `GW Dev Accumulated`: Glueware that has been coded, in number of tasks.

By applying a development rate, work flows from a level to the following. Rates mainly depend on the productivity of the personnel assigned to the development of each phase. They also are influenced by other factors such as the GW Multiplier *(Glueware Multiplier),* or component drivers (see section COTS component factors).

The variable `Concurrence constraint COTS development` determines when the activities of glueware development can start. Figure 3 shows the diagram for the computation of this variable. `Percentage of GW designed needed to development` keeps the percentage of design tasks that need to be accomplished before the implementation or coding phase can be initiated. Knowing the initial size of this phase and the amount of tasks that have been achieved at each moment (`initial GW Design Size` and `Glueware Designed`, respectively), it is possible to calculate the fraction of glueware that has been designed at a given time. By comparing this value with the parameter that keeps the percentage needed to begin development, it is possible to model the decision of start the following phase. Hence, `Concurrence constraint COTS development` will act as a flag variable that helps to determine the beginning of the coding phase.

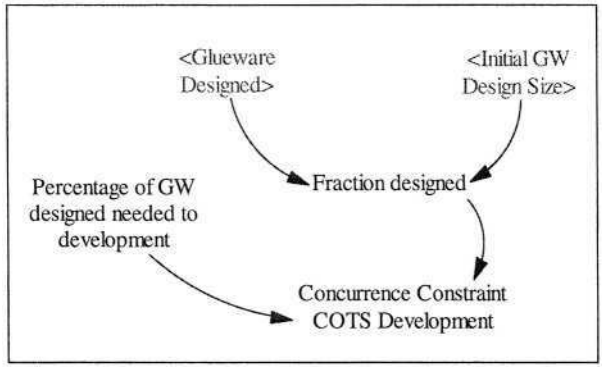


Fig. 3. Concurrence constraint for COTS development modeling

There is another group of variables that need an explanation now because the ideas underlying their modeling are used in almost every module of this framework. These variables do not have a proper semantic regarding the system under modeling, but are

needed to shut down the generation of non-real values in the model. In order to stop activity on task completion, the rates variables are all affected by some other flag variables (`GW design done, GW development done`). These flag variables act in a similar way of the variables that model the concurrence constraints in the model. By comparing the estimated size of a task with the current size accomplished, the percentage of achievement is calculated. When this percentage is 100%, these flags are activated to indicate the end of the activity, and hence, the respective development rate falls to a value of zero.

The last level variable GW Dev Accumulated, does not serve to a specific semantic feature of the problem, and it is only maintained as an accumulation variable in order to know, at every moment, the amount of glueware that has been coded. Without this variable, it would not be possible to know how much work has been completed as it flows from a level to the following one (in this case, glueware flows to the system integration phase).

Application Integration

Depending on the type of application, the amount of application development needed to obtain the final product can vary, but it is obvious that first, a process of application development is needed before the integration of the components can start. This module represents the application development and integration processes. Again, like the previous development process described above, it consists of three level variables, one for each phase of the development process, plus an additional level to represent the process of system integration. Figure 4 shows the simplified stock and flow diagram of this module. It can be seen that this module is made of two subsystems: the first one deals with the process of application development itself. It consists of three level variables, as three is the number of phases of the application development process. The second subsystem models the application integration itself, and it consists of one level variable as the process needs only one phase to be executed.

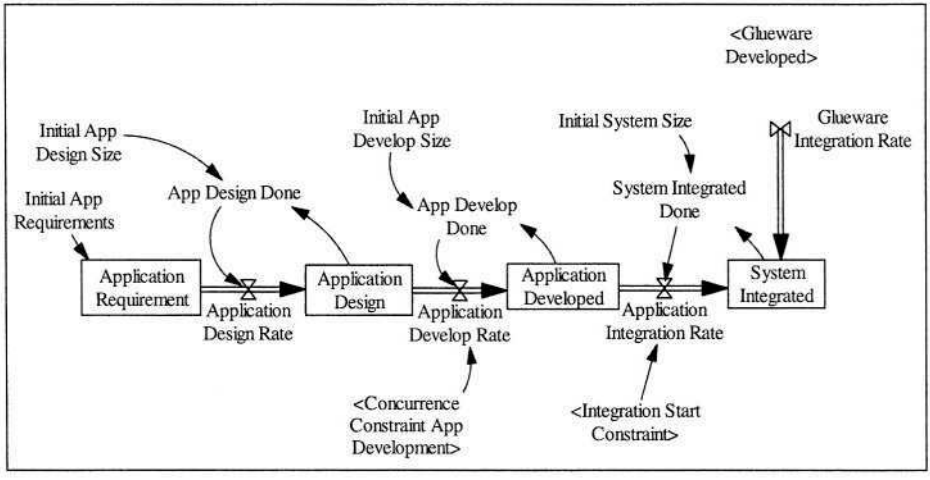


Fig. 4. Simplified stock and flow diagram for Application Integration

The level variables in the model are the following:

- `Application Requirement`: Number of application tasks that represent the application requirements left to be implemented.
- `Application Design`: Number of application tasks that have been designed.
- `Application Developed`: Number of application tasks that have been coded.
- `System Integrated`: Number of application tasks that have been integrated. It consists of application code and glueware.

It is important to notice that the components that control the concurrence constraint for the beginning of the coding phase, and those that control the workflow between whatever two phases are included in this model too. From an abstract point of view, what is done is the instantiation of a generic dynamic meta-constructor that can be used to model a component whenever a task development process is needed to be modeled. This generic meta-constructor does not only help to formalize mathematically the development process, but to effectively implement the principles of modularization and reutilization in System Dynamics. The ultimate representation of this abstraction is the re-engineering of the models using Java™ technology. Using this technology, these constructors have been represented under the concept of interface to define a set of methods and a protocol of behavior that can be then implemented by any class in the class hierarchy.

One important parameter that plays a decisive role in the integration subsystem is `Integration Start Constraint`. This parameter determines the starting point of the integration process based on the glueware that has been developed. For instance, if this parameter is set to 65%, then the integration process will start when the glueware developed has achieved 65% of glueware requirements. Different values in this parameter can have significant effects on the integration process resulting in different quality and costs outcomes. These outcomes can be studied and analyzed using simulation within a non-cost process.

COTS Component Factors

This module represents COTS component factor module. The variable GW multiplier is calculated by multiplying the COTS component drivers suggested by Abts [2]. For the purpose of this study, these factors have been transformed into a set of input parameters of the dynamic model. The parameters take their values from a qualitative domain ranging from very low (point value 0) to very high (point value 5). A brief description of the parameters associated with the COTS component factors follow:

1. COTS Product Documentation: How strongly is effort/productivity affected by the extent to which the COTS product comes with the necessary documentation to install, maintain and use the product? Does the software come with extensive and well written documentation? Or does it come with little documentation?
2. COTS Product Vendor Support: How strongly is effort/productivity affected by the extent to which the vendor offers technical support for the COTS product? Does the vendor provide extensive support for its products? Or no support?

3. COTS Product Ease of Installation: How strongly is effort/productivity affected by the ease or difficulty anticipated to install and integrate the COTS product? Are the interfaces required between the COTS product and the larger system simple or complex?
4. COTS Product Ease of Maintenance or Upgrade: How strongly is effort/productivity affected by the ease or difficulty anticipated to maintain or upgrade the COTS product, particularly after it has been integrated into the larger system? Are upgrades to the COTS product simple to perform, or difficult?
5. COTS Product Ease of Customization: How strongly is effort/productivity affected by the ease or difficulty anticipated to customize or modify the COTS product to make it suitable for use in the larger system if adaptation is necessary? Is customization simple, or difficult?
6. COTS Product Portability: How strongly is effort/productivity affected by the portability of the COTS product across platforms? Is the product easily portable, or difficult to port?
7. COTS Product Ease of Use: How strongly is effort/productivity affected by the ease or difficulty anticipated for the user to operate the COTS product, particularly after it has been integrated into the larger system? Is the product easy, or difficult to use?
8. COTS Product Training: How strongly is effort/productivity affected by the extent of the training the user will require learning to operate the COTS product? Will the user need a lot of training, or little training?
9. COTS Product Dedicated Database: How strongly is effort/productivity affected by the extent to which the COTS product has specialized data needs? Does the product require a specialized database? Or require the population of new elements within an existing database? Or are the product's specialized data needs minimal?

3 First Results of the Simulation

This section shows some of the results obtained with the simulation of the resulting dynamic model that simulates the life cycle illustrated in Figure 1. It is essential to notice here the important lack of real data that organizations have about their own processes. This lack of data often reveals a major problem which is the lack of knowledge and definition for the software process. With the aim of validate and test our modules, we have used data from the literature, mainly those shown in [2].

To show an example of how these models can be used, the effect of different integration starting points on key project variables will be analyzed. The following figures show the sensitivity analysis of the integration start constraint and its effect on the delivery time of the final product (see figure 5), the productivity (see figure 6), and the effort needed to end the project (see figure 7). The results obtained with these simulations can help to determine the most efficient value for this parameter in terms of delivery time, productivity, and cost.

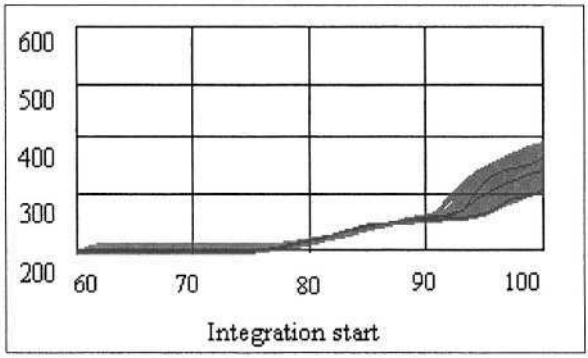


Fig. 5. Sensitivity analysis of the integration start constraint on the delivery time

In order to start the integration phase parts of the glueware code and the application system need to be completed. In the model, the parameter that determines the starting moment of the integration phase is measured as the percentage of the glueware code that needs to be coded before the integration phase can start.

Figure 5 shows different moments to begin the integration phase (expressed in percentage of the glueware coded) and its effect on the delivery time. The simulation outputs suggest that the best delivery time is achieved when the integration phase starts when 60% to 80% of the glueware has been developed.

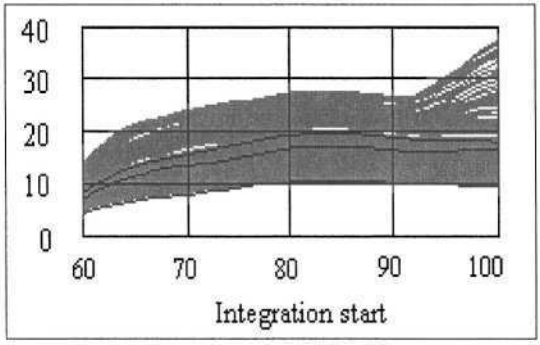


Fig. 6. Sensitivity analysis of the integration start constraint on productivity

Figure 6 shows the influence of the moment at which the integration phase starts on the productivity needed in the project. The simulation outputs suggest that the later the integration process starts, the higher values of productivity need to be achieved to meet the deadline and the objectives of the project.

In Figure 7, the effect of the integration start point on the effort needed to end the project is shown. Best results are again obtained when the integration phase starts when 60% to 80% of the glueware has been developed. For higher values, the effort needed, and therefore the overall costs, grow rapidly.

Sensitivity analysis of the model demonstrates that, from a qualitative point of view, the patterns of behavior shown are consistent with reality and are the ones that one can expect under the scenarios modeled.

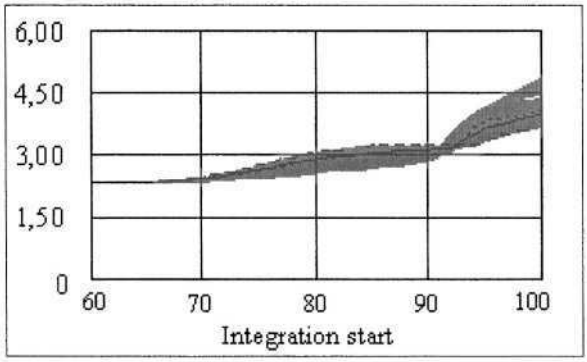


Fig. 7. Sensitivity analysis of the integration start constraint on effort needed

Figure 8 shows the evolution of the global number of tasks developed in each one of the phases of the lifecycle. The number of tasks accomplished shown here is the result of adding the number of tasks accomplished given by each dynamic module that is simulating the specific phase. Note that the figure represents this variable as the percentage of accomplished tasks, showing the temporary evolution of the life cycle in accordance with the sequence and prerequisites restrictions loaded in the inputs of the model.

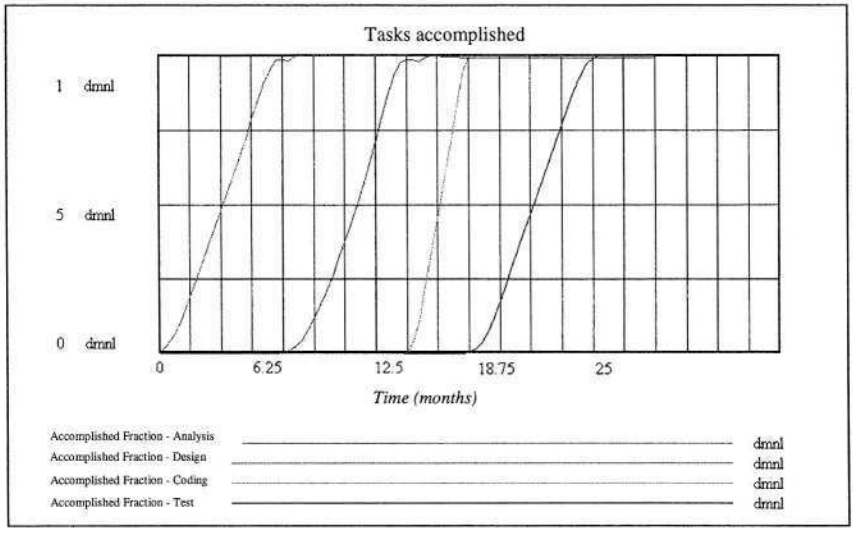


Fig. 8. Tasks accomplished evolution of lifecycle phases

Finally, as it was previously mentioned, all the models within this framework have been re-engineered in order to develop a tool for software process improvement. The new dynamic modules developed for the specific features of COTS have been coded into a family of Java™ classes that inherit and implement the protocol of behavior defined in the framework. As it has been previously said, the tool is intended to asses

in the design of software process improvement. The experiment of different improvement initiatives can be done by populating the model with, either, ordinal values for the parameters, or a range of values where the parameters can vary. As a result of running the simulations, a database is fed with data that can be then analyzed to determine the effect of the improvement initiative. One of the techniques that we are currently using to perform this analysis is automatic learning [9].

4 Conclusions and Further Work

In this work, we have presented the first results of a research effort aimed to the development of a set of dynamic modules to model and simulate COTS-based software development process. These modules are then integrated in a dynamic framework that had been previously developed to help organizations design and evaluate process improvement initiatives [12].

The resulting and enhanced framework integrates a set of traditional techniques for software process management, measuring, monitoring and control, with an extensive use of System Dynamics to build models for the software process. It is important to notice that one of the main features of this dynamic framework is that the process of model building triggers itself a metrics collection program [10]. This metrics collection program contributes to a better understanding of the software process carried out in the organization. In addition, the data collected by these programs are useful too to validate and populate the dynamic modules. In the case of COTS development this is very important as the number of parameters or process drivers that have been proposed in literature is high [2].

As it has been previously said, the building approach followed has the features of modularity, abstraction, and reusability. These are features that intend to ease and promote the use of this kind of modeling, which has been proved to be successful in other areas of engineering, in the software industry.

The conceptual ideas have been implemented to develop a tool using VenSim® (design, development, and testing of the dynamic modules) and Java™ technology. The results obtained from the simulation can be graphically displayed in order to merge in a single view the static data offered by the traditional models with the dynamic data provided by the simulation runs. After this, it is possible to experiment different process improvements and alternative plans just by changing the values of the parameter(s) required and running new simulations. All the results obtained are saved in a database. This database may then be used to feed some machine learning algorithms in order to automatically obtain management and process improvement rules.

Our future work is mainly concentrated on the full development of dynamic modules to model the formal reviews that take place after each of the phases shown in Figure 1. In addition, although the experiments carried out with the current modules prove that they reproduce the expected behavior from a qualitative point of view, we intend to obtain real data to validate them from a quantitative perspective.

Acknowledgements. The authors wish to thank the Comisión Interministerial de Ciencia y Tecnología, Spain, (under grant TIC2001-1143-C03-02) for supporting this research effort.

References

1. Abdel-Hamid T. Madnick S., Software Project Dynamics: an Integrated Approach. Prentice-Hall, 1991.
2. Abts CM. Boehm, B., COTS Software Integration Cost Modeling Study. June, 1997 [on-line] (January 2004) <http://sunset.usc.edu/COCOTS/cocots.html>
3. Brooks FP. No silver bullet. Essence and Accident of Software Engineering. IEEE Computer, 20 (4) 10-19, April, 1987.
4. Kim W. Baik, J. Dynamic Model for COTS Glue Code Development and COTS Integration, 1999 [on-line] (January 2004) <http://sunset.usc.edu/classes/cs599_99/projects/COTS.pdf>
5. Morisio M. Sunderhaft N., Commercial-Off-The-Shelf (COTS): A Survey. December, 2002.
6. Paulk M. Garcia S.M. Chrissis M.B. Bush. M., Key practices of the capability maturity model. Version 1.1 Technical Report CMU/SEI-93-TR-25. Software Engineering Institute, Carnegie Mellon University, Pittsburg, PA (1993)
7. Pope A., The CORBA Reference Guide: Understanding the Common Object Request Broker Architecture. Addison-Wesley, 1998.
8. Raffo D. Spehar G. Nayak U., Generalized Simulation Models: What, Why and How? Proceedings of Software Process Simulation Modeling Workshop, ProSim 2003. Oregon, USA, May 2003.
9. Ramos I. Aguilar J. Riquelme JC. Toro M., A new method for obtaining software project management rules. SQM 2000, Greenwich, UK, June 2000,149-160.
10. Ruiz M. Ramos I. Toro M., A Dynamic Integrated Framework for Software Process Improvement. Software Quality Journal, Vol. 10, N°2, 2002,181-194.
11. Ruiz M. Ramos I. Toro M., An Integrated Framework for Simulation-Based Software Process Improvement. Proceedings of Software Process Simulation Modeling Workshop, ProSim 2003. Oregon, USA, May 2003.
12. Ruiz M. Ramos I. Toro M., Integrating Dynamic Models for CMM-Based Software Process Improvement, in Markku Oivo, Seija Komi-Sirviö (Eds.): Product Focused Software Process Improvement, 4th International Conference, PROFES 2002, Rovaniemi, Finland, December 9-11, 2002, Proceedings. LNCS 2559, Springer-Verlag, 2002,
13. Senge P., The Fifth Discipline. Currency, 1st edition, 1994.
14. Williamson H., XML: The Complete Reference. McGraw-Hill Osborne Media, 1st edition, 2001.

This page intentionally left blank

Author Index