Lecture Notes in Computer Science 2959
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Rick Kazman Dan Port (Eds.)

COTS-Based Software Systems

Third International Conference, ICCBSS 2004
Redondo Beach, CA, USA, February 1-4, 2004
Proceedings

Springer

Volume Editors

Rick Kazman
Dan Port
University of Hawaii
Department of Information Technology Management
Honolulu, HI, 96825, USA
E-mail: {kazman, port}@hawaii.edu

Library of Congress Control Number: 2004104869

CR Subject Classification (1998): K.6.3, D.2, J.1

ISSN 0302-9743
ISBN 3-540-21903-X Springer-Verlag Berlin Heidelberg New York

Springer-Verlag is a part of Springer Science+Business Media

springeronline.com

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Protago-TeX-Production GmbH
Printed on acid-free paper SPIN: 10997864 06/3142 5 4 3 2 1 0

Preface

In the short space of about a decade, Commercial-Off-the-Shelf (COTS) software has evolved through being a relatively minor aspect of software development; a top-management-endorsed silver bullet solution for software development; a disruptive technology requiring people and organizations to extensively rethink their approaches to software development; to an increasingly well-understood software phenomenon for which effective solutions are being developed.

Part of this understanding has been to recognize that different COTS application sectors can be at different stages of this evolution. Some sectors are just beginning to become COTS-intensive. Some have evolved COTS solutions that are very well matched to their problem domain. Others, including most large-scale applications, still involve their developers in rethinking how to adapt their traditional software architectures, processes, management practices, and personnel skills to accommodate economically attractive but complex combinations of powerful but incompletely compatible and independently evolving COTS products.

The series of International Conferences on COTS-Based Software Systems (ICCBSS) has been established as a continuing forum for bringing together CBSS developers, suppliers, and researchers to summarize and discuss progress toward understanding and resolving CBSS problems. This year's conference theme, "Matching Solutions to Problems," reflects this objective. We have been fortunate to have three outstanding keynote speakers, David Carr, Tricia Oberndorf, and Douglas Schmidt, who have contributed significantly both in analyzing CBSS problems and developing better CBSS solutions.

The contributed papers and summaries of workshops, panels, and tutorials in these Proceedings give a good understanding of the nature and directions of evolution of CBSS problems and solutions. As has been my experience with previous ICCBSS Proceedings volumes, I believe that you will find lasting value in the content of the Proceedings.

I would like to express a special note of thanks to all of the members of ICCBSS 2004 organizing committee, program committee, and individual committees listed in the Proceedings. Their capable and dedicated volunteer efforts are what continues to make the ICCBSS series a timely and useful experience and contribution toward improved CBSS practices. I would also like to thank the Northrop Grumman Corporation for its sponsorship of ICCBSS 2004, and the overall sponsoring organizations of the ICCBSS series: the Canadian National Research Council, the CMU Software Engineering Institute, the European Software Institute, and the USC Center for Software Engineering.

January 2004 Barry Boehm, General Chair

Editors' Introduction

Welcome to the Proceedings for the 3rd International Conference of COTS Based Software Systems. The conference is still young, but it is vital and growing fast. This year there were a total of 57 submissions on all aspects of COTS, with about 60% of these coming from the United States and the remainder from Europe, and Asia. Equally encouraging, we had about equal numbers of submissions coming from academia and industry. This shows that ICCBSS is hitting our target audience-both practitioners and researchers interested in the effective use of COTS.

The specific program statistics are as follows:

4 tutorials submitted
3 tutorials accepted

4 panels submitted
3 panels accepted

10 experience presentations submitted
8 experience presentations accepted

39 refereed papers submitted
17 refereed papers accepted

2 invited workshops

We were uniformly impressed with the high quality and broad scope of these submissions. There were about an equal number of technically focused and managerially oriented submissions whose topics generally fell into three tracks: COTS Product Evaluation and Selection, COTS-Based System Definition and Development, and COTS-Based System Evolution and Management.

The superb quality of the submissions and the invited workshops continues to indicate the importance and interest in COTS based system development and issues. With this trend, ICCBSS 2005 will prove to be even more exciting!

Dan Port, Rick Kazman
ICCBSS Proceedings Editors

Organization

ICCBSS 2004 Conference Committee

Planning Committee

General Chair	Barry Boehm (University of Southern California)
Program Chairs	Ceci Albert (Software Engineering Institute) Dan Port (University of Hawaii)
Proceeding Chairs	Rick Kazman (University of Hawaii) Dan Port (University of Hawaii)
Tutorial Chair	Sergio Bandinelli (European Software Institute)
Panels Chair	Ioana Rus (University of Maryland)
Posters Chair	Hakan Erdogmus (National Research Council Canada)
Publicity Chairs	Lisa Brownsword (Software Engineering Institute) David Morera (European Software Institute) Mark Vigder (National Research Council Canada)
Finace & Local Arrangements	Hal Hart (Northrop Grumman)
Chair Emeritus	John Dean (National Research Council Canada)

Program Committee

Cecilia C. Albert – Software Engineering Institute, USA
Sergio Bandinelli – European Software Institute, Spain
David M. Bennett – POWERflex Corporation, Australia
David P. Bentley – South Carolina Research Authority, USA
Ljerka Beus-Dukic – University of Westminster, UK
Jørgen Bøegh – DELTA Danish Electronics, Light, & Acoustics, Denmark
Pere Botella – Universitat Politecnica de Catalunya Barcelona, Spain
William G. Chismar – University of Hawaii, USA
Daniel Dumas – IBM Belgium Software Group, Belgium
Anthony Earl – Sun Microsystems Inc, USA
Suellen Eslinger – The Aerospace Corporation, USA
Rose F. Gamble – University of Tulsa, USA
Suzanne M. Garcia – Software Engineering Institute, USA
Anatol Kark – National Research Council Canada, Canada
Rick Kazman – University of Hawaii and SEI, USA
David Klappholz – Stevens Institute of Tech.& New Jersey CSE, USA
Ron Kohl – R.J. Kohl & Associates, USA
Lech Krzanik – University of Oulu, Finland
Grace A. Lewis – Software Engineering Institute, USA
Fred Long – University of Wales, Aberystwyth

Mike Looney – University of Portsmouth, UK
Ray Madachy – University of Southern California, USA
Jean-Christophe Mielnik – Thales Research & Technology, France
Maurizio Morisio – Politecnico di Torino, Italy
Diane Mularz – MITRE Corp, USA
Michael Ochs – Fraunhofer-Institute for Experimental Software Engineering, Germany

Table of Contents

Tutorials

Workshops

Panels

Posters

Poster Sessions

Papers

Experience Reports

Using eCots Portal for Sharing Information about Software Products on the Internet and in Corporate Intranets

Jean-Christophe Mielnik[1], Vincent Bouthors[2], Stéphane Laurière[3], and Bernard Lang[3]

[1]Thales Research and Technology, France
[2]Jalios, France
[3]INRIA, France

The growing use of COTS software components instead of in-house developments brings non-negligible loss of control of the systems in which they are used, and increases dependency on COTS components' producers, particularly critical in the case of obsolescence. This loss of control and dependency can be compensated only by extremely reliable, accurate and continuously updated knowledge of the software component market and its trends. It is a matter of factual data, not subjected to interpretation, on both the actors (producers, distributors and consulting companies) and the products in this market. These data must be processed on technical, commercial, economical, financial, and legal dimensions. Most industrial groups now try to organize the collection of COTS information to make it available in-house, but the effort is considerable due to the size and variability of the software component market and the difficulty in collecting and updating information. This assessment and selection phase is consequently a hard task for enterprises, particularly for SMEs, which cannot invest enough time or money into COTS management to gain qualified information. Although specialized companies dedicated to technological analysis and market monitoring can bring help in the process of collecting software descriptions, the analysis they provide are often expensive and short- lived. In addition, this information market has not developed a standard for COTS description. This tutorial will describe eCots, a platform that gathers on a common mutualized portal the raw data on COTS products that is held both by producers and by the very large community of COTS users.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, p. 1, 2004

Testing Component-Based Software – Issues, Challenges, and Solutions

Jerry Zeyu Gao[1] and Ye Wu[2]

[1]San Jose State University, USA
[2]George Mason University, USA

Many regard widespread development and reuse of software components as one of the next biggest phenomena for software. However, widespread reuse of a software component with poor quality may lead to disasters. Improper reuse of software components of good quality may also be disastrous. Testing and quality assurance is therefore critical for both software components and component-based software systems. This tutorial provides an in-depth look at the technical issues, challenges, managerial aspects, and needs in testing of components and systems. Moreover, this tutorial reports on the recent advances and research efforts in developing new solutions to solve those problems and meet those needs, from the perspectives of component-based software engineering. The tutorial will discuss the state-of-the-art practice, issues, and challenges, new solutions and research efforts in third-party component testing, component-based program validation, and test automation. The targeted audience includes technical managers, software testing engineers, quality assurance people, and development engineers who are working on component-based software projects. The tutorial will be useful for professionals, researchers, and students interested in understanding the general concepts and methods in component testing and component-based software validation. This tutorial assumes that participants have a general understanding of software engineering and software testing methods, and have some working experience in software development and validation.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, p. 2, 2004.

All You Have to Know When Using Commercial Components to Build Your Software Systems

David Morera

This tutorial introduces the key aspects and implications of using commercial components (COTS) from the market to build software systems. Nowadays most software systems use commercial components in some way. These components may range from database systems, run-times, windows-based user interfaces, specific functional-components, security systems, etc. The integration of third-party elements in a software system has a significant impact on project management, and this impact needs to be taken into account right from the early phases of the project.

While there are many valid business reasons for using commercial components, their use introduces new challenges such as:

- Establishing priorities in system requirements
- Understanding the COTS market, its rules and legal constraints
- Evaluating and selecting the appropriate COTS
- Ensuring the compatibility of the architecture
- Managing configuration updates and evolution

The tutorial addresses all these issues from a practical perspective. If you are a business manager, project leader, or systems analyst that is using or considering to use COTS within your software development activities and you want to fully understand the real implications of this decision, then you should attend this tutorial..

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, p. 3, 2004.

COTS Terminology and Categories: Can We Reach a Consensus?

Betsy Clark[1] and Marco Torchiano[2]

[1]Software Metrics Inc.
betsy@software-metrics.com
[2]Politecnico di Torino
marco.torchiano@polito.it

1 Introduction

There are a variety of classification schemes related to COTS products. Desktop applications, math libraries, operating systems and complex enterprise resource planning (ERP) product suites are all examples of COTS products but differ dramatically in their scope, resource requirements to implement and associated risks. The Terminology Panel held during ICCBSS 2003 was a surprising "sleeper" in generating lively discussion about the lack of standard terms, definitions, and categories. As a follow up, during this workshop, we will examine alternative categorization schemes and COTS-related definitions. Our objective will be to arrive at a consensus where possible and more clearly identify areas where differences of opinion exist.

The workshop will cover the following topics:

- Definition of COTS and related terms (e.g., NDI)
- Types of systems (e.g., COTS-Solution versus COTS-Intensive)
- Components, packages, products
- Types of components (SEI has a classification scheme, there may be others) both functional and problem-oriented classifications
- COTS-Based System, COTS-Based Application
- Organizational issues: familiarity with products/architecture, integrator/vendor relationships
- “Easy COTS” and “Hard COTS” (Basili talked about this in his 2003 ICCBSS Keynote) (perhaps special case of previous item)
- Reference process in terms of typical COTS-related activities (assessment/evaluation, tailoring, glue code), roles involved (vendor, customer, users, integrator), and artifact (COTS component, COTS-based system)
- Workshop Outcome
- The desired outcome is a recommended set of terms and definitions that will be forwarded to the IEEE for inclusion in relevant standards. In addition a list of the relevant factor that could help identifying categories will be de-

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 4–5, 2004.

- fined. The workshop will also produce a workshop summary that the organizers will present during the general session.how to reverse engineer
- how to design product lines with COTS
- how to build domain-specific architectures with COTS
- how to test COTS-based systems

2 Organization

The desired outcome is a recommended set of terms and definitions that will be forwarded to the IEEE for inclusion in relevant standards. In addition a list of the relevant factor that could help identifying categories will be defined. The workshop will also produce a workshop summary that the organizers will present during the general session.

Prospective participants are requested to submit a position paper in advance (see How To Submit below). The organizers will review submissions and then select those to be accepted based on relevance, soundness and novelty. The accepted submissions will be published on the workshop website to allow the participants to know each other's position.

The organizers will give a presentation to set the context for the workshop. The presentation will survey COTS-related terminology and definitions currently in use. The participants will have the opportunity to present their positions. We aim at a lively and productive discussion.

3 Organizing and Program Committee

- Betsy Clark: betsy@software-metrics.com

- Marco Torchiano: marco.torchiano@polito.it

First International Workshop on Incorporating COTS into Software Systems

Alexander Egyed[1] and Dewayne Perry[2]

[1] Teknowledge Corporation, 4640 Admiralty Way, Suite 1010,
Marina Del Rey, CA 90292, USA
aegyed@ieee.org

[2] Electrical and Computer Engineering, The University of Texas at Austin,
Austin TX 78712, USA
perry@ece.utexas.edu

Abstract. This workshop explores innovative ways of integrating COTS software into software systems for purposes often unimagined by their original designers. It emphasizes tools and techniques for plugging COTS into software systems safely and predictably. The past has predominantly explored how to deal with COTS integration during requirements engineering, risk assessment, and selection. This workshop focuses on how to complement ordinary software development with techniques for designing, implementing, and testing COTS integration.

1 Introduction

There is empirical evidence that COTS integration is not like ordinary software development. It has been shown that, for example, writing glue code is several times more difficult than writing ordinary application code. Thus the emphasis of this workshop is on software engineering principles for COTS integration. This includes but is not limited to the following topics:

- how to write the glue code
- how to implement data and control dependencies
- how to mediate between incompatible interfaces
- how to make the COTS tool aware of its surroundings
- how to architect/design/simulate COTS integration
- how to do code generation
- how to resolve stumbling blocks and risks
- how to integrate user interfaces
- how to handle new COTS releases and other evolution issues
- how to reverse engineer
- how to design product lines with COTS
- how to build domain-specific architectures with COTS
- how to test COTS-based systems

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 6–7, 2004.

2 Organization

The call for papers is available at http://www.tuisr.utulsa.edu/iwicss/. Prospective participants may submit a position paper of up to 6 pages. To focus contributions, both theoretical contributions and experience reports are welcome. The submission of a position paper is not mandatory; the workshop is open to anyone who is interested in the problems of COTS integration

The submissions are subject to review by at least three different program committee members and selection is based on relevance, soundness, and novelty.

The workshop is divided into sessions. Topics of the working sessions will be determined based on the distribution of accepted position papers. Each session will cluster presentations of varying lengths where authors will have an opportunity to present the main ideas of their position papers. The presentations shall serve as an opening statement of the sessions, after which there will be time reserved for in-depth discussions of the presentations, related issues, and the implications for future research.

The best position papers will be selected for expansion and subsequent journal publication.

3 Organizing and Program Committee

- Francis Bordeleau
- Lisa Brownsword
- Alexander Egyed
- Rose Gamble
- Anna Liu
- Nenad Medvidovic
- Maurizio Morisio
- Dewayne E Perry
- Judith Stafford
- Tarja Systa
- Ye Wu

Panels Introduction

The panels for ICCBSS 2004 address various hot topics for COTS-based systems development, such as a) the new role of requirements and their corresponding activities; b) information security as a built-in software feature; and c) standardization of components and their interfaces and consensus on reference architecture.

The organizers are experienced practitioners (Judy Kerner from the Aerospace Corporation) and researchers (Carol Sledge from the SEI and Xavier Franch from Universitat Politecnica de Catalunya, Spain). The panels' members are selected representative of various industry domains, from both the COTS vendor and consumer camps, as well as of research institutes and universities, from the US and Europe.

Lively and fruitful discussions are expected between the audience and the panelists. Critical questions will be raised and even if not all the problems will find solutions on the spot - as usually happens with panels - they will certainly increase awareness of issues and will facilitate communication among diverse participants.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, p. 8, 2004.

COTS Components for Spacecraft Ground Systems

Moderator
Judy Kerner (The Aerospace Corportation, USA)

Panelists
David Cadmus (Boeing Satellite Systems, USA)
Dave Dzaran (Space Based Radar Joint Program Office, USA)
Barry Grasso (Integral Systems, Inc., USA)
Sidney Hollander (Aerospace Corporation, USA)
Mike Low (Braxton Technologies, USA)
Ramesh Rangachar (Intelsat Global Service Corporation, USA)

This panel will address issues relating to development and evolution of spacecraft ground systems (SGS) using COTS components. Possibly the biggest challenges facing SGS developers are their increasing dependence on COTS products and the implications of that trend. These include issues relating to granularity of components, applicable interface standards, COTS component standardization, and the consequences of the lack of a consensus reference architecture. Several workshops on SGS architectures and product lines have explored the impacts of ground system development using COTS software. Although the SGS community is still not close to a consensus reference architecture, there is broad agreement on many of the functions and services required in any SGS, and on the interfaces required in order to access those services.

The panel moderator led a workshop on ground system product lines at the Second Software Product Line Conference (SPLC2) in August 2002 in which some of the panelists participated. That workshop achieved significant consensus on the kinds of services needed in a ground system, but the allocation of these functions to components was a subject for tremendous divergence. The annual Ground System Architectures Workshops (GSAW) have been addressing many of these issues both in plenary sessions and in breakout groups. Another focus of concern from these workshops is the definition of relevant interface standards and processes for standardizing them. Identification of appropriate areas for standardization depends to a large extent on agreement on aspects of the architectures as well as on the granularity of the COTS components to be integrated.
This panel will use the SGS context to explore these and other critical COTS issues from multiple perspectives, with the goal of identifying desired future directions and actions.

The panelists have experience in selecting, integrating, and developing COTS software products, for both creation and evolution of SGSs. Panelists include David Cadmus, Manager, Project Systems Engineering, Ground Systems Department, Boeing Satellite Systems; Lt. Col. David J. Dzaran, Chief, System Engineering and

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 9-10, 2004.

Spacecraft Divisions, Space Based Radar Program, USAF Space and Missile Systems Center; Barry Grasso, Command and Control System - Consolidated Deputy Program Manager, Integral Systems, Inc.; Sidney Hollander, Systems Director, The Aerospace Corporation; Mike Low, Ground Segment Program Manager, Braxton Technologies, Inc.; and Ramesh M. Rangachar, Manager, Systems Development, Intelsat Global Service Corporation.

Do We Need Requirements in COTS-Based Software Development?

Moderator

Xavier Franch (Universitat Politecnica de Catalunya, Spain)

Panelists

Barry Boehm (USC Center for Software Engineering)
Neil Maiden (City University, UK)
Mike Moore (Lockheed Martin MDS)

COTS software products radically change the way in which software systems are developed. Existing sequential or iterative development methods and techniques are no longer relevant. Likewise we need new techniques to support the greater focus on selection, customization and integration. One of the most important characteristics of COTS-based development is that it is solution- or feature-driven, that is the final application is heavily influenced by the existing features of the available or selected COTS software systems.

This has major implications for how to develop software systems, and in particular for the role of requirements in this process. One extreme position is that requirements, often seen as one of the most important factors in a successful development project, are no longer important as the stakeholder typically has to work with the COTS software product as is, with little to say over customization. On the other hand, requirements still appear to be important during COTS-related activities to provide selection criteria and drivers for product change. So what is the answer, or answers?

The *11th IEEE International Conference on Requirements Engineering* (RE'03) hosted RECOTS (http://www.lsi.upc.es/events/recots/), a one-day workshop to investigate the role of advanced and practical requirements techniques in COTS-based systems development. The workshop raised some interesting questions that could be used as the starting point for the discussion, such as how to integrate the classical requirements activities in the COTS-based systems development processes, which are the new types of requirements particular to COTS-based systems, and others.

Panelists represent different positions in the COTS community. Prof. Neil Maiden (Center for HCI Design, City University London) provides an applied-research point of view with strong foundations on requirements engineering. He is expected to give arguments supporting the important role that requirements still play in COTS-based development, while recognizing the particularities of these types of systems. Prof. Barry Boehm (USC Center for Software Engineering) is expected to report about IKIWISI (I'll know it when I see it) requirements and their applicability in COTS-based development, and also the role that requirements engineering may play in cost estimation models. Mike Moore (Lockheed Martin MDS, former NASA Goddard

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 11–12, 2004.

Space Flight Center) plays the role of a practitioner involved in the development of systems defined from thousands of requirements, systems that integrate a huge amount of COTS components. Also, a COTS supplier representative is expected to join as panelist to provide the vendor‘s point of view about the topic of the panel, especially by addressing the question of how COTS vendors capture user’s requirements or whether they play a more proactive role as requirements’ generators.

The Added Dimension: Information Security in COTS-Based Software Systems

Moderator

Carol Sledge (Software Engineering Institute, USA)

Panelists

C. Warren Axelrod (Pershing LLC, USA), Steven B. Lipner (Microsoft Corporation), Gail M. Talbott (Lockheed Martin Corporation, USA), Shadi Wegerich (Oracle Corporation, USA)

Issues related to information security and survivability usually are not considered in the requirements, design, acquisition, and maintenance stages of COTS-based software development. Security, if considered at all, is generally an afterthought, bolted on after the application has been developed and integrated, as opposed to an initial requirement of the proposed system or upgrade to the system.

Recent legislation and regulations elevate issues of information security to the Board and C-level, but few organizations routinely and effectively address issues surrounding the management of risks specific to information security in COTS-based systems.

For those who produce COTS products, security is rarely a market differentiator: consumers vote with their dollars, and, at least in the past, appear to have voted for new features, rather than more secure software. However, in many cases, the purchaser does not have a real choice and individually has little influence on the vendor. It is only through buyers joining together that they might have an impact.

The first step to a better solution is to increase awareness of the COTS-based software systems community with respect to the critical information security issues.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, p. 13, 2004.

Poster Title: Systemic Quality of the Component-Based Development Process

Names and affiliations of all contributors/presenters:
Maryoly Ortega. mortega@reacciun.ve Universidad Ezequiel Zamora
María Angélica Pérez. movalles@usb.ve Universidad Simón Bolívar.
José María Troya. troya@lcc.uma.es. Universidad de Málaga.

Main presenter:
Maryoly Ortega

Abstract. The concept of quality is used largely in software development through models in order to improve and certify software products as well as to determine process maturity. The approaches used to specify quality models are oriented toward product or process. However, different studies have treated simultaneously development process maturity and product quality. Component-Based Software Development (CBSD) modifies considerably the traditional software development process. There are some attempts to adjust the accepted quality models into a new component based approach, for example OOSPICE, Cai model, and Preiss Model. This research has as objective identify the model elements needed for quality specification, that considers at the same time the component quality characteristics required for CBSD and the processes that support the presence of these characteristics in the products. The model is based on the feature analysis evaluation method (DESMET), focusing on the processes that support each characteristic. As a result, we obtain a Component-Based Software Development model with a systemic approach. The systemic approach allows us to study the relationship product-process in a twofold framework: the component development process and the component-based software development process. The comprehension of these relationships allows us to improve component quality and ease component integration into applications.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, p. 14, 2004.

Poster Title: COTS Services

Names and affiliations of all contributors/presenters:
Pearl Brereton, Keele University

Main presenter:
Pearl Brereton

Abstract. The focus of COTS-based systems development and research at present is on COTS products and on the building of integrated systems from COTS components. However, the COTS products approach to delivering software functionality has some limitations. In particular, such systems are static in nature and are not easily adapted to meet the needs of emergent organisations - "organisations in a state of continual process change, never arriving, always in transition. An alternative is to provide software functionality as a COTS-based service - which is composed from COTS component services selected, accessed, and paid for on demand. The poster will present a model of COTS-based software service engineering and the key issues that need to be addressed in order to realise the model A demonstrator information broker which has been implemented using a service-oriented architecture will also be described.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, p. 15, 2004.

Poster Title: AIAA (Draft) Guidebook "Managing the Use of Commercial Off-the-Shelf (COTS) Software Components for Mission Critical Systems"

Authors & Affiliations:
Ronald J. Kohl; R. J. Kohl & Associates, Inc.
Nancy M. Sodano; The Charles Stark Draper Laboratory, Inc.
Terry Morris; NASA Langley Research Center
Joe Marshall; BAE SYSTEMS
Shawn Rahmani; The Boeing Company
Richard J. Kwan; AerospaceComputing, Inc.

Main presenters:
Ronald J. Kohl

Abstract. Commercial off-the-shelf (COTS) software products are being included in ever more complex and critical systems. There are clearly advantages to considering the use of COTS in such systems, but given the rigorous needs of such critical systems or subsystems, there are concerns about the suitability of COTS software for such applications. The AIAA's Software Systems Technical Committee and the AIAA's Computer Systems Technical Committee have undertaken an effort to produce a guidebook that captures issues related to the consideration and use of COTS products in these large, complex systems, with a special emphasis on those risks and risk mitigation approaches that relate to mission critical systems. This guidebook identifies a set of characteristics of mission critical systems that makes the consideration, selection and validation processes of COTS products (hardware, software, subsystems, etc.) an emerging factor in systems definition, development and acceptance. The guidebook discusses a large number of risk areas related to using COTS software products in mission critical systems and identifies various mitigation approaches to the risks. It includes detailed processes for selecting and evaluating the products and considerations for using COTS software products within the overall context of the software development life cycle standardized in IEEE/EIA 12207. Mission critical system characteristics such as reliability, safety, availability, maintainability and certification tend to influence whether or not COTS software should be considered for a given application. Once the suitability of COTS software has been determined, then it is possible that additional requirements may be placed on the product and/or the product's vendor for such mission critical applications. Further, it is possible that certain system requirements and expectations may need to be modified because of the inclusion of COTS software products in that system. As COTS products continue to be considered as candidates for inclusion within mission critical systems, there are likely to be additional concerns and factors to emerge that will influence how both acquirers and suppliers decide if and/or when to use COTS products.

Ongoing monitoring of this technology area seems to be warranted.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, p. 16, 2004.

Poster Title: CMMI Compliance in COTS-Based Development

Names and affiliations of all contributors/presenters:
Rick Hefner, Ph.D., Northrop Grumman; rick.hefner@ngc.com

Main presenter:
Rick Hefner, Ph.D.

Abstract. The Capability Maturity Model Integrated (CMMI) provides a reference model of industry best-practices for software and systems engineering. Like its predecessor, the Capability Maturity Model for Software (SW-CMM), the wording of the CMMI seems to reflect custom development of a new software product or system. However, the model was intended to apply to all types of systems development, including the extensive use of COTS.

In the author's experience, project personnel often struggle with interpreting the model in a COTS-based en-vironment. Managers may perceive that some CMMI practices are impossible or inappropriate to perform in a COTS effort. However, improved project performance can result from understand the fundamental principles behind the CMMI practices, and translate them into the appropriate best-practices for COTS-based development. The poster will highlight the critical implications of CMMI compliance on the use of COTS in software and systems engineering, and how the CMMI can reduce risk and rework. A detailed mapping and interpretation of the CMMI planning, management, and engineering practices will be provided.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, p. 17, 2004.

Security in Large System Acquisition[1]

Marshall Abrams[1], Joe Veoni[1], and R. Kris Britton[2]

[1]The MITRE Corporation, 7515 Colshire Drive
McLean, VA 22102
{abrams, jveoni}@mitre.org
[2]National Security Agency, Suite 6740
Ft. George Meade, MD 20755-6740
kris@empire.eclipse.ncsc.mil

1 Introduction

Large systems are typically composed of multiple hardware and software components. Most of the components are Commercial Off The Shelf (COTS) products. All of the COTS components have security properties, as will the custom software, and the resultant system.

Traditionally, incorporating security into the acquisition of these large systems and creating their system security requirements has been an “ad hoc” task. Resulting requirements are often unjustified presenting a poor requirement framework from which to build and evolve the system and lacking any record of rationale for decisions that have been made. Often there is no central repository of recorded security analysis that is kept throughout the life-cycle from project inception through to the maintenance phase. Such rationale and recorded security analysis would be valuable for future decision makers to consider when evolving the system.

This paper describes a method for developing security properties for large system acquisitions that can be used to support not only the acquisition process but can also be used through the lifecycle of the system development. It has been piloted by the Federal Aviation Administration (FAA) for the National Airspace System (NAS).

1.1 About the FAA NAS System

The NAS is a network of interrelated air traffic management systems and support systems including: navigation and landing, surveillance (radar and radar-like systems), ground and air communications, traffic flow management, en route and terminal area control systems, oceanic control systems, avionics systems, and infrastructure manage-

[1] This work was produced for the U.S. Government under Contract DTFA01-01-C-00001 and is subject to Federal Aviation Administration Acquisition Management System Clause 3.5-13, Rights In Data-General, Alt. III and Alt. IV (Oct., 1996). The contents of this document reflect the views of the author and The MITRE Corporation and do not necessarily reflect the views of the FAA or the DOT. Neither the Federal Aviation Administration nor the Department of Transportation makes any warranty or guarantee, expressed or implied, concerning the content or accuracy of these views.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 18–30, 2004.

ment systems. It is an “enterprise system” (a.k.a. “system-of-systems”) comprising hundreds of integrated COTS and custom components. The NAS also includes other assets such as; air navigation facilities, equipment and services; airports or landing areas; aeronautical charts, information and services; rules, regulations and procedures; technical information; and manpower and material. Given the size and complexity of the NAS, it is in a continuous state of modernization and evolution, with many large-scale Information Technology (IT) system acquisitions occurring at any one time. In 2003 alone there are twelve NAS safety related Facilities and Equipment (F&E) programs and eleven NAS efficiency related programs underway [1].

1.2 General Solution

For years the acquisition community and the security community have struggled to come together in a fashion that would result in successful procurement of very large systems with appropriate security properties. A major issue has often been that the security engineers articulated their design using their own paradigm and language while the acquisition community procured systems and components using a different paradigm and language. Only at the very highest levels of reference has the procurement paradigm and security community language come together in an attempt to acquire secure systems. Even at this highest level of specification, most procurement strategies have been based on product level specification without consideration of the security properties of the system as a whole.

One approach to determining the security properties of a composite system has been bottom-up. The security properties of the system are determined by the security properties of the components. There is a 20-year history of failure to develop a composition methodology using a bottom-up approach. Gambel and Hemenway showed a formal proof of the problem with composition and provided a technique for avoiding the problems. Composition can work, but not from the conventional “more is better” approach to system engineering. Their paper showed that minor differences between sibling properties were the proximate cause of multivendor composition problems. [2].

In principle the specification of security properties should be no different than the specification of other properties of the system. We anticipate typical conflicts among specifications and goals. Trade-offs will have to be made when all specifications and goals cannot be achieved. The customer will be involved in some of the decisions concerning trade-offs and will help the integrator decide how to evolve the specifications and goals as well as the system.

Security requirements should be methodically derived. Specifically, they should be driven by threats to the system and security objectives to counter those threats. System security requirements should also embrace the notion of assurance. Assurance is the grounds for confidence that a system meets its security objectives and performs no extraneous functions that may represent an insecure state. Assurance is often measured in developer activities such as the amount of testing or the amount of security analysis performed on a system. Activities that offer this confidence must be considered when security requirements are being formed. In general, no requirement should be adopted without due consideration and rational.

The International Standard ISO/IEC 15408 [3], the *Common Criteria (CC) for Information Technology Security Evaluation*, provides a model for specifying and evaluating security properties of COTS products and small systems.[2] The model offered by the CC can be adopted and extended to specify security properties of very large systems comprising multiple components and systems in a multiyear (evolving) procurement environment. Specifically, the approach includes using the notion of a Security Protection Profile (PP)[3] to be created and used during the course of a large system acquisition as a focal point to drive and support security related architectural and functional decisions.

The FAA adopted this paradigm in the context of the FAA Information Systems Security Program. FAA Order 1370.82 requires a PP for each Information Technology (IT) system being procured by the FAA. To build a security philosophical/policy "blue print" for the NAS and to facilitate the creation of these system security requirements that resulted in a consistent integrated organization security policy, the FAA Chief Scientist for Information Technology established an initiative to provide assistance to concerned parties (e.g., system developers, acquisition specialists, IT security personnel), on preparing PPs. The goal was to culturally and technically integrate information security requirements into the acquisition process in a manner that would support the whole development and maintenance lifecycle of the NAS while establishing a baseline security philosophy. The intent was that all NAS systems would "sing to the same security song sheet". Requirements had to be expressed

- Clearly (preferably in a standards based way)
- Flexibly (to accommodate multiple acquisition strategies, and in particular the prevalent spiral development model used at the FAA
- Completely (to address the entire system life-cycle)
- Understandably (to all stakeholders, and in particular to provide a common ground for the acquisition and security engineering communities)
- Economically (minimize the effort to write an IT system PP) and
- Scaleable to a very large system environments

The result was version 1.0 of the National Airspace System (NAS) System Protection Profile Template (SPPT). As the name suggests, the SPPT is a template that provides a standard set of NAS System IT security specification statements based on acknowledged organizational security philosophy, policy, requirements, and risks. In addition, it offers a common approach to the philosophy of protection and a format for the security requirements author to record the results of the security engineering analysis. The National Institute of Standards and Technology (NIST) has published Special Publication 800-64 *Security Considerations in the Information System Development Life Cycle* [4]. The SPPT provides a worked example of concepts following this guidance. The rest of this paper describes notions embodied in the SPPT and how they can be used to support the acquisition of security properties in large systems.

[2] The CC can be seen as a dictionary of security functional and assurance requirements from which an author can draw requirements to create a security requirements document. Further information concerning the international CC project may be found at http://www.commoncriteria.org.; the U.S. implementation is described at http://niap.nist.gov/cc-scheme/

[3] A Protection Profile is a Common Criteria term for a security specification comprising functional and assurance requirements, environmental statements to which the requirements apply and rationale for the selection of all requirements.

2 Acquiring Security in a "System of Systems"

The SPPT approach produces acquisition requirements for a system-of-systems in essentially top-down fashion. The approach acknowledges that typical systems of this size evolve over the course of many years; personnel and even contractors will change during this period. It provides for (and requires that) security analysis and requirement trade-off analysis be recorded to support architectural and design direction. The security philosophy and properties of the system are specified at the organizational or enterprise level in the context of a security requirements template (i.e., termed the Security Protection Profile, SPPT at the FAA). The template is then used to create requirements for individual systems that are to become part of the enterprise (e.g., the NAS).[4] The purpose of the SPPT is to provide guidance for the creation of individual System Security Requirements (i.e., System Protection Profiles). These requirements then describe security specifications for each individual system as they are procured over the lifecycle of the enterprise system.

When a new system is to be added to the enterprise, the SPPT is applied to create its System Protection Profile (PP). The integrator is then given the PP as part of the procurement package and is responsible for architecting the allocation of functions and structures to the components in order to implement the specified security properties.

2.1 System Protection Profiles

The goal of a System Protection Profile is to create the security requirements for the system that takes into account the system environment, security policies of the enterprise and cost in the context of its integration into the enterprise. A System Protection Profile comprises 7 parts:

1. Introduction
2. System Description
3. System Security Environment
4. Security Objectives
5. Functional Security Specification
6. Assurance Security Specification
7. Rationale

The Introduction states the system security problem. The System Description provides the context for a security specification by describing the Information Technology (IT) system to be procured. The System Security Environment section describes the threats, security policies and the assumptions that will drive the specifications. Each threat, policy or assumption is individually identified with a label (e.g., T.1, T.2) to facilitate rationale and justification statements later. Security Objectives offer a high level description of the IT security measures provided by the system to counter the identified threats and/or satisfy identified organizational security policies. Like

[4] It should be noted that enterprise is used in this context as a synonym for a "very large system." At the FAA and other organizations there could be numerous very large, disparate system-of-systems that would fall under this "enterprise" category.

the threats and policies, each objective is individually labeled (e.g., O.1, O.2). The Functional Requirement Specifications section comprises the technical requirements derived from the CC insofar as possible, or created by the PP author in the style of the CC, that implement the security objectives to ultimately counter threats and implement policies. These too are individually labeled for later use in creating completeness arguments and rationale. The Assurance Security Requirements Specification comprises all evidence and confidence building measures that will be required of the integrator to show that the requirements have been met. The Rationale section justifies all requirements by mapping each individual requirement to a security objective. Each objective is then ultimately mapped to a threat or policy. Using the mapping technique (coupled with prose description) allows all stakeholders (present and future) to understand security trade-offs that were made in the requirements phase.

2.2 The System Protection Profile Template

The purpose of the SPPT, as adopted by the FAA for the NAS, is to provide guidance for the creation of a set of system Protection Profiles that describe security specifications for component systems of the enterprise architecture. These component systems will be procured over the lifecycle of the Enterprise System. The SPPT establishes a set of baseline requirements and provides a common format for PP authors to record the results of the security engineering analysis and provides a set of specification statements, based on the CC, and supporting structures for recording the results of the security analysis.

The SPPT uses the CC as a resource to identify a set of security specifications for PP authors to use in formulating their own System security specifications. Where appropriate, CC text is referenced when the intent of the specification is the same as stated in the CC. In the CC methodology, these are referred to as "refined requirements." The following is an example of a refined requirement from the SPPT where FAU_SEL.1.1 is the reference to the CC class, family, and component:

> The NAS System shall be able to include or exclude auditable events from the set of audited events. (FAU_SEL.1.1)

Specifications not having a CC reference are those that have been created to meet the special needs of the System. In CC terminology these specifications are considered to be "explicit requirements." The following is an example of an explicit requirement:

> The NAS System shall provide the tools for an authorized security administrator to include or exclude auditable events from the set of audited events.

2.3 SPPT Functional Requirements

In the FAA SPPT, the following categories of functional requirements are incorporated:

- **Identification and Authentication.** Identification and Authentication address functions to establish and verify a claimed identity. These functions are required to ensure that entities are associated with the proper Security Attributes (e.g., identity, groups, roles, confidentiality or integrity levels).
- **Security Audit.** Security auditing involves recognizing, recording, storing, and analyzing information related to security relevant activities (i.e., activities controlled by the System Security Policy). The resulting audit records can be examined to determine which security relevant activities have taken place and which entity is responsible for them.
- **Security Management.** Security Management is intended to specify the management of several aspects of the System: security attributes, data, and functions. The different management roles and their interaction, such as separation of capability, are specified.
- **Cryptographic Support.** The System may employ cryptographic functionality to help satisfy several high-level security objectives. These include (but are not limited to): identification and authentication, non-repudiation, trusted path, trusted channel, and data separation.
- **Network Security Protection.** Network Security Protection addresses the responsibility for maintaining the overall security posture of a network.
- **Application Data Protection.** Application Data Protection specifies specifications for System security functions and related policies for protecting System application data.
- **Protection of Security Data and Mechanisms.** Protection of the System Security Data and Mechanisms addresses the integrity and management of the data and mechanisms that implement the System Security Policy.
- **Resource Utilization.** Resource Utilization supports the availability of required resources such as processing capability and/or storage capacity.
- **User Session Access Control.** System Access specifies functional specifications for controlling the establishment of a user's session.
- **Trusted Path.** Trusted Path defines the specifications to establish and maintain trusted communication to or from users and the System. A trusted path may be required for any security-relevant interaction. Trusted path exchanges may be initiated by a user during an interaction with the System, or the System may establish communication with the user via a trusted path.

2.4 Example: Operational Attack and Vulnerability Analysis and Remediation

Operational vulnerability analysis provides specifications to determine whether vulnerabilities exist during operation of NAS System that could allow violations of the NAS System Security Policy. Remediation analysis supports manual or automated risk reduction measures.

Statement of Work Element

a. The developer shall conduct a survey of available product capabilities, perform necessary analysis, and recommend host-based, server based, network-based, and hybrid automated operational vulnerability analysis and remediation tools with which to perform and document an analysis of ways in which the NAS System security policy could be violated. Tools include, but are not limited to, detection and remediation of: viruses and other malicious code, published vulnerabilities, insecure configuration, and unauthorized change.

Functional Security Specification Elements

b. The NAS System shall be able to employ operational vulnerability analysis and remediation tools. Said tools shall be capable of detecting and analyzing stand-alone and network-based violations.

c. The NAS System shall have the capability to audit use of and vulnerabilities identified by the operational vulnerability analysis tools.

2.5 Assurance Requirements

The security assurance requirements include:

- **Configuration Management.** Configuration Management (CM) is one method or means for establishing that the functional requirements and specifications are realized in the implementation. CM meets these objectives by requiring discipline and control in the processes of refinement and modification of the Subsystem and the related information. CM systems are put in place to ensure the integrity of the portions of the Subsystem that they control, by providing a method of tracking any changes, and by ensuring that all changes are authorized. The developer's CM system and data is transferred to the System operator. This is an example of an assurance specification that includes a deliverable that might be considered part of the System.
- **Delivery and Operation.** Delivery and operation specifications address the measures, procedures, and standards concerned with secure delivery, installation, and operational use of the System, ensuring that the security protection offered by the System is not compromised during transfer, installation, start-up, and operation.
- **Development.** Development specifications address the stepwise refinement of the System from the summary specification down to the actual implementation.
  - **Functional Security Specification.** The functional security specification is a high-level description of the user-visible interface and behavior of the security functions of the System. The functional security specification has to show that all the System security specifications are addressed.
  - **High-Level Security Design.** The high-level security design of the System provides a description of the security properties in terms of major structural units (i.e., subsystems) and procedures, and addresses the adequacy of the security functions provided. The high-level security design specifications are intended to provide assurance that the System provides an architecture appropriate to meet the security objectives.
- **Guidance Documents.** These specifications are directed at the understandability, coverage, and completeness of the operational documentation provided by the developer.
  - **Security Administrator Guidance.** Administrator guidance refers to written material that is intended to be used by those persons responsible for configuring, maintaining, and administering the System in a correct and secure manner. Be-

cause the secure operation of the System is dependent upon correct performance, persons responsible for performing these functions are necessarily trusted. Security administrator guidance is intended to help security administrators understand the security functions provided by the System, including both those functions that require the security administrator to perform security-critical actions and those functions that provide security-critical information.

  - **User Guidance.** User guidance refers to material that is intended to be used by non-administrative users of the System, and by others (e.g., programmers) using System external interfaces. User guidance describes the security functions provided by the System and provides instructions and guidelines, including warnings, for its secure use. The user guidance provides a basis for assumptions about the use of the System and a measure of confidence that non-malicious users, application providers, and others exercising the external interfaces of the System will understand the secure operation of the System and will use it as intended.

- **Developer and Consumer Testing.** Testing demonstrates whether the System satisfies the security functional specifications. Supplementary guidance is found in *FAA Security Test and Evaluation* [5].

  - **Analysis of Coverage.** This specification addresses those aspects of testing that deal with completeness of test coverage. The objective is to establish that the System has been tested against its security functional specification in a systematic manner. It addresses the extent to which the System Security Function is tested, and whether or not the testing is sufficiently extensive to demonstrate whether the System Security Function operates as specified.
  - **Analysis of Developer's Functional Tests.** Depth deals with the level of detail to which the developer tests the System. The objective of testing is to counter the risk of missing an error in the development of the System. Testing that exercises specific internal interfaces can provide assurance not only that the System exhibits the desired external security behavior, but also that this behavior stems from correctly operating internal mechanisms. Testing at the level of the mostly COTS system components, in order to demonstrate the presence of any flaws, provides assurance that the System components have been correctly implemented and integrated.
  - **Independent Testing.** Independent testing performed by the customer demonstrates whether the System Security Function perform as specified and helps counter the risk of an incorrect assessment of the test outcomes on the part of the developer that results in the incorrect implementation of the specifications, or overlooks code that is non-compliant with the specifications.

- **Vulnerability Assessment.** Vulnerability Assessment defines specifications directed at the identification of exploitable vulnerabilities introduced in the architecture and design, construction, operation, misuse, or incorrect configuration of the System.

  - **Strength of Security Functions.** Strength of function analysis addresses security functions that are implemented by a probabilistic or permutational mechanism (e.g., a password or hash function). Even if such functions cannot be bypassed, deactivated, or corrupted, it may still be possible to defeat them by direct attack because there is a vulnerability in the concept or implementation of its underlying security mechanisms.

- **Developer Vulnerability Analysis.** Developer vulnerability analysis is performed by the developer to ascertain the presence of vulnerabilities that could allow users to violate the System Security Policy or reduce the security of any other part of the system-of-systems, and to confirm or not confirm that they cannot be exploited in the intended environment.

The SPPT advocates PP authors mandate assurance through active investigation (evaluation) of the IT system in order to determine its security properties.

Evaluation techniques can include, but are not limited to:

- Analysis and checking of process(es) and procedure(s)
- Checking that process(es) and procedure(s) are being applied
- Analysis of the correspondence between system design representations
- Analysis of the system design representation against the specifications
- Verification of proofs
- Analysis of guidance documents
- Analysis of security test plans, procedures, and results
- Independent functional security testing
- Analysis for vulnerabilities (including flaw hypothesis)
- Penetration testing

Developer action (Statement of Work) elements identify the activities that shall be performed by the developer. This set of actions is further qualified by developer provided evidence referenced in the content and presentation of evidence elements (Data Item Description) that identify the evidence required, what the evidence shall demonstrate, and what information the evidence shall convey. The developer actions and content and presentation of evidence define the assurance specifications that are used to represent a developer's responsibilities.

Not every specification stated in the SPPT is needed for every NAS System. For example, an air traffic control system may have significantly different security specifications than that of a radar unit. Policy or architecture may stipulate certain security features that must be included independent of the risk analysis. All other specifications must be based on risk assessment; residual risk acceptance must be the basis for all security specifications and associated mitigation measures. The specifications should be tailored to ultimately be driven by the objectives, threats, vulnerabilities, and available countermeasures that are relevant to the System. The System PP may even have brand new specifications based on new threats that are identified as well as new assumptions of the environment, given the rapidly changing security environment. Ultimately, the System PP authors must use their own judgment and rationalization to arrive at the correct security specifications.

3 Invoking the SPPT as Part of a Contract

It is axiomatic among information security engineers that security concerns should be considered from the very beginning of an acquisition. There are security concerns at each stage of a system's lifecycle, and they need to be anticipated as soon as possible. The security concerns are relevant to mission needs, investment analysis, system engineering, specifications analysis, design, security planning, security administration, and security operations. Security objectives must be balanced against objectives arising from other system engineering disciplines. For example, security must be balanced against other important System demands including safety, performance, schedule, and cost in order be successful. The following fundamental observations should be kept in mind when developing a PP for a specific System:

- There will be trade-offs among system engineering concerns and objectives throughout the lifecycle.
- When the System project is in its early stages; much of the information and analysis that the CC assumes will precede PP development does not yet exist.
- The PP reflects real world business practices in that the project advances even though some relevant analysis and data remain unfinished.
- Most Systems will be developed incrementally over a long lifecycle, during which almost everything may change.
- The PP may be implemented in stages if the System is developed in increments or spirals. Assurance methods and techniques need to be consistent with this approach.

The resultant PP will be incorporated by reference in the contract. The Functional Security Specifications that prescribe what the NAS System shall do are contractual specifications properly referenced in the specifications section of the contract. Instructions to the developer on how to perform the work are considered part of the statement of work (SOW). Specifications levied on the developer to perform analysis or conduct other activities are properly referenced in the SOW. Specifications that describe the contents of a document are candidates for inclusion as Data Item Description (DID) items. The Assurance Specifications in the PP are mostly, if not all, part of the SOW. Assurance specifications that refer to documentation produced by the developer may also require Contract Data Requirements List (CDRL) specifications.

In developing the SPPT, most of the compliance actions have been omitted because the PP is envisioned to become part of contract specifications for the developer. The validation and verification (e.g., IV & V) actions apply to the customer, or a contractor acting on behalf of the customer. A separate *Security Testing Guide* has been developed for the FAA [5], incorporating this material.

3.1 Deferred Decisions

Some requirements may evolve during the System life-cycle. Therefore, the SPPT introduces a notion of Deferred Decisions. Deferral of design decisions is a natural byproduct of the evolutionary system development lifecycle. It enables the developer to take advantage of the latest technology and other advances in IT security and to respond to changed threats. For example, the application of biometric-based authentication may be a preferred choice, due to quality enhancement and decrease in cost, at the time the developer selects specific authentication mechanisms. It is assumed that the developer will be tasked to perform the analysis and design and fill the results in as part of the contract. The PP indicates this deferral with elements of the form "the developer shall recommend." The requirement that the developer make a recommendation implies that the customer retains approval authority. The tasking of the developer indicates that this element belongs in the SOW.

To some extent the SPPT is written to encompass the anticipated eventual functionality and connectivity of the system-of-systems, not only the present or near-term configuration. Many decisions must be left unresolved in the SPPT to be resolved at the time that a specific task order is negotiated instructing the developer to implement a function. While the specifications section of the SPPT clearly identifies these deferred decisions and the responsible parties, it does not emphasize that operations have occurred on CC components nor does it emphasize the CC terminology. The security engineer should anticipate the alternative upgrades and identify the cost-effective countermeasures that will be needed.

Not all operations are deferred because necessary analysis and design have not been performed; they are deferred for other reasons. Some operations require knowledge of ambient conditions at the time that the task is executed. Since the System is expected to evolve over many years, the execution of appropriate tasking will occur one or more times during that evolution. For example, the cost-effectiveness of COTS products and the attack scenarios will be assessed periodically.

Some operations are deferred to authorized security administrators who respond to changing ambient conditions. The strength of authenticators and secrets (e.g., passwords and cryptographic keys) may change due to heightened security concerns or advances in attack technology.

4 Experience at the FAA

The work described herein was initiated as part of the development of specifications for system one[5], a new addition to the NAS system-of-systems. The first two authors were charged to write the security specifications part as part of a solicitation. After an extensive effort including architecture and design studies and risk analysis, we decided to try to adapt the PP concept to a large system acquisition. The system one project management appointed an FAA staffer as security lead for the project. We consider this one measure–making a specified individual responsible for the IT security–the single most effective move that project management can make.

Although the CC claimed applicability to systems, experience indicated that these were small systems consisting of a hardware platform, and operating systems, and a few applications that were to be evaluated according to the CC evaluation paradigm. In contrast, we were specifying a system-of-systems involving multiple legacy and new computer systems connected by a NAS network encompassing the entire U. S. airspace and connected to the international civil aviation community. There was no interest in mutual recognition. This was a one-of-a-kind system. The CC was selected as a starting point because of the extensive prior work that the authors could leverage. The discipline of explicitly describing the security environment and stating assumptions was very appealing. The explicit gradations of functional and assurance specifications had no peer.

[5] The name is suppressed for the obvious reasons.

After completion of the system one specifications we recognized that there was a great commonality among NAS systems, at least with respect to their security specifications. We proposed generalizing the system one PP into a template that could be used to generate other PPs for other new NAS systems. FAA management saw the potential and took ownership of the concept. We interested the National Information Assurance Partnership (NIAP)–a joint effort of NIST and the National Security Agency (NSA) and the U.S. member of the CC consortium–in the potential for system-level PPs. NIAP provided the third author as a member of the FAA authors team to develop a NAS SPPT. Upon completion of version one of the FAA SPPT, NIAP sponsored a workshop to publicize the work. The General Accounting Office (GAO) joined in sponsoring the workshop.

FAA system two developed its PP more-or-less simultaneously with the development of the SPPT. The system two solicitation did not include adequate assurance specifications or Data Item Descriptions (DIDs). There have been problems with the quality of deliveries as a consequence resulting in adverse schedule and cost impacts to the program.

System three was initially specified with vague high-level security requirements that were more motherhood and apple pie than testable criteria. Following September 11, 2001, and for other reasons, system three was re-baselined. An engineer intimately acquainted with system three was given the SPPT and access to the authors. A highly creditable SPPT was produced along with SOW and DID statements.

System four is our best success story so far. A project engineer used the SPPT to write the specifications. Following award, the developer suggested revisions to the specifications based on its architecture and design, with which the FAA concurred.

Lessons learned from these systems and the FAA review process have fed into SPPT version two and its guidance. Writing good assurance specifications into the SOW, CDRLs, and DIDs is essential to success. Experience and knowledge in the system domain, security, and acquisition are required in order to produce good specifications, leading to a secure system. Skilled practitioners of both disciplines can team together to achieve success. The SOW and DID materials can be part of the common template. A close working relationship between government and developer with incremental feedback enhances the probability for success and controlling adverse cost and schedule impact.

There is a difference between “requirements” and “specifications” that is more often honored in the breach than in the observance[6]. Among other distinctions, requirements are high level, generated at the time of project definition as part of mission need and investment analysis (e.g., budget determination), while specifications are much more detailed, as discussed herein. We found that work on the SPPT (specifications) increased an awareness of need for improved requirements. The idea that requirements evolve into specifications has been proposed. It is almost a tautology to observe that adequate requirements will lead to an achievable schedule and adequate budget.

There is a regrettable tendency in acquisitions to economize on pre-solicitation costs at the risk of much greater costs later in the life-cycle. It is imperative that architecture and design be security-aware. Risk analysis must be conducted on conceptual architectures and designs as well as on implemented systems in both the labora-

[6] Apologies to Shakespeare, Hamlet. Act i. Sc. 4.

tory test-bed and the field. Acquisition project management may require assistance from acquisition and security oversight bodies.

5 Conclusion

The SPPT concept has proven very useful in the FAA. Establishing a central enterprise security requirements template has created a common security philosophy from which to evolve the enterprise. Incorporating it into the acquisition process has made security planning an integral part of the general system planning effort. In general, it has expedited the production of higher quality security specifications in much less time and cost than done previously.

Grounding the SPPT in the CC has helped ensure that the resulting security specifications are complete in terms of: 1) the different security components appropriate for a system 2) clear language for the individual specification components, and 3) coverage against all the threats, security policies, and security objectives for the given system. The SPPT attention to the assurance specifications is also beneficial by providing a set of CDRLs and DIDs that fit easily into an acquisition SOW to provide the necessary security assurances. Based on the FAA experience, other organizations would benefit from the development of their own SPPT. The FAA's SPPT can be used as a model. The latest versions can be found at
http://www1.faa.gov/aio/common/documents.htm#4-docs and at
http://www1.faa.gov/aio/ChiefSci/index.htm

References

1. *Fiscal Year 2003 FAA Budget in Brief*, available at http://www.faa.gov/aba/ (February 2002).
2. Gamble, D., Hemenway, J.: "The Use of Generic Architectures in System Integration," *Proceedings 18th National Information Systems Security Conference*, pp. 431—446 (October 1995).
3. Common Criteria Project, *Common Criteria for Information Technology Security Evaluation*, version 2.1 (1999). Or, International Standard ISO/IEC 15408 (1999-12); Parts 1-3, Information Technology Security Techniques Common Criteria for IT Security Evaluation (CCITSE). Available from: http://csrc.nist.gov/cc/,
http://www.radium.ncsc.mil/tpep/library/ccitse/ccitse.html, or
http://www.commoncriteria.org/cc/cc.html.
4. Grance, T., J. Hash, and M. Stevens, October 2003, *Security Considerations in the Information System Development Life Cycle*, National Institute of Standards and Technology (NIST) Special Publication (SP) 800-64. Also available at
http://csrc.ncsl.nist.gov/publications/nistpubs/800-64/NIST-SP800-64.pdf**.**
5. Abrams, M. D.: *FAA System Security Testing and Evaluation*, MITRE Technical Report 02W0000059, The MITRE Corporation, McLean, VA (May 2003). Available from:
http://www.mitre.org/work/tech_papers/tech_papers_03/abrams_faa/index.html.

On the Measurement of COTS Functional Suitability

Alejandra Cechich[1] and Mario Piattini[2]

[1] Departamento de Ciencias de la Computación
Universidad Nacional del Comahue, Buenos Aires 1400
Neuquén, Argentina
acechich@uncoma.edu.ar

[2] Grupo Alarcos, Escuela Superior de Informática
Universidad de Castilla-La Mancha, Paseo de la Universidad 4
Ciudad Real, España
Mario.Piattini@uclm.es

Abstract. Within the last years both researchers and practitioners alike have moved beyond establishing COTS quality as an important field to resolving CBS quality problems. However, the science of CBS quality has not yet advanced to the point where there are standard measurement methods, and few enterprises routinely measure COTS quality. Here, a suite of measures is presented to address this problem within a COTS-based software measurement activity. Our measures are based on a formally defined component-based model, aiming at expressing and measuring some aspects of component integrations. Measures are in terms of provided and required services, hence functional suitability might be quantified.

1 Introduction

Software project managers need to make a series of decisions at the beginning of and during projects. Because software development is such a complex and diverse process, predictive models should guide decision making for future projects. This requires having a metrics program in place, collecting project data with a well-defined goal in a metrics repository, and then analysing and processing data to generate models. According to the proposal in [11], metrics can guide risk and quality management, helping reduce risks encountered during planning and execution of CBSD.

Metrics let developers identify and quantify quality attributes in such a way that risks encountered during COTS selection are reduced. For example, the QESTA approach to evaluate COTS components [8] defines for each desired quality one or more metrics, either symbols or numbers. Then, the selected candidate components are each measured against the metrics previously identified. As another example, based on the ISO/IEC 9126 Standard for software product evaluation [10], the proposal in [5] restricts the set of features applicable on COTS components and defines two classes of measurable features: run-time measured features and life cycle measured features.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 31–40, 2004.

All CBS projects require a cost estimate before actual developments can proceed. Usually, the qualities of the desired COTS components are not directly measurable but are instead vague statements about like “acceptable performance”, “small size”, and “high reliability”. Thus, most cost estimates for CBS developments are based on rules of thumb involving some size measure, like adapted lines of code, number of function points added/updated, or more recently, functional density [1, 7, 9]. In practical terms, rules such as functional density imply that there must be a way of comparing one CBS design to another in terms of their functionality, there must be possible to split functionality delivered via COTS from that delivered from scratch, and there must be a way to clearly identify different COTS functionalities [1].

On the other hand, the model introduced in [2] 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. In our proposal, we are adapting this model as a basis for quality measurement. It allows to express the semantics distance in terms of a functional suitability measure, which provides a better identification of the different COTS functionalities.

In section 2 of the paper, we introduce the component-based model for measurement (from [2]) (called here “component mapping diagram”) along with a motivating example. Then, section 3 presents a compact suite of measures – including functional suitability measures. Finally, section 4 addresses conclusions and topics for further research.

2 A Component-Based Model for Measurement

Component architectures divide software components into requiring and providing: some software components can register the services they provide, while other components can subscribe to and consume these services. Components are plugged into a software architecture that connects participating components and enforces interaction rules. The model in [2] supposes that there is an architectural definition of a system, whose behaviour has been depicted by scenarios or using an architecture description language (ADL).

The 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 (S_A) defines a specification S_C for the abstract component type C (i.e. $S_A \Rightarrow S_C$). Any component K_i, that is a concrete instance of C, must conform to the interface and behaviour specified by S_C, as shown in Figure 1 (from [2]).

The process of composing a component K with A is an act of interface and semantic mapping. In this work, only the semantic mapping will be addressed. We focus on incompatibilities derived from behavioural differences between the specification of a component K_i (S_{Ki}) and the specification S_C. Another necessary condition for using K (or a combination of K_i) to satisfy S_C is that the input and output domains of K include some of those specified by S_C. An additional necessary condition is that K provides at least the functional mapping between the domains as specified by S_C.

A typical situation for inconsistency in the functional mappings between S_K and S_C is illustrated by [2] in Figure 2, where the dashed lines indicate mappings with respect to S_C , and the solid lines are mappings with respect to S_K. Note that the input and output domains of S_K and S_C are not equal. Also, the domain of S_C is not included in the domain of S_K, and vice versa for the ranges.

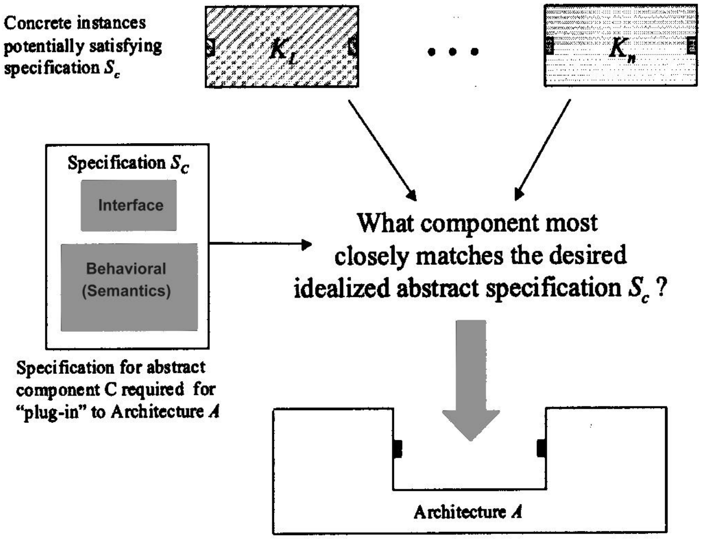

Concrete instances potentially satisfying specification S_c
K_1
• • •
K_n
Specification S_C
Interface
Behavioral (Semantics)
What component most closely matches the desired idealized abstract specification S_c ?
Specification for abstract component C required for "plug-in" to Architecture A
Architecture A

Fig. 1. Instantiation of an abstract component specification

2.1 A Motivating Example: E-payment Components

Authorisation and *Capture* are the two main stages in the processing of a card payment over the Internet. 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. This may take place simultaneously with the authorisation request if the retailer can guarantee a specific delivery time. Otherwise the capture will happen when the goods are shipped.

We suppose the existence of some scenarios describing the two main stages, which represent here a credit card (CCard) payment system. The scenarios will provide an abstract specification of the input and output domains of S_C that might be composed of:

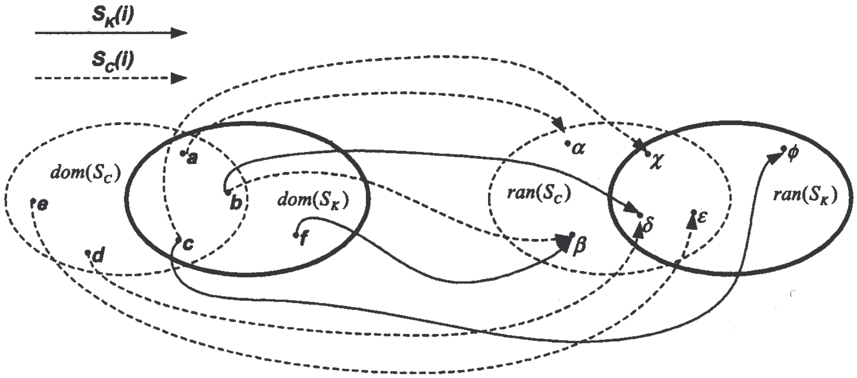


Fig. 2. Functional mappings of S_C and S_K

- Input domain: (AID) Auth_IData{#Card, Cardholder_Name, Exp_Date, Bank_Acc, Amount}; (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. The specification mapping, shown in Figure 3, reveals some inconsistencies that should be analysed. Firstly, 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. Secondly, the input domain of the component K_2 includes more values than the required by S_C, however the mapping satisfies the required functionality. We should note that there is another functionality provided by K_2, which might inject harmful effects to the final composition. Thus a deeper analysis based on previously defined scenarios should be carried out.

3 A Measurement Suite for Functional Suitability

For the measure definitions, we assume a conceptual model with universe of scenarios *Σ*, an abstract specification of a component *X*, a set of components *K* relevant to *X* and called candidate solution *ΣO*, a set of pre-selected components from *ΣO*, called solution *ΣN*, and a mapping component diagram *MXΔ*. In this diagram, $S_C(i)$ represents the map associated to the input value *i* defined in the domain of S_C. This map should provide a valid value on the output domain of S_C, i.e. there is no empty maps or invalid associations. A similar assumption is made on the mappings of S_K.

Let's briefly clarify the concepts associated to *ΣO* and *ΣN*. Candidate components, selected from different sources for evaluation, constitute the members of the set *ΣO*. It could be the case that one of these members does not offer any functionality required by *X*. Hence, an evaluator should not spend more time and effort evaluating other properties or requirements on that component, i.e. the component should be withdrawn from analysis. Then, the solution in which all components potentially contribute with some functionality to get the requirements of *X* is called here *ΣN*.

In the following definitions, we use the symbol "#" for the cardinality of a set. To simplify the analysis, we also assume input/output data as data flows, i.e. data that may aggregate some elemental data. For the credit card example, input/output data are represented by {AID, CID}, {AOD, COD} respectively.

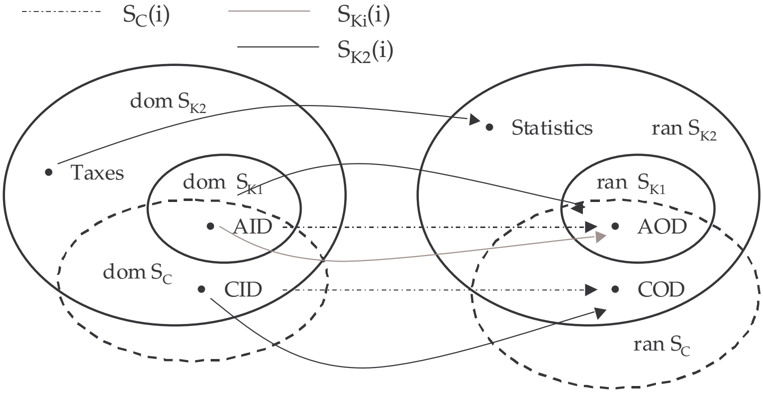


Fig. 3. Functional mappings of S_C and S_{K1}/S_{K2}

3.1 Domain Compatibility Measures

The importance of defining domain compatibility measures comes from the importance of simplifying the COTS selection process. When analysing components, it might be the case that the data required by a concrete component *K* does not semantically match with the data required by its abstract specification *X*. Then, after determining the input/output compatibility, the analysis of the component *K* might stop (depending on the incompatibility detected), avoiding higher selection effort investments.

Table 1 lists the proposed measures for detecting input domain incompatibilities. The measures have been grouped into two main groups: component-level measures and solution-level measures. The first group of metrics 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 *X*. In this case, the second group of metrics evaluates the domain compatibility of all components that constitutes the candidate solution *ΣO*, as we previously defined.

Table 1. Description of the ID-Compatibility measures

Measure Id.	Description
Component-level	
CID_C Compatible Input Domain	The number of input data provided by S_K and required by S_C in the scenario $\mathcal{S}$.
MID_C Missed Input Domain	The number of input data required by S_C in the scenario $\mathcal{S}$ and NOT provided by S_K.
AID_C Added Input Domain	The number of input data NOT required by S_C in the scenario $\mathcal{S}$ and provided by S_K.
CC_{ID} Component Contribution	Percentage in which a component contributes to get the input data required by S_C in the scenario $\mathcal{S}$.
Solution-level	
SN_{ID} Candidate Solution	The number of components that contribute with compatible input data to potentially get the requirements of S_C in the scenario $\mathcal{S}$.
CID_S Compatible Input Domain	The number of input data provided by $\mathcal{SN}$ and required by S_C in the scenario $\mathcal{S}$.
MID_S Missed Input Domain	The number of input data required by S_C in the scenario $\mathcal{S}$ and NOT provided by $\mathcal{SN}$.
AID_S Added Input Domain	The number of input data NOT required by S_C in the scenario $\mathcal{S}$ and provided by $\mathcal{SN}$.
SC_{ID} Solution Contribution	Percentage in which a solution contributes to get the input data required by S_C in the scenario $\mathcal{S}$.

The input domain measure definitions are shown in Table 2. Similarly, a compatibility analysis of the output domain should be done, considering the data provided by the component *K* and using a similar suite of measures.

To clarify the reading, we should note that the expression $\text{CID}_\text{C}(\text{K,C}) \geq 1$ has been included to reduce the candidates for evaluation. This expression limits the analysis of missed and added inputs to those components that have already showed having at least a compatible input data. We should also remark the importance of determining semantics incompatibilities through the use of scenario specifications, even thought the scenario Σ is not explicitly included into our measure definitions. This is due to the fact that we consider the definition of metrics as a process included into a broader measurement process, which defines some activities for setting the measurement context – such as defining scenario specifications or identifying stakeholders [6].

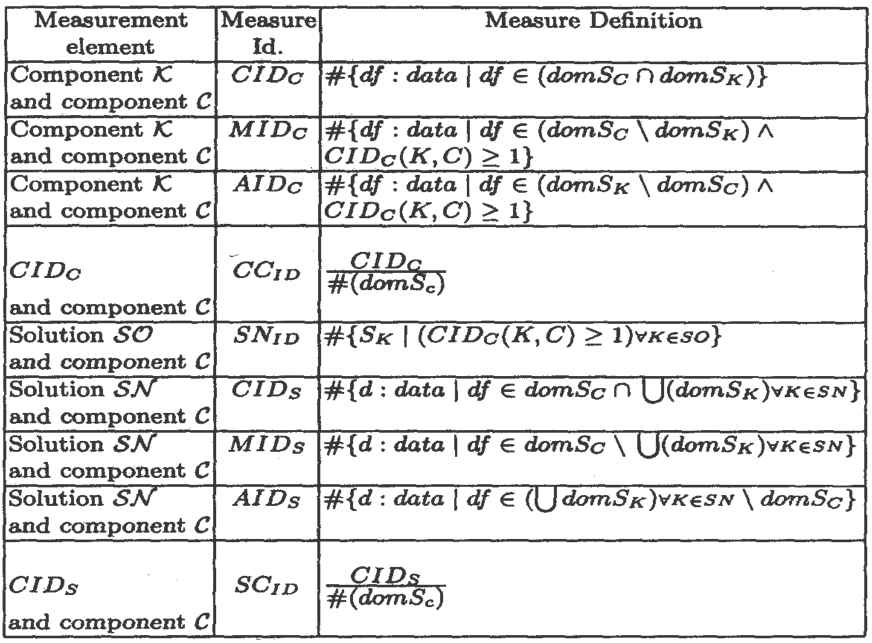
Now, let's calculate the input domain compatibility measures for our credit card example. The input domain of the abstract specification S_C is {AID, CID}, and the input domains for *K1* and *K2* are {AID} and {AID, CID} respectively.

The following values of the measures:

$$\text{CID}_\text{C}(K1) = 1;\ \ \text{MID}_\text{C}(K1) = 1;\ \ \text{AID}_\text{C}(K1) = 0;\ \text{and}\ \ \text{CC}_\text{ID}(K1) = 0.5$$
$$\text{CID}_\text{C}(K2) = 2;\ \ \text{MID}_\text{C}(K2) = 0;\ \ \text{AID}_\text{C}(K2) = 1;\ \text{and}\ \ \text{CC}_\text{ID}(K2) = 1$$

show that the component *K*1 is a candidate to be discharged due to the existence of another component, *K*2, that is completely input compatible ($\text{CC}_\text{ID}(K2) = 1$). Hence, solution-level metrics are not calculated since our candidate solution has only one component. Then, our functional suitability measurement will continue only considering *K*2 for analysis.

Table 2. ID-Compatibility measures



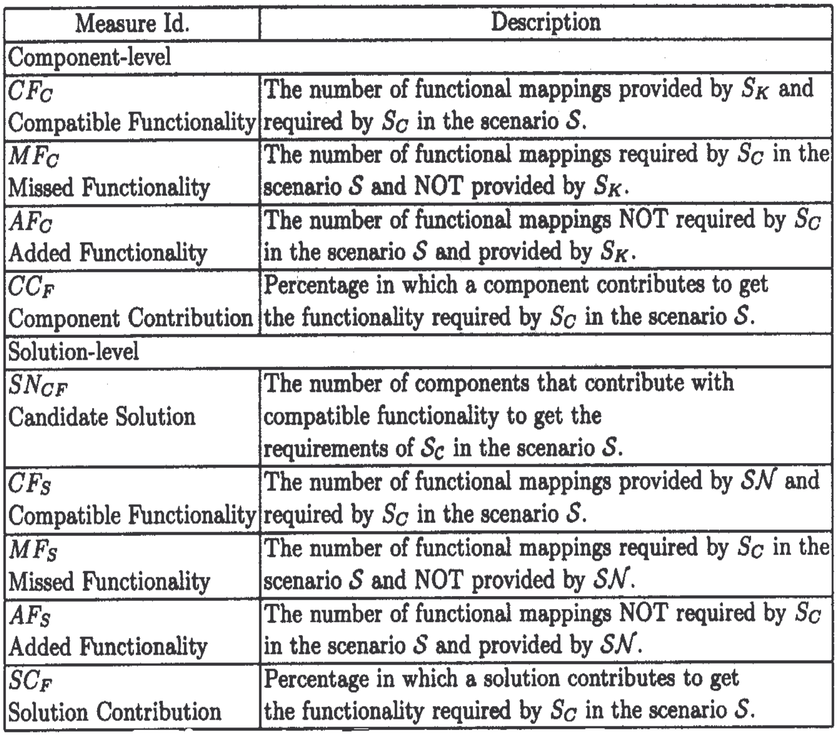
Measurement element	Measure Id.	Measure Definition
Component $\mathcal{K}$ and component $\mathcal{C}$	CID_C	$\#\{df : data \mid df \in (domS_C \cap domS_K)\}$
Component $\mathcal{K}$ and component $\mathcal{C}$	MID_C	$\#\{df : data \mid df \in (domS_C \setminus domS_K) \wedge CID_C(K,C) \geq 1\}$
Component $\mathcal{K}$ and component $\mathcal{C}$	AID_C	$\#\{df : data \mid df \in (domS_K \setminus domS_C) \wedge CID_C(K,C) \geq 1\}$
CID_C and component $\mathcal{C}$	CC_{ID}	$\frac{CID_C}{\#(domS_c)}$
Solution $\mathcal{SO}$ and component $\mathcal{C}$	SN_{ID}	$\#\{S_K \mid (CID_C(K,C) \geq 1)_{\forall K \in SO}\}$
Solution $\mathcal{SN}$ and component $\mathcal{C}$	CID_S	$\#\{d : data \mid df \in domS_C \cap \bigcup(domS_K)_{\forall K \in SN}\}$
Solution $\mathcal{SN}$ and component $\mathcal{C}$	MID_S	$\#\{d : data \mid df \in domS_C \setminus \bigcup(domS_K)_{\forall K \in SN}\}$
Solution $\mathcal{SN}$ and component $\mathcal{C}$	AID_S	$\#\{d : data \mid df \in (\bigcup domS_K)_{\forall K \in SN} \setminus domS_C\}$
CID_S and component $\mathcal{C}$	SC_{ID}	$\frac{CID_S}{\#(domS_c)}$

3.2 Functional Suitability Measures

The domain compatibility measures show that there are some candidate components able to provide some functionality. However, we cannot be certain of the amount of functionality that is actually provided. For example, the component *K*2 is full domain compatible, but some of the domain values might produce different functionalities from the required by the abstract specification of *X*, i.e. the input AID might producCOD or any other output value. Therefore, even a component might be full domain compatible, there is still another set of measures to be applied in order to determine the functional suitability. Table 3 lists our suite of functional suitability measures, which are again classified into two groups: component-level measures and solution-level measures. A more formal definition of the measures is shown in Table 4[1].

[1] Comparison between output domain values has been simplified by considering equality. A more complex treatment of output values might be similarly specified, for example, by defining a set of data flows related by set inclusion.

Table 3. Description of the Functional Suitability measures



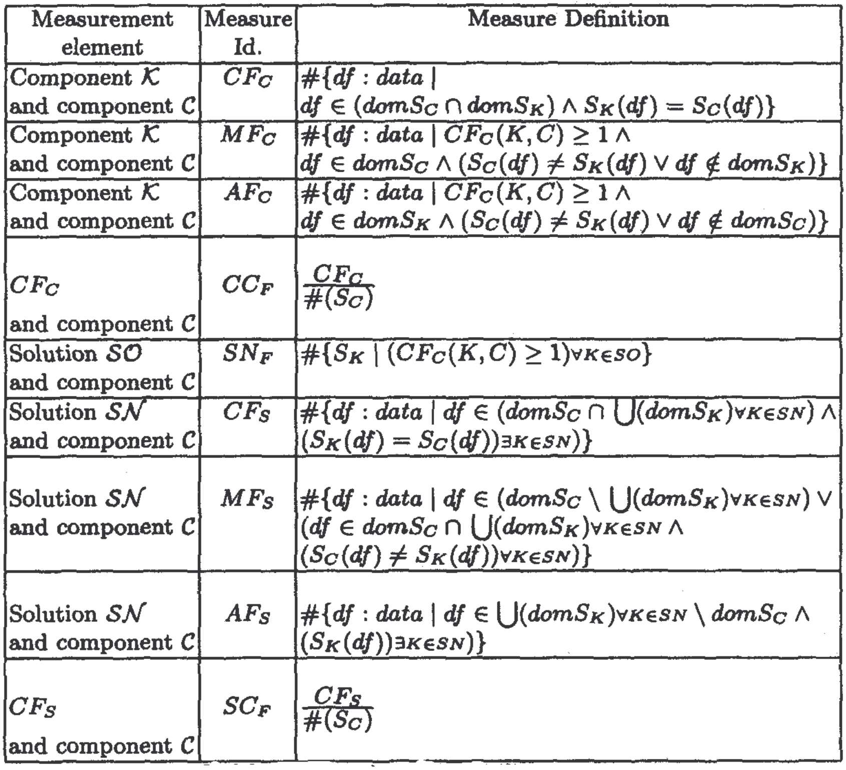
Measure Id.	Description
Component-level	
CF_C Compatible Functionality	The number of functional mappings provided by S_K and required by S_C in the scenario S.
MF_C Missed Functionality	The number of functional mappings required by S_C in the scenario S and NOT provided by S_K.
AF_C Added Functionality	The number of functional mappings NOT required by S_C in the scenario S and provided by S_K.
CC_F Component Contribution	Percentage in which a component contributes to get the functionality required by S_C in the scenario S.
Solution-level	
SN_{CF} Candidate Solution	The number of components that contribute with compatible functionality to get the requirements of S_C in the scenario S.
CF_S Compatible Functionality	The number of functional mappings provided by $\mathcal{SN}$ and required by S_C in the scenario S.
MF_S Missed Functionality	The number of functional mappings required by S_C in the scenario S and NOT provided by $\mathcal{SN}$.
AF_S Added Functionality	The number of functional mappings NOT required by S_C in the scenario S and provided by $\mathcal{SN}$.
SC_F Solution Contribution	Percentage in which a solution contributes to get the functionality required by S_C in the scenario S.

Now, let's calculate the functional suitability measures for our credit card example. The functional mapping of the abstract specification S_C is {AID → AOD; CID → COD}, and the functional mapping for *K*2 is {AID → AOD; CID → COD; *Taxes* → *Statistics*}. Then, the component-level measure results show the following values:

$CF_C(K2) = 2$; $MF_C(K2) = 0$; $AF_C(K2) = 1$; and $CC_F(K2) = 1$.

These values indicate that the component *K*2 is a candidate to be accepted for more evaluation, i.e. the component is functionally suitable but there is one added functionality 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 [5], analysis of vendor viability [3], and so forth. Our set of measures are only providing a way of identifying suitable components from a functional point of view. Measuring the other aspects is still a remaining issue. Another interesting discussion will be on analysing the representation of the input/output domain, trying to close the gap between the information provided by component vendors and the information required for evaluation, as the work in [4] remarks.

Table 4. Functional Suitability measures



Measurement element	Measure Id.	Measure Definition
Component $\mathcal{K}$ and component $\mathcal{C}$	CF_C	$\#\{df : data \mid df \in (domS_C \cap domS_K) \wedge S_K(df) = S_C(df)\}$
Component $\mathcal{K}$ and component $\mathcal{C}$	MF_C	$\#\{df : data \mid CF_C(K,C) \geq 1 \wedge df \in domS_C \wedge (S_C(df) \neq S_K(df) \vee df \notin domS_K)\}$
Component $\mathcal{K}$ and component $\mathcal{C}$	AF_C	$\#\{df : data \mid CF_C(K,C) \geq 1 \wedge df \in domS_K \wedge (S_C(df) \neq S_K(df) \vee df \notin domS_C)\}$
CF_C and component $\mathcal{C}$	CC_F	$\frac{CF_C}{\#(S_C)}$
Solution $\mathcal{SO}$ and component $\mathcal{C}$	SN_F	$\#\{S_K \mid (CF_C(K,C) \geq 1)_{\forall K \in SO}\}$
Solution $\mathcal{SN}$ and component $\mathcal{C}$	CF_S	$\#\{df : data \mid df \in (domS_C \cap \bigcup(domS_K)_{\forall K \in SN}) \wedge (S_K(df) = S_C(df))_{\exists K \in SN})\}$
Solution $\mathcal{SN}$ and component $\mathcal{C}$	MF_S	$\#\{df : data \mid df \in (domS_C \setminus \bigcup(domS_K)_{\forall K \in SN}) \vee (df \in domS_C \cap \bigcup(domS_K)_{\forall K \in SN} \wedge (S_C(df) \neq S_K(df))_{\forall K \in SN})\}$
Solution $\mathcal{SN}$ and component $\mathcal{C}$	AF_S	$\#\{df : data \mid df \in \bigcup(domS_K)_{\forall K \in SN} \setminus domS_C \wedge (S_K(df))_{\exists K \in SN})\}$
CF_S and component $\mathcal{C}$	SC_F	$\frac{CF_S}{\#(S_C)}$

Finally, our measures on functional suitability could provide a more precise indicator when calculating the maintenance equilibrium value as introduced in [1]. The number of components in the solution (SN_F) should be minimised and the contribution of functionality (SC_F) should be maximised to satisfy the CBS Functional Density Rule of Thumb: "*Maximise the amount of functionality in your system provided by COTS components but using as few COTS components as possible*" [1].

4 Conclusions and Future Work

We have presented a preliminary suite of measures for determining the functional suitability of a component-based solution. However, our measures are based on functional direct mappings, i.e. there is no semantic adaptation between the outputs provided by a component *K* and the required functionality in *X*. Therefore, we are extending the suite presented here to quantify the semantic adaptation providing an integral suite of measures.

Finally, all the measures need to be empirically validated, so much research must still be done to demonstrate the applicability of our proposal.

Acknowledgments. This work is partially supported by the CyTED (Ciencia y Tecnología para el Desarrollo) project VII-J-RITOS2, by the UNComa project 04/E048, and by the MAS project supported by the Dirección General de Investigación of the Ministerio de Ciencia y Tecnología (TIC 2003-02737-C02-02).

References

1. C. Abts. COTS-Based Systems (CBS) Functional density - A Heuristic for Better CBS Design. In *Proceedings of the First International Conference on COTS-Based Software Systems*, Springer Verlag LNCS 2255, pages 1–9, 2002.
2. R. Alexander and M. Blackburn. Component Assessment Using Specification-Based Analysis and Testing. *Technical Report SPC-98095-CMC, Software Productivity Consortium*, 1999.
3. K. Ballurio, 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. B. Bertoa, J. Troya, and A. Vallecillo. A Survey on the Quality Information Provided by Software Component Vendors. In *Proceedings of the ECOOP QAOOSE Workshop,* 2003.
5. B. Bertoa and A. Vallecillo. Quality Attributes for COTS Components. In *Proceedings of the ECOOP QAOOSE Workshop*, 2002.
6. A. Cechich and M. Piattini. Defining Stability for Component Integration Assessment. In *Proceedings of the Fifth International Conference on Enterprise Information Systems*, pages 251–256, 2003.
7. J. Dolado. A Validation of the Component-Based Method for Software Size Estimation. *IEEE Transactions on Software Engineering,* 26(10):1006–1021, 2000.
8. W. Hansen. A Generic Process and Terminology for Evaluating COTS Software –The QESTA Process. http://www.sei.cmu.edu/staff/wjh/Qesta.html.
9. L. Holmes. Evaluating COTS Using Function Fit Analysis. Q/P Management Group, INC - http://www.qpmg.com.
10. ISO International Standard ISO/IEC 9126. ISO/IEC 9126 - Information technology - Software product evaluation - Quality characteristics and guidelines for their use, 2001.
11. S. Sedigh-Ali, A. Ghafoor, and R. Paul. Software Engineering Metrics for COTS-Based Systems. *IEEE Computer Magazine*, pages 44–50, May 2001.

A Case Study in COTS Product Integration Using XML

Grace A. Lewis and Lutz Wrage

Software Engineering Institute, COTS-Based Systems Initiative
4500 Fifth Ave.
Pittsburgh, PA 15213
U.S.A.
{glewis, lwrage}@sei.cmu.edu

Abstract. The increased adoption and support for XML, as well as its clarity, extensibility, platform independence, easy validation, and support for internalization have made it an option for COTS product integration. This paper presents options for COTS product integration using XML, an XML Integration Architecture, and a real case study of how a set of COTS products were successfully integrated using the proposed XML Integration Architecture.

1 Introduction

The case study presented in this paper is currently being executed as a demonstration project within the Technology Insertion, Demonstration, and Evaluation program (TIDE) at the Software Engineering Institute [1]. The goal of this demonstration project is to insert dynamic scheduling and simulation (DSS) software into a small manufacturing enterprise. This includes the integration of several COTS products and a legacy ERP system.

In the first part of the paper we describe general issues of using XML for COTS product integration and we develop a generic architecture that can be applied to this type of integration problem. In the final section we show how the generic architecture has been applied in the DSS integration project.

2 Integration Using XML

The goal of XML since its creation has been to reduce ambiguity between applications sharing information. The software community has adopted XML in many ways: programming languages have libraries to parse and create XML documents; applications are capable of creating XML documents from information they manage; middleware products can transform information from XML to a specific format and vice versa; databases are able to store data in XML format; and e-business XML messaging standards are emerging as the solution for business-to-business integration [2].

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 41–52, 2004.

This increased adoption and support for XML, as well as its clarity, extensibility, platform independence, easy validation, and support for internationalization have made it an option for COTS product integration.

2.1 Importance of XML Syntax and Semantics Agreement

For COTS products, semantics is of increased importance because products are “black boxes”. To explain the difference between syntax and semantics consider this very simple example.

Two COTS products will share the same syntax to exchange the unit price for an item. The following XML Schema contains the type used by both products to define amounts expressed in dollars:

```
<xs:simpleType name="amount">
    <xs:restriction base="xs:decimal">
        <xs:totalDigits value="10"/>
        <xs:fractionDigits value="2"/>
        <xs:minInclusive value="0.00"/>
    </xs:restriction>
</xs:simpleType>
```

The data type `amount` specifies a maximum of ten digits where two of the ten digits are fraction digits, and a minimum allowed value of 0.00. Data of this type can therefore range between 0.00 and 99999999.99. The COTS products integrate perfectly! The problem arises when after hours of trying to figure out why the balances on both systems do not match it happens that COTS Product A exchanges the unit price before tax and COTS Product B exchanges the unit price after tax. The products share syntax but not semantics! This is not unique to XML integration. It holds true for any type of data integration. The problem is that XML appears to be more meaningful than other forms of data exchange (comma-delimited text files, Excel spreadsheets, Postscript). While an XML tag can be seen as a form of “metadata,” the truth is that an XML parser does not know about semantics. It only sees tags and syntax validation rules.

2.2 Integration Options

If the COTS products to be integrated are in a domain in which XML standards have already been developed, it only makes sense to rely on these standards instead of developing a proprietary vocabulary. If there are no standards, a custom vocabulary will need to be developed.

Non-Standard XML Integration (Proprietary)

When there are no existing standards for integration in the domain in which the COTS products are going to work, it is necessary to develop proprietary XML Schemas that describe the structure of the data to be exchanged, as well as the metadata that define semantics. On the positive side, since it is proprietary and will be used only by your organization, you have increased flexibility to create a simple format because you do not have to accommodate all possible options. Using custom-developed schemas, you will find it easier to get agreement on semantics than with standard vocabularies. On the negative side, developing a proprietary vocabulary is not an easy task. There are no rules or path to follow that will automatically lead to the correct vocabulary. There is also additional overhead involved with creating and, consequently, maintaining this vocabulary.

Standard XML Integration
If there are domains in which XML vocabularies have been defined, it makes sense to leverage this work. In fact, COTS products may come with a feature that produces data conforming to one of these standards. Several industries have created their own XML vocabularies. Examples are B2B (business-to-business) standards such as ebXML [3], RosettaNet [4], and OAGIS [5]; DocBook, a technical publications standard [6]; WML, a wireless markup language [7]; and CML, a chemistry markup language [8]. On the positive side, you do not have the additional overhead of creating and maintaining the vocabulary. Also, other COTS products may use the same standard, and that could simplify integration. This is especially true in the B2B domain. On the negative side, most of these standard vocabularies are very strong in syntax but weak in semantics. This means that even if two products say they are compliant to a certain standard, integration is not guaranteed because the two products might interpret a tag in a different way. Another problem is that the standards might not be complete. You might have to add elements or full schemas to fulfill all operations. This, of course, will cause integration problems, as well as problems with future updates to the standard. To make matters worse, most XML standards vocabularies are still evolving. This might cause frequent updates and testing to make sure the integration still works.

3 XML Integration Architecture

A high-level generic architecture for XML integration is shown in Fig. 1. The core of the architecture is the XML Integration Component. COTS products use an XML wrapper to communicate with other COTS products through the XML Integration Component.

3.1 XML Integration Component

The XML Integration Component is the core of the XML Integration Architecture. Its main function is to act as an intermediary between products. At a high-level, the XML Integration Component architecture is described in Fig. 2.
The XML Integration Component adds a level of abstraction and a level of indirection to the system. The XML Integration Component is the only component that can access the XML Schemas that will govern the data exchange between COTS products. There is obviously an exception if a COTS product can produce XML documents according to a standard. In this case, the repository is shared between the XML Integration Component and the COTS product. The advantage is that changes to the XML Schemas will only affect this component. There is also a separation of concerns since this is the only component that needs to know how to perform XML processing.

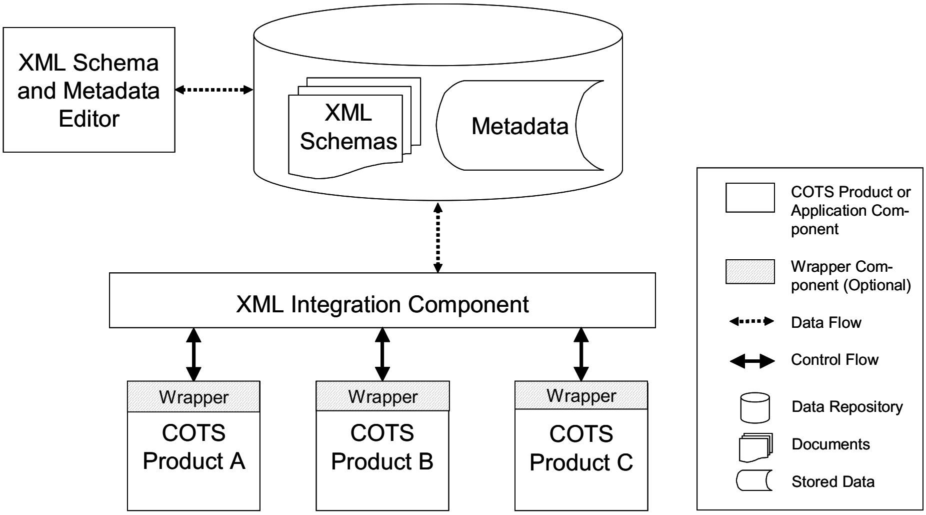


Fig. 1. High-Level generic architecture for XML integration

There is no direct communication between COTS components. Even if none of the COTS components understands XML, this level of indirection guarantees that the data being exchanged is consistent at the syntax level. If both applications agree on the semantics, they should be consistent at this level as well. On the down side, as with any architecture where there is a mediator between components, performance is likely to be affected due to the additional layer of indirection and XML processing.

3.1.1 Integration Knowledge

The Integration Knowledge sub-component can have different levels of integration knowledge. It could simply be a component that provides the following functionality:

- Creates XML documents conforming to a specific schema from data received as parameters
- Parses and validates XML documents against a specific schema
- Sends XML documents conforming to a specific schema to a certain product in the system
- Extracts data from an XML document and passes it to a product in the system

The Integration Knowledge sub-component could also provide information about the products it integrates, such as if the product is capable of generating or receiving XML documents. It can also maintain information about the way in which two components integrate. For example, every time the XML Document X is received from Product A, it has to call the following functions in Product B. This integration knowledge feature is a part of B2B XML messaging standards such as RosettaNet and ebXML.

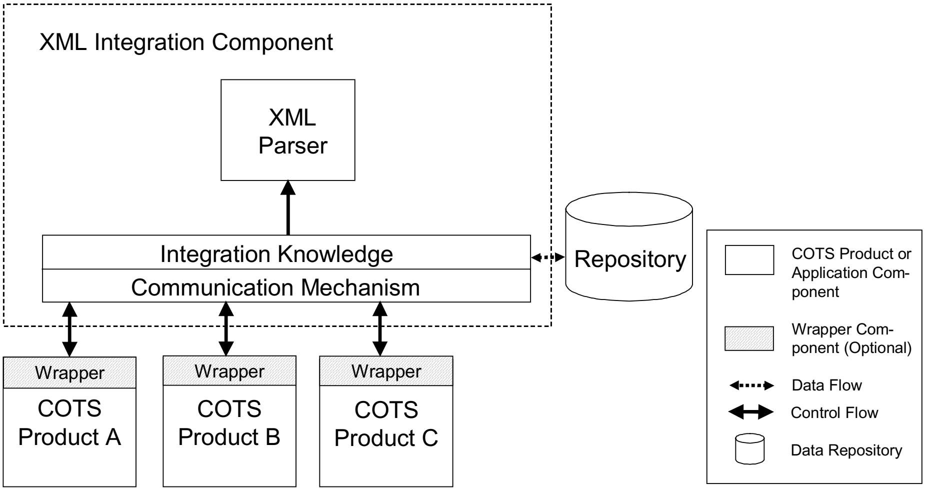


Fig. 2. High-level architecture for the XML Integration Component

3.1.2 Communication Mechanism

The Communication Mechanism sub-component determines how COTS products communicate with the XML Integration Component. There are multiple options that are implementation-dependent. For example:

- The products can access the XML Integration Component using a simple procedure call if they all reside on the same platform (in this case, the Communication Mechanism sub-component may not exist).
- If products are spread out over a LAN (Local Area Network), they can use RPC (Remote Procedure Call) or RMI (Remote Method Invocation) to communicate with the XML Integration Component.
- If the XML Integration Component is implemented as a service, it can be accessed using CORBA (Common Object Request Broker Architecture), for example.
- If products are spread out over a WAN, they can use HTTP, File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP), or message-oriented middleware such as Microsoft Message Queue (MSMQ) or IBM MQSeries.
- If products are involved in B2B transactions, they can use an XML Messaging Infrastructure such as the one provided by BizTalk or RosettaNet.

3.1.3 XML Parser

XML Parsers are available as libraries from numerous vendors. Parsing is the process of reading an XML document and reporting its content to a client application, while checking that the document is well-formed. There are two common techniques for parsing XML: those using the simple application programming interface for XML (the Simple API for XML (SAX)) and those using the document object model

(DOM). Examples of XML Parsers are Apache's Xerces Parser [9], IBM's XML Parser for Java (XML4J) [10], and Sun's Java API for XML Processing (JAXP) [11][1].

3.2 Wrapper

A wrapper in the XML Integration Architecture has two main functions:

- If the XML Integration Component is deployed as a service or is expecting a message that is more sophisticated than just a simple procedure call, the wrapper acts as a translator between the COTS component and the XML Integration Component.
- If the COTS product is not able to produce XML documents, the wrapper intercepts and/or translates calls from the component into instructions for the XML Integration Component to build an XML document conforming to a specific XML Schema stored in the repository.

In the special case that the COTS product communication mechanism matches that of the XML Integration Component and the product is capable of producing XML, the wrapper is optional.

3.3 Repository

The repository contains the XML Schemas that define the data to be exchanged between COTS products. It also contains the Metadata that provides the semantics for the data elements defined in these XML Schemas.

The XML Schemas can be used at run-time by the XML Integration Component. The Metadata is used by the integrators defining the XML Schemas and the developers working on Wrappers. As with any data repository, configuration management is necessary to assure that all products to be integrated use the same version of the XML Schemas and Metadata.

The repository can reside in a database or on any file system. Metadata can be maintained using any data management tool that can access the database or file system. The XML Schemas themselves can be edited with XML and XML Schema editors such as Altova's XML Spy [12], TIBCO's TurboXML [13], and Corel's XMetaL [14]. Some of these editors have built-in repositories or are capable of integrating with external ones.

4 The DSS Case Study

In this section we describe the software components used in the DSS project, the development of an appropriate XML document format, and how the previously presented XML integration architecture has been instantiated for this project.

[1] JAXP is not an XML parser, but an API to access any compliant XML parser.

4.1 DSS Integration Scope

The DSS project includes the integration of several COTS products. In the first major phase the project integrates a manufacturing execution system (MES) and scheduling software (SCHED). In addition it connects the legacy enterprise resource planning system (MICS) to the MES system. The second phase adds a simulation package (SIM) to the deployed capabilities.

Manufacturing Execution System. An MES controls the production of manufactured items on a shop floor. Based on production orders that typically come from an ERP system, resources and machine capacity are allocated to a production order. The MES also collects different types of real-time data from the shop floor such as machine usage and work in process, and tracks material and inventory. The MES must send data to all other systems and receive data from the simulation and from MICS.

Dynamic Scheduling. The scheduling software has been developed by the Intelligent Coordination and Logistics Laboratory of the Carnegie Mellon University (CMU) Robotics Institute [15]. The scheduler creates optimal sequences of manufacturing tasks to meet promised dates while considering limited capacity (machines and personnel) and material availability. The scheduler must send data to and receive data from MES and the simulation.

Simulation. The simulation part is the responsibility of the Manufacturing Simulation & Visualization group at National Institute of Standards and Technology (NIST) Manufacturing Engineering Laboratory [16]. The simulation package will be used to execute the schedule off-line to test the scheduling capability and to assess sensitivity of the schedule. It allows the user to simulate disruptions of the production process such as unplanned downtime of machines, sick operators, or longer task duration than anticipated. The simulation must send data to the scheduler and receive data from MICS and the scheduler.

Legacy ERP (MICS). The MICS system is currently used to manage sales and purchase orders, as well as billing and accounting. MICS must send data to and receive data from the MES.

4.2 Creating the XML Document Format

One objective of DSS, as well as of all other TIDE demonstration projects, is to contribute to a growing TIDE body of knowledge. This knowledge will form the legacy of the TIDE program and will be made available to small manufacturing enterprises in the future. As a consequence, the newly developed integration solution must provide a system architecture and other artifacts that can be applied to other projects as well.

Current real-world integration projects often rely on ad hoc or proprietary integration mechanisms because the main focus is on the COTS software products and not on the integration solution itself. There are numerous reasons for this, one of them being that it is difficult to estimate the effort that is required to perform the integration as there are plenty of unknowns at the beginning of a project. It may even happen that the cost of the integration is being negotiated before all COTS components are known in detail. So it is very easy to focus on the products and to neglect integration issues. Of course this is not a good practice, and therefore much care was given in the DSS project to get the integration right.

The project team decided that using XML data exchange for the integration would be not only a viable approach but also a good opportunity to demonstrate and document benefits and weaknesses of this type of XML usage, thus contributing to the TIDE body of knowledge.

In the early phases of the project it became quickly obvious that existing XML vocabularies were not sufficient to express the data that needs to be exchanged between the MES, SCHED, and SIM applications. The main issue was the lack of detail in XML documents that contain information about the manufacturing process. Most standard vocabularies justifiably do not address this kind of data at all because they are meant for B2B data exchange. In B2B communication the level of abstraction is usually too high to represent data as used inside a manufacturing company.

Without a pre-existing standard that can be adopted, it may be necessary to develop XML documents from scratch. Another option is to create a new standard. The creation of a new standard is usually not viable in individual integration projects. In the DSS project, however, it was possible to choose this route because the NIST team was interested in creating a shop data exchange standard for use by simulations. As a result, the XML document format for the DSS integration serves as a prototype for this new shop data standard.

The actual development of the XML format seemed to progress quickly at first, but we soon encountered some difficulties. Some of these were of a more technical nature while others – the more serious ones – were related to misunderstandings and the different viewpoints of the participating teams.

The technical problems were primarily about representation details of hierarchical structures of finished products, sub-assemblies, and sub-sub-assemblies; and of machine configurations for manufacturing tasks. Other issues were differences in the definitions of the basic concepts such as “job”, “task”, or “process plan”. While everybody thought to have a clear understanding of these concepts, it showed there were many different and incompatible views of what these concepts meant.

Another controversial issue was whether XML Schemas should be developed early on or not. We decided to use a simple ad hoc graphical representation based on UML class diagrams. This approach had the benefit that all participants – especially user representatives – were able to understand the graphics and no time was spent trying to teach everybody the intricacies of the XML Schema language.

When the XML schema was finished, we realized that it was very complex because we wanted it to be “everything for everyone”. The part of the document that was mandatory was very small compared to all the optional parts. This reduced the XML schema to one document and made the protocol very simple because only one document was exchanged between all products.

In the end the process was successful but took longer than anticipated because of the necessary discussions to clarify the semantics of proposed XML elements; once again proving that the hard part is the semantics and not the syntax.

4.3 XML Integration Architecture in Practice

The XML Integration Architecture outlined in this paper represents a generic architecture to use in the integration of COTS products using XML. In practice, it may be necessary to tailor it to address project constraints or to make tradeoffs between system qualities, as will be seen in the following case study.

Overall the DSS project follows the XML integration architecture presented earlier, but some project constraints made it necessary to restructure the components somewhat. As shown in Fig. 3, the XML Integration Knowledge component had to be split into three independent parts, one for each component. Each part was integrated with the wrapper of the MES, SCHED, and SIM components and each uses a separate XML parser. The rationale for tailoring the architecture in this way is that there is no system integrator in the DSS project and therefore no single entity to coordinate and develop the integration knowledge component. This is certainly a less than ideal situation because there are now three different XML parsers instead of one. Simple XML parsing is not problematic, but document validation using schemas is a fairly complex process with different parsers possibly exhibiting different errors. In practice, however, we experienced no problems. The wrappers and integration knowledge components for the three COTS products are developed by the MES vendor, the CMU team, and the NIST team. The integration effort is therefore reduced to defining the XML document format and defining a communication protocol to exchange these documents. This can easily be done for DSS because the patterns of interaction are relatively simple and there are only a few, fixed patterns the components must support. This would not be recommended in a project with complex interaction and patterns of communication because the knowledge of these complexities would be dispersed in three components instead of a single component, making configuration management and version control a nightmare.

The final implementation does no longer contain the repository component. This simplification is possible since XML Schemas are not necessary at run-time provided all components are well-behaved and produce only valid XML data. At development time, however, XML Schemas provide value to test if components do in fact produce valid output. Once this has been established it is no longer necessary to accept the validation overhead. This would not be true in a B2B environment where components cannot rely on other components to the same degree. There exists a general tradeoff between performance and validation. To achieve high performance it is desirable to turn off validation. But if it is not possible to trust all participating components to produce valid documents, validation is essential to discover problems before processing malformed documents. In our case, because this was a very controlled project, we were able to turn off validation and therefore had no need to keep the repository.

4.4 Legacy System Integration

The legacy MICS system is a fairly old application that has been heavily customized over the years by the customer's development team. It is written in a BASIC dialect that is not widely known.

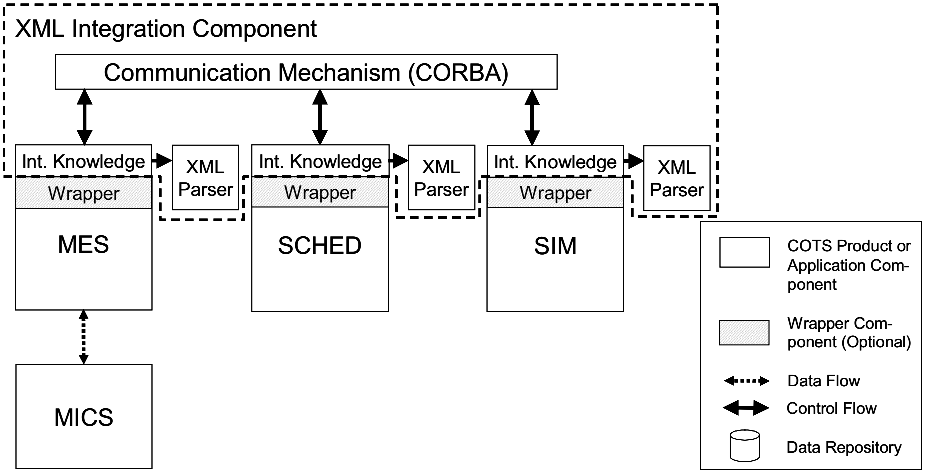


Fig. 3. DSS architecture

After careful consideration a decision was made not to use the XML Integration Architecture to connect MICS to the MES system, mostly for practical reasons:

- The customer's development team was responsible for developing the MICS integration, but the developers have no experience with current XML technology. They are, however, very familiar with file-based data exchange between systems. In addition there were few resources available in the development team to implement the interface.
- For the legacy platform and programming language, there are few libraries and programming tools for XML processing available, and these were considered too expensive.
- The MES system provides an off-the-shelf ERP system interface based on data exchange through database tables. This can be extended to support file-based exchange at a level of effort that fits the available resources.

Because it becomes more and more difficult to maintain the MICS system, the customer has a strong desire to replace the MICS system with a more modern ERP system, so the integration is seen as a temporary solution. With this background we considered the cost of extending the XML vocabulary to also include additional ERP data as too high compared to the benefit of having a more uniform solution. The devised MICS interface uses file exchange and utilizes the ERP interface provided by the MES since this was considered the less costly solution.

4.5 Lessons Learned

There are many lessons to be learned from the DSS integration project that can be applied to XML integration projects in general.

- The time to create a custom XML vocabulary should not be underestimated. XML is really only a syntax and agreeing on semantics is where the largest effort lies and unfortunately, in our experience, where the least effort is spent. In this regard XML is not different from any other data integration technology. However, using XML allows the integrator to focus on these issues instead of spending effort on defining the low-level syntax of a data exchange format and on writing and maintaining parsers for proprietary or ad hoc data formats.

- XML documents are easy to understand, also by non-technical people. This makes it easy to focus discussions on content as opposed to syntactical details. But there is also a negative aspect: XML is sometimes deceptively simple and by just looking at tag names it is easy to take semantics for granted. XML schemas have to be accompanied by a glossary that defines each tag in a clear and concise way so that there is no ambiguity for people developing or using XML documents based on this schema.
- XML schemas can be difficult to understand without a graphical representation. They cannot be used to communicate with end users, especially if the users are not technical people. In our project simple UML class diagrams proved to be an excellent medium to discuss XML document contents with technical as well as non-technical people.
- There is a tradeoff in complexity between interaction patterns and XML Schemas. You can have a very simple protocol and hide all the complexity in the documents, or you can have very simple documents and a more complex protocol that exchanges different documents depending on the type of interaction. This is achieved by introducing optional parts into the XML document. The receiving component can analyze which optional parts are filled and from that infer details about the request.
- There is no one-size-fits-all solution. Deciding what elements of the high-level generic architecture presented earlier should be combined or eliminated will depend on the system constraints.

5 Conclusions

The case study presented is a successful example of COTS product integration using XML. It proves that this type of integration is best achieved by placing an intermediary component between the COTS products. This mediator guarantees that all applications share the same data definitions and concentrates all XML processing and knowledge in one component in the system. The COTS components could well not be aware that there is "XML between them".

In the last couple of years XML has been touted by some as the solution to integration problems. Our experience with the DSS case study shows that XML based integration is not that different from any other data integration technique. It is only a syntax and does not solve any issues that are related to the meaning of data. But as a syntax XML showed to be an excellent candidate to represent data because it relieves development from the burden of defining a new syntax and creating parsers. In addition, standard XML vocabularies are available or being developed for many domains that can be leveraged to achieve data-based integration. So in the future it should be easier to find an appropriate vocabulary, and adapt it if necessary, instead of starting from scratch.

The presented integration architecture has been a viable solution for the DSS project. Whether it is also applicable to other projects and how it scales to many COTS components is mainly a function of the patterns of interaction and not so much of XML usage. XML only adds some parsing/validation overhead to each individual message, whereas the number of messages is more important for scalability. For a large number of messages it might be necessary to eliminate central components such as the repository and the integration knowledge component that would form bottlenecks that limit scalability.

XML integration has many benefits, such as increased adoption and support, clarity, extensibility, platform independence, easy validation, and support for internationalization. But, only if there is agreement on syntax and semantics it can guarantee that only correct data flows between the products being integrated.

References

[1] Technology Insertion and Demonstration Evaluation. Software Engineering Institute. *http:// www.sei.cmu.edu/tide*

[2] World Wide Web Consortium (W3C). Extensible Markup Language (XML) v1.0 Specification, Second Edition. *http://www.w3.org/TR/REC-xml*.

[3] ebXML. *http://www.ebxml.org/*

[4] RosettaNet. *http://www.rosettanet.org/*

[5] Open Applications Group. Open Applications Group Integration Specification (OAGIS). *http://www.openapplications.org/*

[6] Organization for the Advancement of Structured Information Standards (OASIS). DocBook. *http://www.oasis-open.org/docbook/*

[7] Open Mobile Alliance. Wireless Markup Language (WML). *http://www.wapforum.org/what/technical.htm*

[8] Chemistry Markup Language (CML). *http://www.xml-cml.org/*

[9] The Apache XML Project. Xerces. *http://xml.apache.org/*

[10] AlphaWorks. XML Parser for Java. *http://www.alphaworks.ibm.com/tech/xml4j*

[11] Sun Microsystems. Java API for XML Processing (JAXP). *http://java.sun.com/xml/jaxp/index.html*

[12] Altova. XMLSpy. *http://www.xmlspy.com/*

[13] TIBCO. TurboXML. *http://www.tibco.com/solutions/products/extensibility/turbo_xml.jsp*

[14] Corel. XMetaL. *http://www.corel.com/servlet/Satellite?pagename=Corel/Products/productInfo&id=1042152754863*

[15] Carnegie Mellon University. The Robotics Institute. Intelligent Coordination and Logistics Laboratory (ICLL). *http://www.ri.cmu.edu/labs/lab_5.html*

[16] National Institute of Standards and Technology (NIST). Manufacturing Engineering Laboratory. Manufacturing Simulation & Visualization Group. *http://www.mel.nist.gov/proj/simvis.htm*

COTS Product Selection for Safety-Critical Systems

Fan Ye and Tim Kelly

High Integrity Systems Engineering Group
Department of Computer Science
University of York, York YO10 5DD, UK
{fan.ye, tim.kelly}@cs.york.ac.uk

Abstract. There is an increasing interest in acquiring commercial-off-the-shelf (COTS) functionality for safety-critical applications. However, the selection of COTS products for such applications is still carried out in an ad hoc manner. This creates great difficulties for realistic cost and effort estimation, integration of the selected COTS product, and the certification of final COTS-based safety-critical systems. We believe that selection of an appropriate COTS product is the vital first step towards a successful COTS-based solution, especially for safety-critical applications. In this paper, we propose a pragmatic COTS selection approach in order to alleviate the perceived difficulties by providing a safety-informed decision on COTS selection. Reasoning from the perspective of the application context and application-specific hazards, the proposed approach defines a COTS acquisition contract from the safety requirements derived for the required COTS functionality. The terms of the COTS acquisition contract act as the evaluation and selection criteria against which any COTS candidates must be evaluated thus providing informed decisions on COTS selection for safety-critical applications.

1 Introduction

More and more software applications are being built using commercial-off-the-shelf (COTS) products. Few companies or organisations can have the luxury of trying to develop a complex system from scratch. This shift towards COTS-based solution is largely due to its perceived massive cost and time saving compared with the in-house development solution. However, as warned by many COTS researchers and practitioners, and also experienced by many COTS-based software projects, it is unrealistic to realise the goal of cost and time saving without adopting a systematic approach towards the use of COTS products [3], [4], [19].

Due to the black-box nature of COTS products, typical problems with COTS-based solution include: increased assessment cost, integration difficulties resulted from architectural mismatches [7], and continual investment as the result of forced upgrading, or vendor liquidation [14].

The idea of using COTS products in a safety-critical[1] application has encountered even bigger challenges because of the paramount requirements for safety [1]. Due to

[1] By “safety-critical” we mean that the failure of such a system would result in loss of human life/lives or/and environmental damage.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 53–62, 2004.

the nature of safety-critical systems, there are many more concerns to be addressed when using COTS products in such a system compared with the use of COTS products in a general (non-critical) context. These extra concerns include safety assurance of the final system, construction of the safety case, and final system certification. These have caused extensive difficulties for many COTS-based safety-related/safety-critical projects even for successful cases such as the DoD's Global Transportation Network (GTN) [20], and NASA's use of a commercial Global Positioning System (GPS) component for space shuttle navigation [8].

Regardless of all the difficulties, there is still a growing interest in acquiring COTS products in a safety-critical context, in order to alleviate the increasing cost and time pressures of development. However, most COTS product use is carried out in an ad hoc manner. We perceived that the single biggest problem with COTS-based solution is the lack of systematic approach in supporting the use (selection, evaluation, integration and maintenance) of COTS products in complex safety-critical applications.

We contend that the selection of an appropriate COTS product is the crucial first step towards the success of a COTS-based safety-critical project. Selection and evaluation of COTS products requires a systematic approach. To our best knowledge, there is no COTS product selection method that is dedicated to COTS-based safety-critical applications.

In this paper, we propose a novel approach called Contract-Based COTS Product Selection (CBCPS) for safety-critical systems. Reasoning from the perspective of the application context and application-specific hazards, the proposed approach defines a COTS acquisition contract from the safety requirements derived for the required COTS functionality. The terms of the COTS acquisition contract act as the evaluation and selection criteria against which any COTS candidates must be evaluated thus providing informed decisions on COTS selection for safety-critical applications.

The rest of this paper is organised as follows. Section 2 briefly summarises current methods for COTS selection and describes some specific challenges in regard to COTS selection for safety-critical applications. Section 3 presents an overview of the proposed CBCPS approach and its phases. Section 4 identifies what is beyond COTS selection. Section 5 discusses the feasibility of the proposed approach. Finally, Section 6 presents the conclusions of this work.

2 COTS Product Selection Challenges

There are a number of COTS product selection methods for use in a general application context. Examples of these methods are Procurement-Oriented Requirements Engineering (PORE) [16], Off-The-Shelf Option (OTSO) [11], Social-Technical Approach to COTS Evaluation (STACE) [12] and COTS Acquisition Process (CAP) [17].

The common shortcoming of current methods for COTS selection lies in the fact that they have put more weight on functionality and cost factors over non-functional requirements. Beus-Dukic [2] claims that the role of non-functional requirements becomes more important in regard to COTS selection. In a safety-critical context, the required safe behaviour of a component may need to be expressed in terms of both functional and non-functional properties. Such safety-related factors often play a much more important role when making decisions on "build versus buy" and on selection of a suitable COTS product.

Safety is a decisive factor when choosing a COTS product for a safety-critical application. Careless COTS selection without thorough consideration of the safety implication would ultimately result in inability to establish an acceptable safety case (i.e. project failure), or much higher cost and longer development time.

The above discussions bring us some major challenges **specific** to COTS selection for safety-critical applications. These challenges are:

- Identifying and documenting safety requirements for potential COTS functionality – Safety requirements are hard to identify and elicit. It is necessary to highlight the importance of hazard analysis in terms of eliciting and defining component safety requirements.
- Defining a set of evaluation criteria – based upon the safety requirements.
- Prioritising the defined evaluation criteria – based upon their relative importance to system safety.
- Conformance validation – the necessity of determining the relevance and strength of different types of evidence gathered from a COTS product to support the selection decision.

The CBCPS method has been developed in hopes of providing a systematic solution to the identified challenges.

3 The CBCPS Method

In this section we discuss the proposed contract-based COTS product selection method in more detail. A method overview is given first (section 3.1), and each phase of the method is then described in turn (sections 3.2-3.5).

3.1 Method Overview

The CBCPS method was developed to facilitate a systematic, repeatable and hazard-driven COTS selection process for safety-critical applications. The main **principles** of the CBCPS method are the followings:

- Careful safety analysis (component failure mode and criticality analysis) is used to define a formal acquisition contract between COTS products and the safety-critical application;
- Contract terms are used as selection and evaluation criteria;
- The required level of assurance for each contract term indicates its relative importance as a selection and evaluation criterion and determines the required type and amount of evidence needed for satisfying the contract term;
- COTS Acquisition Contract (predominately identifying safety considerations) + all general criteria (other considerations for COTS use in general context – upgrading, vendor support) to inform COTS selection.

The overall phases of the method are illustrated in Fig. 1 (major phases of the method are shown as shaded boxes). As illustrated in the figure, CBCPS is a protective approach towards COTS selection because the final COTS acquisition contract is established mainly from safety concerns of *the application context* (illustrated by the round-cornered rectangle), balanced against the considerations of the availability of COTS functionality. The selected COTS product may affect the proposed system architecture, for example, due to the request of extra risk mitigation such as wrapper (depicted by the dashed line with an arrow head).

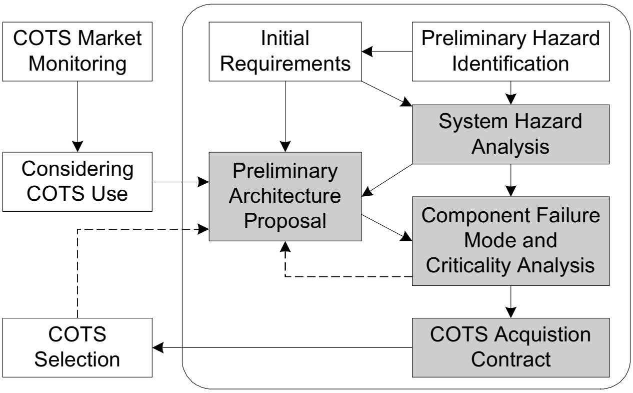


Fig. 1. Overview of the CBCPS method process

Pre-requisites for the method include:

- COTS Market Monitoring – for optimal selection
- Preliminary Hazard Identification (PHI) – the first step in the safety process. It identifies potential system level hazards through past experience, checklists or/and brainstorming, and translates them into high-level system safety design constraints (i.e. initial safety requirements) that form an important part of the initial requirements.
- Initial Requirements – a complete set of core requirements specifying the essential system functionality and quality of service (QoS) properties (e.g. performance, availability, reliability).

Detailed discussion on each major phase is presented in the subsequent sections (sections 3.2 – 3.5).

3.2 System Hazard Analysis

The initial requirements present a black box view of the system by clearly specifying the system functionality and interfaces between the system and its operational environment. System Hazard Analysis (SHA) takes such a black box view of the system and explores how system level hazards (identified by PHI) can arise as the results of system operation, interfaces and interactions between the system and its operational environment (including human operators) [13].

The major objectives of SHA are to extend and refine the high-level design constraints generated during PHI [18]. These refined high-level design constraints will greatly influence the choice of applicable architecture styles for the proposed system through placing restrictions on how the system should be decomposed into a coherent set of components, and on how different level of protections should be devised onto those components.

3.3 Preliminary Architecture Proposal

Architecture is a crucial element for any safety-critical system, because many well-known safety mechanisms (e.g. "Control + Monitor" [5], design diversity/N-version programming [10]) are rooted in the system architecture and a slight change of the architecture may have an enormous safety implication.

An architecture proposal for the intended safety-critical system is a key element of the CBCPS method. The interconnections among system components identified in the architecture proposal provide the basis for carrying out the component failure mode and criticality analysis, which in the end leads to a precise definition of the COTS acquisition contract.

The architecture proposal at this stage is influenced by a number of factors as follows (see Fig. 1):

- Safety requirements – initially defined in accordance with the system level hazards identified by PHI, and refined through SHA
- Other non-functional requirements (e.g. performance, availability)
- Functional requirements – functional decomposition that allocates a set of coherent system functionality across architectural elements (e.g. components, connectors) whilst addressing non-functional requirements.

The consideration of the use of COTS product would also affect the proposed architecture in a number of ways: firstly, it would influence the functional decomposition constrained by the availability of the required COTS functionality; secondly, safety and other non-functional requirements may request the potential COTS functionality to be isolated/contained in the proposed architecture (e.g. by wrapping).

IEEE P1471 [9] noticed that the system architecture could have multiple views in order to address the concerns of different stakeholders (e.g. user, designer, integrator, maintainer). Two architecture views are of particular interest to the CBCPS method:

- Behavioural architecture view – shows clear functional responsibility of each component including the potential COTS component (i.e. detailed functions to be carried out by a component)
- Component and Connector architecture view – shows the interfaces provided and required by a component and the interconnections among components

These two architecture views will be used as the basis for the subsequent Component Failure Mode Analysis (see section 3.4.1) in understanding how possible failure modes of individual components and interactions between components could lead to system level hazards.

3.4 Component Failure Mode and Criticality Analysis

Component Failure Mode and Criticality Analysis explores how component-level behaviours could lead to system hazards and examines the criticality of such behaviours as to system safety. Although it is a general activity of a safety process, failure mode and criticality analysis over the potential COTS component is of major interest to the CBCPS method. The results will form the basis for the final COTS acquisition contract.

3.4.1 Component Failure Mode Analysis

Component failure mode analysis examines individual component to determine how system-level hazards arise as the result of the component's behaviours and the interactions between components.

Some existing safety analysis methods such as Functional Hazard Analysis (FHA) and Software HAZOP are suitable for carrying out the component failure mode analysis. With explicitly specified behaviours of each component (including the COTS component), the behavioural view of the architecture proposal provides a perfect basis for carrying out functional deviation based analysis like FHA in order to identify hypothetical failure modes for the potential COTS component. These failure modes are hypothetical in a sense that they are based upon the major functionality a COTS component is expected to implement without any actual COTS product at hand. We can also identify how each hypothetical failure mode contributes to the system level hazards.

With the component and connector style of architecture where interfaces and interconnections between components are clearly specified, Software HAZOP can be an effective means of identifying potential hazards that may arise as the results of interactions between components, and evaluating their contributions to the system hazards. One advantage of the HAZOP approach is that it encourages consideration of both the functional (e.g. value) and non-functional (e.g. timing) failures of a component.

The resulting hazards that may be posed by the potential COTS functionality are the sources for the final COTS Acquisition Contract. It is straightforward to map an identified hazard to contract term(s). For example, in an air traffic control (ATC) system, one typical hazard is that "a pair of controlled aircraft violate minimum separation standards", and the contract term for this hazard should be "controlled aircraft must not violate minimum separation standards".

3.4.2 Component Criticality Analysis

Component criticality analysis is a way of ranking the relative importance of each identified component level hazard as to system safety. A safety assurance level (SAL) is attached to each contract term. The SAL indicates the required level of assurance for a contract term to be enforced.

Currently, four SAL ratings (1 to 4 with 4 as the highest SAL level) are defined corresponding to the four hazard severity levels (i.e. negligible, marginal, critical and catastrophic) defined in the UK Defence Standard 00-56 [15]. That is, for a COTS component failure mode, if it can directly result in a system level hazard whose severity level is catastrophic, the corresponding contract term will be allocated a SAL 4.

The SAL determines the required type and amount of evidence needed for satisfying the contract term. Contract terms with higher SAL would be more relying on direct product-based evidence such as testing and formal proof.

3.5 COTS Acquisition Contract

Contracts are widely used to precisely specify the benefit/obligation relationship between the client and the supplier. The relationship between a COTS product and a safety-critical application is analogous to the relationship between a service provider (supplier) and a service consumer (client). That is, the COTS product provides the functionality required by the application. In order to guarantee the provision of specified services, a COTS product relies on the satisfaction of certain requirements (e.g. adequate memory space, valid inputs) being provided by the application. Thereby it is natural to use a contract to capture such "rely/guarantee" relationship between the COTS product and the application.

We propose to use COTS acquisition contract as a means of specifying the detailed **safety requirements** for the potential COTS functionality derived from a safety-critical application context, and the **obligations** that the application is willing to take in order to employ the COTS functionality. This COTS acquisition contract will facilitate the selection of an appropriate COTS product for the application.

With all the analysis done in previous phases, we are now in the position of defining the detailed COTS Acquisition Contract for a specific safety-critical application. The contract presents a set of formal criteria against which any COTS candidates will be evaluated for their fitness for the required safety purpose.

Contents of such an acquisition contract include:

- The required contract terms for the COTS functionality – coming from the Component Failure Mode Analysis activity
- The SAL for each contract term – coming from the Component Criticality Analysis activity
- Suggested mitigation mechanisms that may be devised at the application context – the obligation for the application as a service consumer. This may be necessary when the evidence gathered from a potential COTS product candidate is not adequate to assure a contract term to the required SAL. Trade-off must be made between the benefits (cost and time saving) gained from the COTS-based solution and the cost-effectiveness of devising such mitigation mechanisms.

A partial acquisition contract for the selection of a commercial DataBase (DB) product in a safety-critical advisory system is extracted from a case study as follows:

Documented in a tabular form, the COTS Acquisition Contract will present the basis for a sound decision on:

- COTS products selection
- comparing two COTS products with similar functionality

Table 1. Example of a partial COTS Acquisition Contract

ID	Contract Terms	SAL	Mitigation
1	*DataBase Enquiry Service*		
1.1	DB must not return incomplete enquiry results	3	Wrapper to check the results
1.2	DB must not return incorrect enquiry results	4	Design diversity (e.g. use two different Database products)
1.3	The integrity of data item(s) must not be violated	4	Digital signature

If adequate evidence could not be obtained to provide sufficient assurance at the application level dictated by the SAL of a contract term, and there were no mitigation mechanism available or employing an available mitigation at the application context were deemed to be not cost-effective, the COTS candidate should be rejected. An initially selected COTS product may still be rejected on the basis that the safety implication of its unused functionality or some actual (in contrast to hypothetical) failure modes is too high to be mitigated.

4 Beyond Selection

Once a COTS product is selected, extra assessment has to be carried out in order to explore possible unintended interactions between the COTS product and its hosting safety-critical system. These unintended interactions may arise as the results of the COTS product's additional features or the underlying assumptions on how the product should be used. Such unintended interactions could potentially affect the final system architecture (e.g. due to the request of extra risk mitigation like wrapper). We believe that understanding the interactions (both intended and unintended) between a selected COTS component and its hosting system holds the key for the successful integration of the COTS component.

Architectural approaches such as wrapping and design diversity may be used to provide system level safety assurance where component level assurance is lacking. However, discussion of how this can be modelled using the contract approach is outside the scope of this paper.

The production of a safety case is a mandatory requirement for any safety-critical systems (including systems constructed using COTS functionality) [6], [15]. The COTS acquisition contract presents a sound justification for the selection of a COTS product. The hazard-driven COTS product selection approach provides necessary basis for the construction of a preliminary safety case. A convincing preliminary safety case at an early stage justifies the COTS-based solution and guarantees the certification of the final COTS-based safety-critical system.

Both component interaction analysis and safety case construction for safety-critical systems constructed using COTS functionality are outside the scope of this paper.

5 Feasibility of the Approach

We contend that the proposed approach is feasible and pragmatic for the selection of COTS product in a safety-critical context. Because

- firstly, the process largely synchronises with current practice on in-house safety-critical system development process. PHI, SHA, and component failure mode and criticality analysis are all essential activities in a typical safety process for the development of a safety-critical system.
- secondly, the approach reasons from the perspective of the application context and application-specific hazards. By addressing safety concerns from the very beginning, the approach avoids potential difficulties encountered by many COTS-based projects. Safety assessment demands a lot of resources and is also time consuming, therefore it is impractical to take every COTS product that implements the required functionality and perform safety assessment in order to determine its fitness for purpose. That makes the proposed application-oriented approach more practical.
- thirdly, the use of COTS acquisition contract provides a sound basis for the selection of a COTS product. It also justifies the use of COTS products in safety-critical applications by supporting the construction of preliminary and final safety cases.

The effectiveness of the proposed CBCPS approach will be evaluated in further case studies.

6 Conclusion

In this paper, we proposed a systematic approach towards COTS selection for safety-critical applications. Due to the nature of safety-critical systems, we believe that it is more pragmatic if we could reason from the perspective of the application context and application-specific hazards while considering acquiring functionality commercially. The approach defines a COTS acquisition contract from the safety requirements derived for the potential COTS functionality. The terms of the COTS acquisition contract act as the evaluation and selection criteria to provide safety-informed decisions on COTS selection for safety-critical applications. By taking into account safety concerns early at the selection stage, the proposed approach can potentially help with the subsequent integration of the selected COTS product and certification of the final system. We believe that, if the COTS acquisition process is properly managed, the “better, cheaper, faster” proposition may be achieved for the use of COTS software products in safety-critical applications.

References

1. Anderton B., Armstrong J., Frankis D., Saddleton D., Taylor J., and Thombs D., "Can You Afford COTS Software in Safety Critical Applications?," *in Proceedings of the 2nd European COTS User Group (ECUA) Workshop*, Orsay, Paris, France, 2001.
2. Beus-Dukic L., "Non-Functional Requirements for COTS Software Components," *in Proceedings of ICSE workshop on COTS Software*, Limerick, Ireland, 2000, ACM.
3. Boehm B. and Abts C., "COTS Integration: Plug and Prey?," *IEEE Computer*, pp. 135-138, January 1999.
4. Brownsword L., Carney D., and Oberndorf T., "The Opportunities and Complexities of Applying Commercial-Off-The-Shelf Components," *Crosstalk*, April 1998.
5. Douglass B. P., *Doing hard time: developing real-time systems with UML, objects, frameworks, and patterns*: Addison Wesley, 1999.
6. DSTO, "DEF(Aust) 5679 - The Procurement of Computer-Based Safety-Critical Systems," Defence Science and Technology Organisation, Australia, Australian Defence Standard, August 1998.
7. Garlan D., Allen R., and Ockerbloom J., "Architectural Mismatch or Why it's hard to build systems out of existing parts," *in Proceedings of 17th International Conference on Software Engineering (ICSE'95)*, Seattle, WA, USA, 1995.
8. Goodman J. L., "The Space Shuttle and GPS - A Safety-Critical Navigation Upgrade," *in Proceedings of 2nd International Conference on COTS-Based Software Systems*, Ottawa, Canada, 2003, Springer.
9. IEEE, "P1471 - Recommended Practice for Architectural Description of Software-Intensive Systems," IEEE Computer Society, Standard, September 2000.
10. Knight J. C. and Leveson N. G., "An Experimental Evaluation of the Assumption of Independence in Multiversion Programming," *IEEE Transactions on Software Engineering*, vol. 12 1986.
11. Kontio J., "A Case Study in Applying a Systematic Method for COTS Selection," *in Proceedings of 18th International Conference on Software Engineering (ICSE)*, Technische Universität, Berlin, Germany, 1996, IEEE Computer Society.
12. Kunda D. and Brooks L., "Applying Social-Technical Approach for COTS Selection," *in Proceedings of 4th UKAIS Conference*, University of York, York, UK, 1999, McGraw Hill.
13. Leveson N. G., *Safeware: System Safety and Computers*: Addison-Wesley, 1995.
14. Meyer B. and Oberndorf P., *Managing Software Acquisition: Open Systems and COTS Products*: Addison-Wesley, 2001.
15. MoD, "00-56 Safety Management Requirements for Defence Systems," Ministry of Defence, Defence Standard, December 1996.
16. Ncube C. and Maiden N., "Selecting the Right COTS software: Why requirements are Important," in *Component-Based Software Engineering: Putting the Pieces Together*, G. T. Heineman and W. T. Councill, Eds.: Addison-Wesley, 2001.
17. Ochs M., Pfahl D., Chrobok-Diening G., and Nothhelfer-Kolb B., "A COTS Acquisition Process: Definition and Application Experience," *in Proceedings of 11th ESCOM Conference*, Shaker, Maastricht, 2000.
18. Storey N., *Safety-Critical Computer Systems*: Addison-Wesley, 1996.
19. Voas J., "COTS Software: The Economical Choice?," *IEEE Software*, vol. 15, pp. 16-19, March 1998.
20. Wallnau K., Carney D., and Pollak B., "COTS Software Evaluation," *SEI Interactive*, June 1998.

Driving Component Selection through Actor-Oriented Models and Use Cases*

Vijay Sai[1], Xavier Franch[2], and Neil Maiden[3]

[1]Software Engineering Institute Carnegie Mellon University
4500 Fifth Avenue Pittsburgh, PA 15213
vsai@sei.cmu.edu
[2]Universitat Politècnica de Catalunya (UPC)
Jordi Girona 1-3, CN-C6 08034 Barcelona
franch@lsi.upc.es
[3]Centre for HCI Design City University
Northampton Square, London EC1V OHB, UK
n.a.m.maiden@city.ac.uk

Abstract. This paper reports results from a retrospective application of the REACT approach to COTS selection for a real-world financial planning, forecasting and budgeting application. We derived actor-oriented models from existing use cases to represent the architecture of the system. We then investigated which combinations of components could be plugged into the system architecture as instances of model actors, and applied some assessing methods and techniques to evaluate the resulting architectures and the behaviour of the components. From this case study exercise, REACT demonstrated its exploitability for real-world COTS and component selection exercises.

1 Introduction

In previous research [1] we reported the innovative application of the *i** goal modeling approach to inform the selection of software components. Most reported component selection processes [2, 3] and techniques [4, 5] support the selection of independent software components. In contrast, in real-world applications, the decision to select one component is rarely so simple and depends on which others will be selected [6]. The innovation, implemented in REACT (REquirements-ArChiTecture), was to apply *i** SD models to model software architectures in terms of actor dependencies to achieve goals, satisfy soft goals, consume resources and undertake tasks.

However, the REACT approach has yet to be evaluated on a real-world case study in order to investigate its utility and usability. Rather than apply REACT to a live project with attendant risks, we have chosen to apply it retrospectively to a previous but complex component selection exercise reported in [7]. The selection referred to involves the implementation of a solution comprised of several different COTS and custom components in the suite of financial planning, forecasting and budgeting

* This work has been partially supported by the Spanish project TIC2001-2165.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 63–73, 2004.

packaged application presently nearing production rollout at the Software Engineering Institute (SEI). We believe that this system serves the purpose of validating the *i** models. It had a complex architectural environment, with important interdependencies between major components of the system architecture. These interdependencies gave rise to integration challenges that needed to be overcome in order to satisfy the system's goals. Such a case study was expected to give REACT a thorough road-testing, and point out the need for specific improvements and extensions.

In this paper we extend and apply REACT to the SEI case study in order to explore 3 research questions:

1. Can REACT be applied successfully to a real-world COTS selection problem?
2. What are the potential benefits that REACT can offer to decision-makers?
3. What new problems must REACT overcome to be useful in decision-making?

2 The REACT Method

REACT supports a concurrent requirement modeling and architecture modeling process with clear synchronization stages in which we test candidate architectures for requirements compliance, similar to previous iterative requirements acquisition-component selection processes developed by the authors [6, 8]. It consists of some activities that include development of quality models for components' domains, requirements elicitation, decision-making techniques, and others. For the purposes of this paper, we skip some activities and focus on the formulation of requirements models and their use for assessing architecture comparison and component selection.

REACT is an advance on existing methods described in [6, 8] that support the selection of software components that satisfy stakeholder and system requirements. These methods support acquisition of requirements to discriminate between components and testing component compliance against these requirements. However, they were limited to supporting the selection of one component at a time, and did not support more common selection problems in which different combinations of interdependent candidate components have to be explored during selection.

In REACT, system requirements are used to produce a *socio-technical system*. To do this we apply Yu's *i** approach [9] to model complex systems and actor goals, and in particular its Strategic Dependency (SD) model for modeling networks of dependency relationships among actors. Each dependency can be a *goal*-type, *soft goal*-type, *task*-type or *resource*-type dependency. Opportunities available to actors are explored by matching a *depender*, the actor that "wants", and a *dependee* which has the "ability" to give what the depender wants. Since the *dependee's* abilities can match the *depender's* requests, a dependency model of a system can be developed.

Next we use candidate components to make first-cut decisions about the system architecture. System actors are instantiated by (types of) components, providing other *i** SD models, the *system candidate architectures*. Our innovation here is to model a system architecture in terms of actors and roles to be fulfilled by components linked by dependencies in the requirements domain rather than software connectors [10]. Instantiation of actors by components is many-to-many. Depending on the concrete form of this instantiation, dependencies among actors may be hidden inside components or not. Different architectures with different instantiations will differ in some architectural properties, such as diversity, data integrity and others. We have defined some of these properties by studying the dependencies in both *i** SD models [1].

As an example, fig. 1 shows at the left an excerpt of a socio-technical system for a meeting scheduler with three software actors (meeting scheduler itself, and a mail client and mail server to enable mailing facilities) and a human actor, with some dependencies among them. At the right, fig. 1 shows a candidate architecture that instantiates two actors by the same component, hiding some of the dependencies.

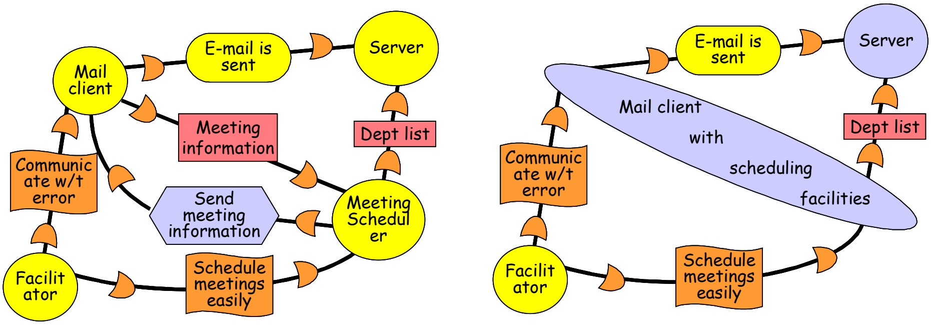


Fig. 1. A socio-technical system for a meeting scheduler (left) and a candidate architecture (right).

3 A Case Study

This section briefly presents the fundamentals of the case study that forms the basis of this paper. The Software Engineering Institute at CMU, USA, recently acquired a system to satisfy financial management and decision-making process requirements. The need arose in the wake of the existing system being rendered technically nonfeasible to maintain. As stated earlier [7], because of the systems integration challenge with complex source and target data structures and database management system technologies, careful selection of a new suite of components was a task probably as complex as the design and implementation phases of the project. This selection task was also influenced by the enterprise architecture. Hence, it was decided that it might be an apt case study in light of the purpose this paper intends to serve.

Fig. 2 shows a model for the New SEI socio-technical system. It includes 3 human actors and others representing systems related with the new one, such as the core financial application (CFA) and the human resources information system (HRIS). It also shows important dependencies among these actors in terms of user-specified goals and soft goals and technical architecture goals. In this paper we will focus on important non-functional requirements, or soft goals, for the new system: security, maintainability and reliability (in terms of availability and accuracy of calculations). The case study process was in several stages - product identification, evaluation and selection based on training with the REACT method, then recommending a solution understood to be one that would satisfy the identified goals.

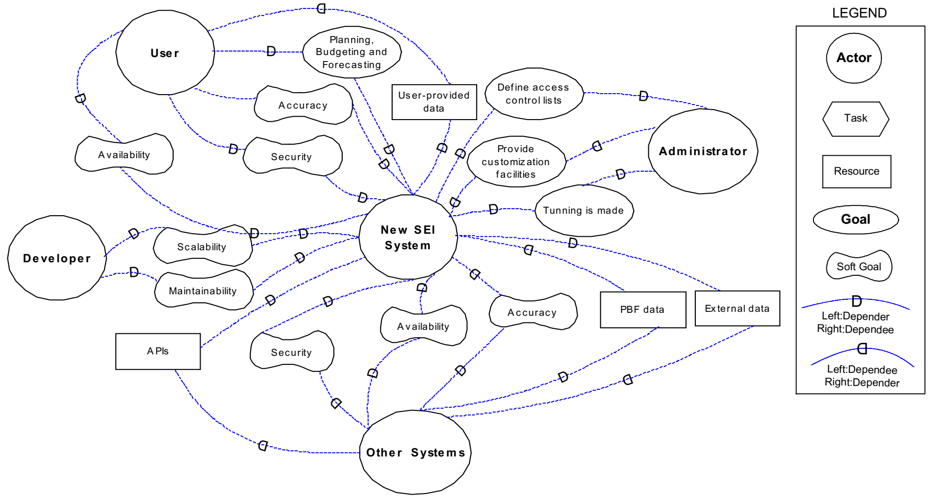


Fig. 2. A high-level socio-technical system for the new SEI system.

4 Deriving an Actor-Based Model from Use Cases

Use cases were used in the SEI project for determining the functional scope of the system. For REACT to be useful in industrial projects, we planned to take these use cases as starting point for deriving actor-based models. We know that use case and scenario walkthroughs are well-established techniques for discovering new system actors and their goals [11]. Use cases and scenarios are also versatile - for example different granularities of use case can be used to discover business goals and software system requirements [12]. However, most published approaches do not consider the representation of the new goals or requirements, or the target requirements model.

Several researchers have explored the relationship between *i** modeling and use case modeling. For example in [13] is described a method to derive use case models from organizational models represented using *i**. In contrast, our approach - which also uses *i** to model system architectures - aims to generate *i** models from use case models and descriptions that are simpler and quicker to start a requirements analysis with. It implements heuristics from RESCUE, a scenario-driven requirements process that supports concurrent use case and *i** modeling [14], to discover *i** model components from use cases. Heuristics applied to a use case model support the generation of the SD model, e.g. each primary actor on the use case model is an *i** model actor that has a direct dependency relationship with the new system actor. RESCUE also provides heuristics for deriving SR models, but they are out of the scope of the paper.

Use Case "Data Extraction"	
Step	Action description
1	The *Scheduler* triggers the *New SEI System* at predefined intervals (nightly).
2	The *New SEI System* derives the username and password.
3	The *New SEI System* asks for connection to the *HRIS*, the *CFA* and the *Current SEI System* using the username and password.
4	The *HRIS,* the *CFA* and the *Current SEI System* acknowledge the connection.
5	The *New SEI System* imports data from the *HRIS*, the *CFA* and the *Current SEI System* through the input ports into the corresponding relational database staging tables.
6	The *New SEI System* lists the data received and sorts it in a predefined manner.
7	The *New SEI System* stores the data; records that fail to be loaded are stored into a log file.
8	The *New SEI System* shuts down.

Fig. 3. Use case for data extraction in the new SEI system.

We show in fig. 3 a use case that specifies the functionality of importing data from other systems, i.e. the process of extraction, transformation and loading of data between source and target systems to satisfy the process requirements of the system. The use case representing the automatic scheduling of some calculations at predefined intervals is not described but is reflected in the ensuing *i** SD model.

Fig. 4 presents a SD model that is derived from this use case by applying our heuristics. Note that there is a single actor that represents the new SEI system, related to the other five actors by means of several dependencies. The final SD model (not included for space reasons) would contain all the dependencies coming from the use cases of the system.

5 Building a Requirements Architecture of the System

The next step of REACT builds the SD detailed model so that it can be analyzed from the architectural and component points of view.
We build this model by applying the steps:

- Build an *i** SR model for the new system. The main goal of this actor is expressed in terms of tasks and resources, which appear as leafs in the SR model
- Refine the existing SD model (see fig. 4) by decomposing the new SEI system actor into other actors that play a well-defined role in the system.
- Assign the SR elements to these new actors.
- Assign the dependencies that appear in the original SD model to the appropriate actors in the new model and identify dependencies among new actors.

We apply this process to the piece of SD model shown in fig. 4. For the sake of brevity, we do not develop the SR model; we present directly the result in fig. 5, including also some other elements coming from another use case (namely burdening, i.e. allocation of overhead to the raw costs spent on a unit of work within the scope of a project) to make the resulting diagram more complete. We obtain 3 actors:

- The planning, budgeting and forecasting actor (PBF), doing most of the financial job and managing the new data base.
- The application integrator (AI), assists in data exchange between source and target systems.
- The business rules actor (BR), supports design and execution of business rules.

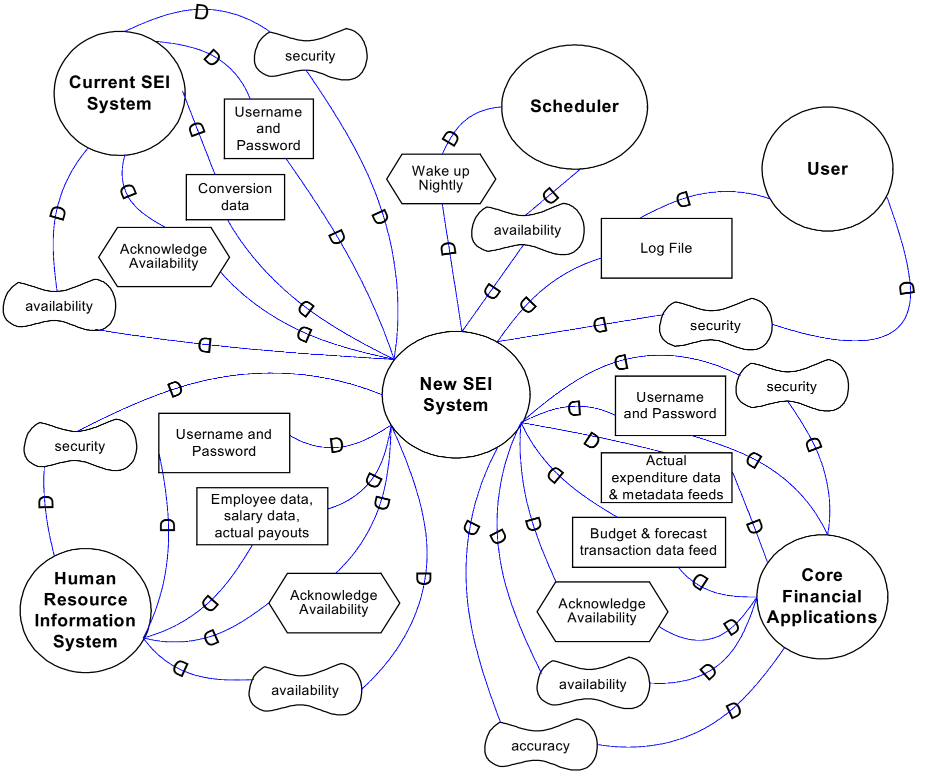


Fig. 4. SD diagram for data extraction in the new SEI system, new system not decomposed.

6 Analysis of the Model

At this stage, when applying this process to the whole SD diagram (which puts together information from all existing use cases), we would obtain a complete requirements-oriented architecture of the system. The REACT methodology proposes to carry out an analysis of the system for informing component selection.

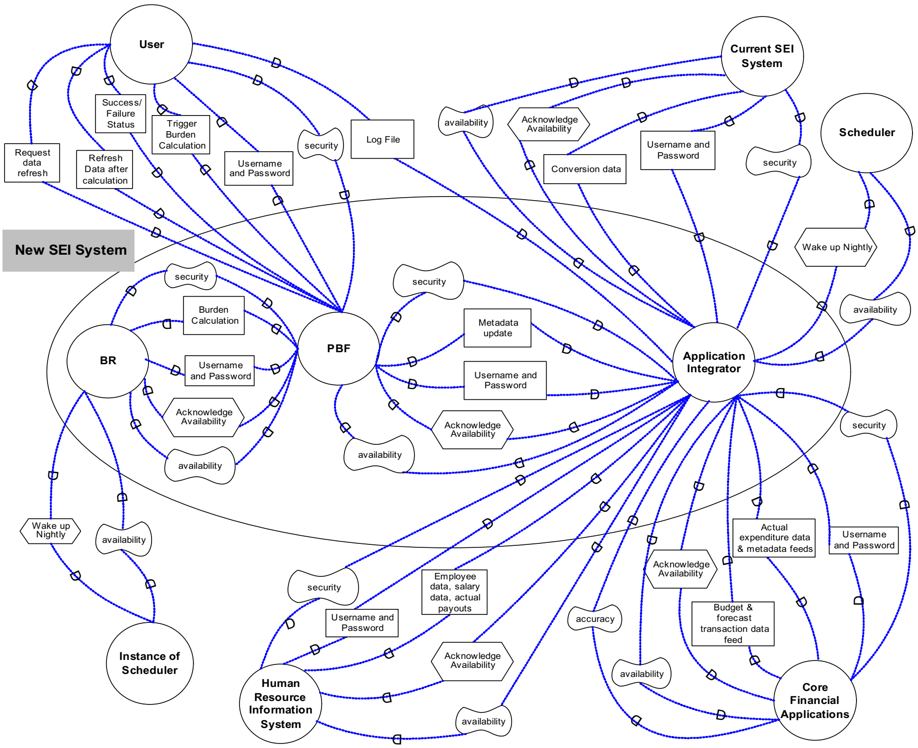


Fig. 5. SD diagram for data extraction and burdening in the new SEI system.

Table 1. Three candidate architectures for the new SEI system.

Actor	Candidate architecture		
	A1	A2	A3
PR	COTS1	COTS2	COTS3
RM	COTS1	COTS2	COTS3
PBF	COTS1	COTS2	COTS3
RD	COTS1 + add-ins	COTS2 + add-ins	COTS3
BR	COTS1	CUSTOM1	COTS3
AI	COTS1	CUSTOM2	CUSTOM3

In addition to the 3 actors already presented, we obtained other 3: a report designer actor (RD), a report manager (RM) and a planning and reporting actor (PR). During system acquisition, the SEI examined which financial products were offered in the COTS market for those goals. After some screening, three alternatives were chosen as finalists, each of them determining a different candidate architecture for the system. This kind of overlapping among determining the candidate architectures and selecting the candidate products is an important difference to our approach in [1], where architecture definition was previous to market screening. The instantiation of actors of each candidate architecture is shown in table 1. We may observe that the 3 architectures heavily rely on a central COTS financial component. Architectures A1 and A2 demand some glue code for the RD component, while A2 and A3 require some in-house component.

Now, relevant architectural properties shall be chosen for driving the analysis. As an example, we use 3 that played an important role in the case study (although others also influenced the selection): maintainability, data security and diversity. At this point, REACT proposes two different strategies for carrying out architectural analysis. First, MCDM techniques may be used to obtain a first-cut criteria. Then, some architectural metrics can be defined and applied to obtain more informative values at the price of investing time.

MCDM techniques are a feasible alternative for comparing architectures, especially when there are few. We apply AHP [15] considering a different hierarchy for each architectural property. The level of detail of the analysis depends on how much effort can be invested. In our case, we simplify matters by comparing pair-wise the candidate architectures with respect to the 3 properties directly instead of building a hierarchy of criteria for each property. Results are summarized in table 2 using AHP scores. Maintenance of A1 is considered much better than others due to the inexistence of custom code. Data security of A1 is considered also much better than others because there is less information flow among different components. But A1 is less diverse than the others since there is a critical component whose failure would compromise the entire system. A2 and A3 are similar applying the same rationale.

In [1] we proposed metrics for data security and diversity, which are defined in terms of SD model dependencies. For maintainability, we may define as metric:

$$\text{Sum}_{c\in \text{Components(SD model)}}\ (\text{weightingFactor}_{\text{Maintainability}}(\text{IncDep(c)))*correctionFactor}_{\text{Maintainability}}(\text{c}))$$

This formula shows the main concepts behind our proposal, some of them not appearing in [1] (in fact, metrics defined in [1] should be adapted to the innovations).

Table 2. AHP matrixes for the 3 candidate architectures with respect to 3 architectural properties (left, maintainability; middle, data security; right, diversity).

	A1	A2	A3
A1	1	7	6
A2	1/7	1	1/3
A3	1/6	3	1

	A1	A2	A3
A1	1	6	6
A2	1/6	1	2
A3	1/6	1/2	1

	A1	A2	A3
A1	1	1/5	1/4
A2	5	1	3
A3	4	1/3	1

IncDep(c) denotes the number of instance dependencies (i.e., dependencies that appear once every actor has been instantiated by components) that are not hidden. A hidden dependency is a dependency that connects two actors that are assigned to the same component (remember fig. 1 that illustrates this concept). Also, the type of component plays a crucial role in the behaviour of the architecture; for instance, the maintainability of a COTS component is completely different of that of a custom component. The metrics includes this concept by means of the correction factor of the component, that is a multiplicative factor that depends on the type of component. Last, we allow to weight the relevance of dependencies in the definition of the metrics. For instance, a resource such as *conversion data* (see fig. 5) can be considered more important than *username and password* from the maintainability point of view and then a greater weight assigned. Of course, both the correction and weighting factors may be left the same (i.e., equal to 1) for each type of component and dependency if a more lightweight analysis is to be carried out.

The detailed analysis using these metrics is very similar to the one presented in [1]. In fact, the metrics confirm the results obtained with the AHP analysis, and thus we may conclude that no architecture is the best selection with respect to all the emergent properties. We need to complement therefore the architectural-level analysis with the component-level analysis, as recommended in REACT.

Component analysis is done on top of the tasks and resources that appear as leafs in the SR model of the system. In [7] it was reported that in the real case study a grading scale from 0 to 10 was used to rank the components. In REACT, we do not apply the scale to components as a whole but to the actor's part of the components.

Results are summarized in table 3, with focus on the diversity property that was deficient in A1[1]. We observe that the component-level analysis makes A1 better than its competitors. From the point of view of our particular research interests this is reassuring since in fact A1 was the architecture finally chosen by the SEI.

Table 3. Actor based *diversity* ratings of the evaluated candidate architectures.

Actor	Candidate architecture		
	A1	A2	A3
PR	10.00	9.00	9.75
RM	9.75	9.75	9.67
PBF	10.00	9.00	9.75
RD	9.75	9.75	9.67
BR	9.75	9.00	9.25
AI	10.00	9.00	9.50

7 Conclusions and Future Work

This paper reports results from a retrospective application of REACT to aid the selection of components for a real-world financial planning, forecasting and budgeting application. The application was a success. The team was able to apply the REACT process, develop its models and undertake its analyses. *i** modeling was effective for modeling the architecture of both the socio-technical and software systems. REACT heuristics were applied to explore the architecture- and component-level properties of the candidate solutions. As such, the REACT method presented in this paper provides greater component selection capabilities that methods such as SCARLET [8]. Whilst SCARLET supports requirements acquisition and compliance evaluation with one component, REACT recognizes that most selection problems involve multiple com-

[1] Since this is a retrospective analysis, the numbers are mostly based on an inference summing evaluation criteria and the evaluators' grading and comments.

ponents with shared features and complex interdependencies. This paper demonstrates the new modeling capabilities needed to support multiple-component selection that will be integrated with established requirements acquisition and component evaluation techniques in future versions of REACT.

The results allow us to answer the 3 research questions reported at the beginning of the paper. The first question was can REACT be applied successfully to a real-world component selection problem. The answer is a qualified yes. REACT was applied, albeit retrospectively. REACT has the potential to aid selection between components in a medium-sized application, although scalability to large applications remains to be shown. Furthermore, several specific lessons were learned:

- Use cases and the RESCUE heuristics [8] were an effective starting point for *i** modeling, thus enabling REACT to be integrated with mainstream development methods such as the RUP, although more work needs to be done;
- Overlapping the application of REACT's architecture-level and component-level analyses can be beneficial, in particular to discover and assess consequences for the architecture from the selection of different permutations of components.

The second question was what are the potential benefits that REACT can offer to decision-makers. One observed benefit is that REACT encourages and supports wider stakeholder involvement during selection by allowing them to define and reason with requirements. Likewise, the models allowed architects, developers and integrators to communicate and develop a shared understanding of the new application. REACT also provides different but complementary methods that model the systems architecture from different points of view. Different architectural properties may be ranked differently using these viewpoints, as it happened with diversity.

SEI was the first user of the REACT method. Results from this case study are positive, as SEI envisages that REACT can be used for both future SEI internal projects and, with extensions, at SEI's client sites if appropriate. The results hold out the promise of a capability for deeper evaluation of COTS system architectures through *i**modeling and pair-wise comparison and analysis, which appear to foster early capture of constraints and while lending visibility of requirement satisfiers.

The third question was what new problems must REACT overcome to be useful and usable in decision-making. We extended REACT beyond the version reported in [1] in four distinct ways that demonstrate how we are already overcoming problems that were encountered: a better defined application process; integration of use case modeling; more accurate definition of metrics; and integration of local, cost-effective use of multi-criteria decision-making techniques such as the AHP [15] to support first-cut architecture selection decisions prior to more rigorous analyses. The approach is made cost-effective by applying the AHP to undertake pair-wise comparisons of the subset of compliant architectures with a subset of the requirements to provide a ranking of the architectures. As such an analyst is required to make dozens of pair-wise comparison decisions rather than hundreds or thousands. This focused use of AHP is critical for its efficient use in component selection.

References

1. X. Franch, N.A.M. Maiden. "Modelling Component Dependencies to Inform Their Selection". In *Procs. 2nd International Conference on COTS-Based Software Systems* (ICCBSS), Ottawa (Canada), LNCS 2580, 2003.
2. S. Comella-Dorda, J.C. Dean, E. Morris, P. Oberndorf. "A Process for COTS Software Product Evaluation". In *Procs. 1st International Conference on COTS-Based Software Systems* (ICCBSS), Orlando (Florida, USA), LNCS 2255, 2002.
3. N.A.M. Maiden, C. Ncube. "Acquiring Requirements for COTS Selection". *IEEE Software* 15(2), 1998, pp. 46-56.
4. J. Kontio. "A Case Study in Applying a Systematic Method for the COTS Selection". In *Proceedings 18th IEEE International Conference on Software Engineering* (ICSE), 1996.
5. C. Ncube, J.C. Dean. "The Limitations of Current Decision-Making Techniques in the Procurement of COTS Software Components". In *Procs. 1st International Conference on COTS-Based Software Systems* (ICCBSS), Orlando (Florida, USA), LNCS 2255, 2002.
6. X. Burgués, C. Estay, X. Franch, J.A. Pastor, C. Quer. "Combined Selection of COTS Components". In *Procs. 1st International Conference on COTS-Based Software Systems* (ICCBSS), Orlando (Florida, USA), LNCS 2255, 2002.
7. V. Sai. "COTS Acquisition Evaluation Process: The Preacher's Practice". In *Procs. 2nd International Conference on COTS-Based Software Systems* (ICCBSS), Ottawa (Canada), LNCS 2580, 2003.
8. N.A.M. Maiden, H. Kim, C. Ncube. "Rethinking Process Guidance for Software Component Selection". In *Procs. 1st International Conference on COTS-Based Software Systems* (ICCBSS), Orlando (Florida, USA), LNCS 2255, 2002.
9. E. Yu. "Modeling Organizations for Information Systems Requirements Engineering". In *Procs. 1st IEEE International Symposium on Requirements Engineering* (ISRE), 1993.
10. M. Shaw. "Heterogeneous Design Idioms for Software Architecture". In *Procs. 6th IEEE International Workshop on Software Specification and Design* (IWSSD), 1991.
11. A. Mavin, N.A.M. Maiden. "Determining Socio-Technical Systems Requirements: Experiences with Generating and Walking through Scenarios". In *Procs. 11th International Conference on Requirements Engineering* (RE), 2003.
12. C. Rolland, C. Souveyet, C.B. Achour. "Guiding Goal Modeling Using Scenarios". *Transactions of Software Engineering* (TSE), 24(12), 1998.
13. V. Santander, J. Castro. "Deriving Use Cases from Organizational Modeling". In *Procs. 10th International Conference on Requirements Engineering* (RE), 2002.
14. N. Maiden, V. Croce, H. Kim, G. Sajeva, S. Topuzidou. "SCARLET: Integrated Process and Tool Support for Selecting Software Components". Book chapter in *Component-Based Software Quality*, LNCS 2693, 2003.
15. T.L. Saaty. *The Analytic Hierarchy Process*. University of Pittsburgh, 1998.

Managed Technology Adoption Risk: A Way to Realize Better Return from COTS Investments

Suzanne Garcia, John Robert, and Len Estrin

Software Engineering Institute,
4500 Fifth Ave, Pittsburgh, PA 15213
{smg,jer,le}@sei.cmu.edu

Abstract. Two companies can install the same COTS (Commercial Off The Shelf) software package, yet one company enjoys more success, and a better return, than the other. Needless to say, many factors could be involved in this common scenario. Yet, chances are, one of the factors is that the successful company actively managed the non-technical aspects of the adoption of the COTS software adoption, instead of just selecting it and installing it. What is the difference? According to technology transition researchers, "installed" means that the system is operational; however, only a few people use the software as intended (or at all!) [Fichman1995]. "Adopted" means that the system is operational, and employees are using it in the way that was intended to support the business need that led to the COTS adoption to begin with. Every organization exhibits different risks for adopting a particular technology, and whether and how those risks are managed often determine whether adoption is achieved, vs merely achieving installation. Managing adoption, especially managing adoption risk, actually starts before acquiring any technology and continues after installation.

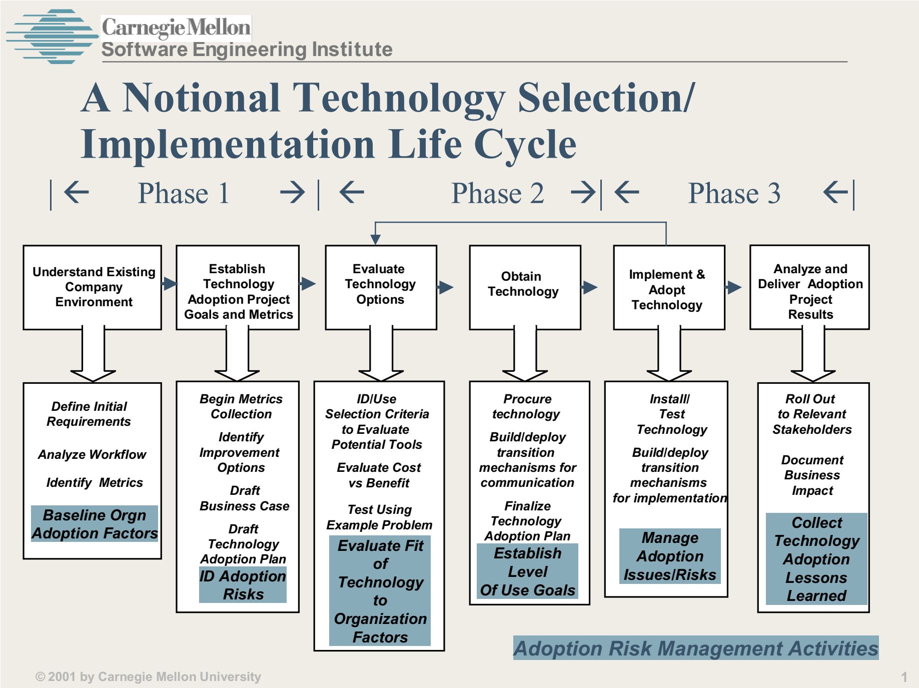
1 Introduction

This paper provides insight into the Software Engineering Institute's (SEI) approach for actively managing adoption risks as part of overall software technology selection and implementation. This life cycle, customized from successful SEI practices for commercial software adoption for larger companies, was used to guide TIDE (Technology Insertion, Demonstration, and Evaluation) projects, which were targeted at small manufacturing companies adopting advanced COTS engineering software solutions, and is the basis for two publicly-presented TIDE technology adoption tutorials available on the TIDE web site (http://www.sei.cmu.edu/tide), "Beyond the Vendor's Checklist" and "Beyond Installation". [Garcia2002a, Garcia2002b] Since many small to medium sized companies or organizational units are facing COTS adoption efforts, we believe that the successful approach we used with the small manufacturers has applicability to small and medium enterprises in general.

Figure 1 defines the notional life cycle we used with small manufacturers. The process elements highlighted in blue are those that were added to specifically address non-

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 74–83, 2004.

technical adoption risk management. They are the ones that are the focus of this paper. Each one is described, and some are elaborated with notes on particular techniques or representations that are used to facilitate execution of that element.



Carnegie Mellon
Software Engineering Institute

A Notional Technology Selection/ Implementation Life Cycle

|← Phase 1 →| ← Phase 2 →| ← Phase 3 ←|

Understand Existing Company Environment → Establish Technology Adoption Project Goals and Metrics → Evaluate Technology Options → Obtain Technology → Implement & Adopt Technology → Analyze and Deliver Adoption Project Results

Define Initial Requirements

Analyze Workflow

Identify Metrics

Baseline Orgn Adoption Factors

Begin Metrics Collection

Identify Improvement Options

Draft Business Case

Draft Technology Adoption Plan

ID Adoption Risks

ID/Use Selection Criteria to Evaluate Potential Tools

Evaluate Cost vs Benefit

Test Using Example Problem

Evaluate Fit of Technology to Organization Factors

Procure technology

Build/deploy transition mechanisms for communication

Finalize Technology Adoption Plan

Establish Level Of Use Goals

Install/ Test Technology

Build/deploy transition mechanisms for implementation

Manage Adoption Issues/Risks

Roll Out to Relevant Stakeholders

Document Business Impact

Collect Technology Adoption Lessons Learned

Adoption Risk Management Activities

© 2001 by Carnegie Mellon University

1

Fig. 1. SEI's TIDE Technology Adoption/Implementation Life Cycle

2 Baseline Organization Adoption Factors

This element involves the process of understanding non-technical factors in the organization that have historically affected (either positively or negatively) COTS implementation. Factors related to the business strategy, work practices, and internal environment of the organization are analyzed to understand the historical pattern of the organization related to technology adoption. Several of these factors are derived from Paul Adler's work related to updating a company's technology base, and they are supplemented by SEI research related to managing technological change in organizations. [Adler1990]. Areas and things to consider include:

- Work practices – How easily do we historically implement work practice changes related to adoption of a COTS product?
- Skills – Do we traditionally ensure that employees have relevant technical experience and/or project management experience related to adopting COTS software?
- History—what lessons have we internalized related to our past history of COTS adoption?

- Values – How well have we matched our own company values to the values implied by COTS packages we have adopted in the past? (ie have we understood that some ERP packages have specific expectations related to how data is shared, and looked at whether or not we agree with those expectations?)
- Structure – How well have we historically recognized the (potential) need for new roles and responsibilities when a COTS package was implemented?
- Reward System – How well have we constructed reward systems that encourage use of the COTS package and discourage continuation of old practices?
- Sponsorship – How well does sponsoring management for a COTS adoption "walk their talk" by recognizing and reinforcing use of the new COTS system?
- Business strategy—how well have we matched the COTS products we choose to relevant elements of our business strategy?

Understanding the company's historical pattern in these non-technical risk areas can help to avoid COTS selections that are likely to play to weaknesses in the organization vs its strengths.

3 Identify Adoption Risks

This element continues the analysis of non-technical adoption risk by moving from the historical perspective to looking at the risks that are inherent in the current COTS software adoption being contemplated. The kind of information gathered falls in the following areas (note these are the same categories as for baselining the organizational factors, however, the questions are now re-oriented to address the probable risks related to the particular product/technology under consideration):

- Work practices – What procedures, techniques, processes, will need to be changed for the COTS software to operate successfully? How drastic are the expected changes?
- Skills – Do employees have relevant technical experience and/or project management experience related to adopting this type of COTS software?
- History -- Has the company adopted this type of COTS software before? What are the lessons learned?
- Values – Do users and managers share a vision of how the COTS software will be used and how it will help the company to achieve its business goals? Does the software assume particular values (for instance, making an assumption about how widely certain data is shared) ? How well does the organization's practice mesh with the values expected by the software?
- Structure – Does the software imply any particular roles, responsibilities, interfaces? How well does the organization mesh with those implications?
- Reward system – Are there incentives for adoption/use of the software and disincentives for not adopting?

- Sponsorship – Does management understand the software and how it will change and help the company? Is it committed to actively supporting the new way?
- Business strategy—how well does the software being considered mesh with the business strategy of the organization?

A helpful analysis aid can be created to guide organizations that are unfamiliar with a particular technology or product type (a consistent issue with small organizations we worked with). This involves creating an "implications table" that addresses each category in relation to the technology under consideration. For each category, the inherent implications of the COTS product type or product is listed, so that, when the self-evaluating organization reviews it, they have a more concrete idea of the types of things they would "see" if they implemented the candidate COTS product. Figure 2 is an example table created for the general category of MES (Manufacturing Execution System) COTS products. Note that 4 specific work practices are called out as being implied by this technology type. If work practice implications are a significant element of the COTS product's implications, then a separate table just devoted to work practice changes could be constructed.

Table 1. MES Technology Implications Table

Adoption Factor	Dynamic Scheduling Implication
Business Strategy	-improving operations is a priority
Work Practices	-organized, decomposed job description throughout the jobs -manufacturing intent is communicated through engineering drawings -master scheduler is an implied role for scheduling system -scheduling system is primary reference point for capacity planning
Reward system	-rewards participation in overall efficiency over individual dept efficiency
Sponsorship	-strong, consistent support for "new way" -penalties for avoiding new system consistently applied
Values	-shared data used for team decision making -metrics are used to improve, not to punish
Skills	-skills to operate the system -project planning/mgmt skills
Structure	-clear roles/ responsibilities
History	-helpful if complex technologies successfully adopted with this mgmt team

The list of adoption risks that this analysis spawns becomes the basis for the Adoption Risk Management Plan for the COTS adoption. You will find that sometimes the risk mitigation activities that would be useful for these risks have already been identified and laid into the implementation plan, but many of them (for example, communications planning) are commonly ignored if an explicit adoption risk exercise is not conducted.

4 Evaluation of Technology Adoption Risk Factors

Once the relevant management stakeholders (usually the leadership team of the company) have identified the organization's adoption risks in the above areas, the next element, "Evaluation of Technology Adoption Risk Factors", involves a self-evaluation by the selection/implementation team, evaluating the risk categories above on a scale of one to five. ("One" meaning "not a good fit with the way we do things now", "Five" meaning "an excellent fit with the way we do things now".) Plotting those factors in a Kiviat diagram or histogram (I favor the Kiviat, but others prefer the histogram representation, either one will do) will depict the organization's adoption readiness profile (see example in Figure 3) in relation to this COTS product type or product (depending on the scope of the adoption). Since the factors haven't been weighted for importance, the presentation cannot accurately predict overall success or failure. However, it can help the organization to identify and start to mitigate the non-technical risks before installing any software.

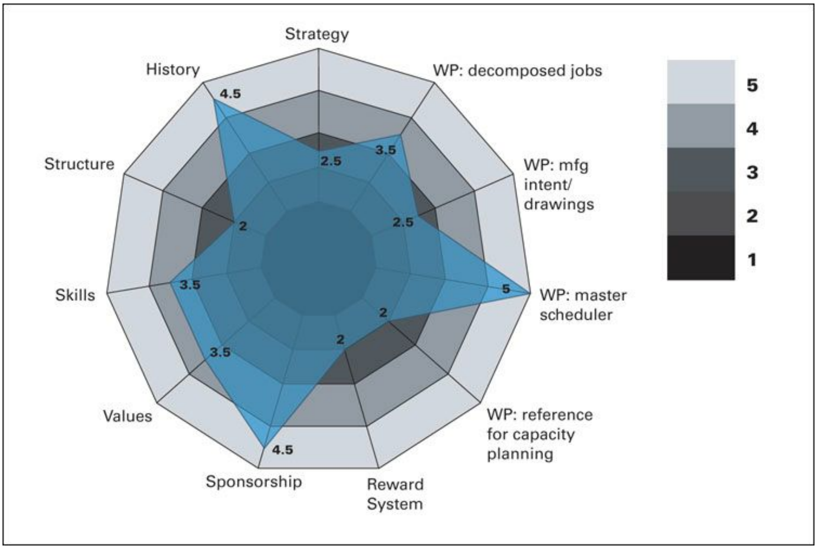


Fig. 2. Example Readiness Profile for MES Adoption in a Small Manufacturing Company

The answers (both in terms of the readiness profile and the actual risks identified) will help determine whether the candidate technology being considered is a good "fit" with the organization's business strategy, work practices, and internal environment. To determine whether individual products are a good fit for the organization, compare candidate software products to current work practices, skill levels, etc., using the risks identified as a guide for areas to probe with vendors. For example, some software products are much more powerful and complex than others. As a result, they may require rigorous training or highly skilled and experienced personnel. They may also require significantly changing workflow or other processes. The organization may or may not be prepared to invest in the skill development and work practice changes required. The point is that being explicit about the decision to install and adopt a particular product with knowledge of what risks are being accepted BEFORE making the costly and often irrevocable decision to buy the software product reduces the overall risk to successful adoption.

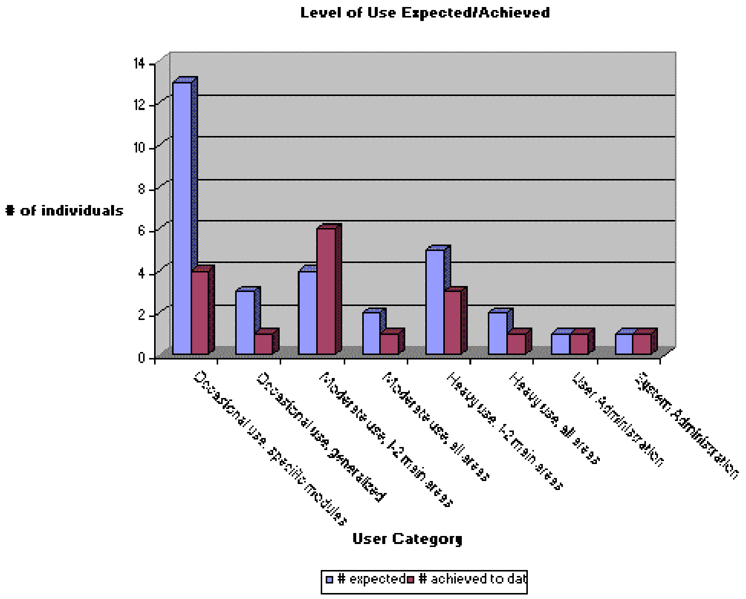


Fig. 3. Level of Use Profile at Time X

5 Establishing Level of Use Goals

At this point, the organization is typically ready to purchase and install the software. The next adoption challenge will be getting everyone in the organization who needs to be using the software to actually use it. The element, "Establishing Level of Use Goals" is one aspect of addressing this issue. Ideally, the organization should establish "level of use" criteria for each role in the company that will be expected to use the product in a significant way. This requires identifying employee tasks, mapping each task to relevant software and specific operations, and, ultimately, making sure that the employees in each role get the training and implementation support needed to achieve the level of use required for their role. In some organizations, it is helpful to categorize users and create a profile that displays a snapshot of the level of use achieved by the different categories of users. Figure 4 provides an example of such a profile.

From a technology transition literature viewpoint, this approach reflects the practices recommended by Zmud and Apple related to measuring technology infusion. [Zmud1992]

6 Building/Deploying Communication Mechanisms and Building/Deploying Implementation Support Mechanisms

"Diffusion" refers to the process of moving technology across the organization/population. For example, an organization may want to implement a manufacturing execution system (MES) to help improve production efficiency. But some of the data can also help sales personnel predict delivery dates. Some data can also help the front office streamline billing. To assure success, employees in ancillary areas, as well as primary use areas, should know how the system fits together, and what part each employee plays in it. To be effective at diffusion of the software throughout the company, it is useful to understand the typical cycle that individuals go through before committing to use a new technology, highlighted below in Figure 5.

In general, the "adoption commitment" process flows through discrete stages:

1. Contact – Introduces the technology in terms of its capabilities, i.e. the big picture.
2. Awareness – Relates the technology to the company, department, individual.
3. Understanding – Identifies individual stakeholder roles, responsibilities and relationships, and establishes sufficient understanding of the technology to know how to change work practices to handle it
4. Trial Use – Establishes test cases for individuals, tasks, or capabilities – the first use of the technology for a particular set of users.
5. Adoption – Promotes proper use by employee and across organization.
6. Institutionalization – Incorporates the technology into the infrastructure of the organization and builds upon it. [Patterson1982]

A transition agent can help move the diffusion process along using the appropriate support mechanisms to help people move productively from one stage to another. For example, "communications mechanisms" such as magazine articles and vendor briefings can introduce the technology in the Contact and Awareness stages. Seminars, case studies, technical briefs, and other detailed communication mechanisms can help employees in the Understanding stage. During this stage, the organization should also begin to use "implementation mechanisms." These include specifying stakeholder roles, responsibilities, and relationships, as well as specifying measures to use to verify that the technology actually provides the anticipated benefits. (See the TIDE tutorial, "Beyond Installation" on the SEI/TIDE website for more detailed lists of typical communication and implementation support mechanisms, at www.sei.cmu.edu/TIDE)

Pilot trials typically require a combination of communications and implementation support. For example, many SMEs rely on user groups or outside consultants for the communications help, and written policies, incentives, and training aids for implementation support.

During the Institutionalization phase, communications support could include new employee orientation and training courses for different users. Implementation support could include training classes and mentoring. Many communications and support mechanisms are available from the software vendor, user groups, outside consultants or other sources. However, the organization may have to develop others in-house.

To make sure the diffusion process is successful, the organization should measure it just as they measured the infusion process. This requires developing measures and exit criteria for each stage. Table 2 can help in producing a diffusion profile at different times during the adoption.

Measuring diffusion provides an interim measure of how the adoption is progressing. Once the technology is firmly in the Adoption stage, then it s appropriate and useful to measure business benefit/ROI…one of the benefits of the diffusion measure is to help identify the appropriate time to push for more traditional, ROI-based business measurements. See the "Beyond Installation" tutorial for example profiles of diffusion at different points in an organization's deployment of a new technology.

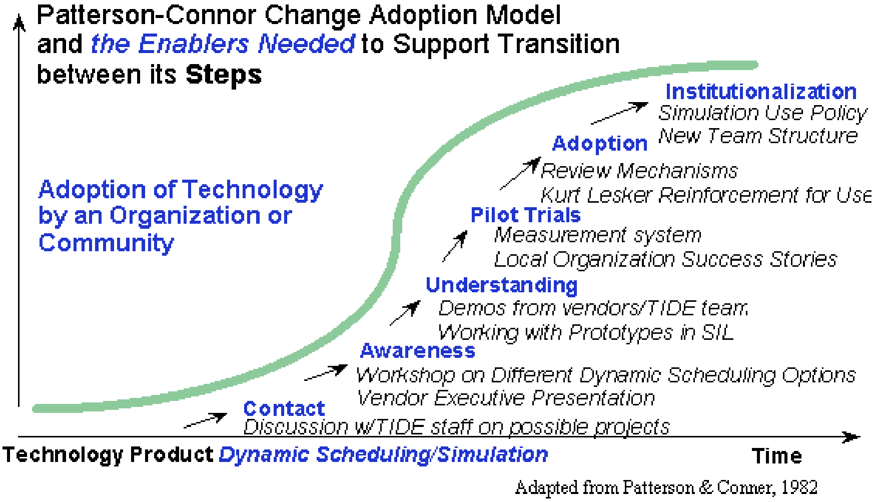

Patterson-Connor Change Adoption Model
and the Enablers Needed to Support Transition
between its Steps
Adoption of Technology
by an Organization or
Community
Institutionalization
Simulation Use Policy
New Team Structure
Adoption
Review Mechanisms
Kurt Lesker Reinforcement for Use
Pilot Trials
Measurement system
Local Organization Success Stories
Understanding
Demos from vendors/TIDE team
Working with Prototypes in SIL
Awareness
Workshop on Different Dynamic Scheduling Options
Vendor Executive Presentation
Contact
Discussion w/TIDE staff on possible projects
Technology Product Dynamic Scheduling/Simulation
Time
Adapted from Patterson & Conner, 1982

Fig. 4. Patterson-Conner Technology Adoption Commitment Curve

Table 2. Aid for Creating Adoption Diffusion Measures

Stage	Questions	Exit Criteria	Example measure: Number of employees
Contact/ Awareness	What is it? Why use it?	Employees should be able to answer questions	Attending meetings, receiving information
Understanding	How will we change?	Employees should have detailed knowledge of software and level of use goals	Attending user courses, Participating in meetings to define work practice changes
Trial Use	How will we implement it?	Explicit knowledge of what must be done	Using technology Participating in pilot
Adoption	Are we using it effectively?	Achieve Level of Use goals	"Passing" audit/review of use
Institutionalization	How do we keep it in place?	Not using it has sanctions	(these are more orgn infrastructure-related) All licenses up to date Procedures documented

7 Managing Adoption Issues and Risks

No matter how smoothly the implementation effort has gone, there are bound to be issues that come up. Some of these issues are predictable and were identified as part of the "Identifying Adoptio Risks" element. The "Managing Adoption Issues and Risks" element involves monitoring the installation effort and the risks that were previously identified, identifying additional trouble spots in the implementation, and developing mitigation tasks as they come up. The goal of this effort is to make sure that the COTS software doesn't become "shelfware". By proactively monitoring the originally-identified adoption risks to see if they have manifested, and by identifying and dealing with unanticipated issues that have been seen during the implementation, adoption risks can usually be contained or mitigated. To achieve this type of monitoring, usually some type of integration of the adoption risk management into the implementation status monitoring/management needs to be achieved.

8 Collecting Technology Adoption Lessons Learned

During the element “Collecting Technology Adoption Lessons Learned,” company executives, implementation team members, and department managers meet to share information about the activities and results related to adopting the COTS software. This activity can also serve as a springboard for continuous improvement activities and hardware and software upgrades. The focus of this lessons learned collection is the practices the organization used, successfully and unsuccessfully, to identify and manage the known adoption risks, and to identify the risks that were not anticipated that turned into issues. This project has now become part of the organization’s history, and the baseline of organizational factors may be revisited to see how the adoption history of the organization has changed (if at all) as a result of this COTS adoption project.

9 Summary

Taken together, adding the adoption risk management elements to the typical COTS selection and technical implementation tasks can help the company management make better informed decisions about selecting a software product, and can help the organization plan and implement the technology successfully to reach their desired usage goals for the COTS product.

Managed technology adoption can help organizations maximize the return on their COTS software investment. For small organizations in particular, looking to increase productivity, improve quality, and reduce time, the investment in managed COTS technology adoption is the natural complement to the decision to invest in the technology itself.

References

[Adler 1990] Managing Your Technological Base, Adler, P. & Shenbar, A., Sloan Management Review, Fall 1990, pp. 25-37

[Fichman1995] The Illusory Diffusion of Innovation: Assimilation Gaps, Fichman, R., & Kemmerer, C. CISR Working Papers, 1995, #294.

[Garcia2002a] Beyond the Vendor's Checklist, Garcia, S.M., TIDE Conference Sept 2003. url: http://www.sei.cmu.edu/tide/

[Garcia 2002b] Beyond Installation, Garcia, S.M., TIDE Conference Sept 2003. url: http://www.sei.cmu.edu/tide/

[Patterson1982] Building Commitment to Organizational Change, Patterson, R. & Conner, D. Training & Development Journal 36, 4 (April 1982): 18-30.

[Thong1997] Effects of resource constraints on small business, Thong, J.Y.L. and Yap, C.S., International Working Conference on Diffusion, Adoption and Implementation of Information Technology, 1997 Tom McMaster, editor, Chapman & Hall, London, 1997, p. 191-206

[Zmud1992] Measuring Technology Incorporation/Infusion, Zmud, RW, & Apple, LE, Journal of Product Innovation 9, 2 (June 1992): 148-155.

Understanding Services for Integration Management

L. Davis and R. Gamble

Department of Mathematical and Computer Sciences
The University of Tulsa
600 South College Avenue
Tulsa, OK 74104 USA
{davisl,gamble}@utulsa.edu

Abstract. With the advent of web services, service-oriented architectures (SOAs), which promise interoperable communication in an application integration, are now primarily web-based. Due to the high degree of encapsulation and heterogeneity among commercial-off-the-shelf (COTS) components, their use is limited within these SOAs. This is because the definitions of web-based services do not differentiate between component services and integration functionality nor has this integration functionality been classified specifically for COTS components. Understanding what inhibits COTS components from participating in a SOA is essential to enabling their integration. In this paper, we identify and define common enablement services that facilitate service-oriented integrations in which COTS components participate as *integration management services.* Software architecture can express integrated system design by identifying needed integration functionality independent of communication mechanisms. Therefore, we describe common web services in architecture terms and, through design patterns, define the integration functionality by which they are distinguished.

1 Introduction

Industry depends on the internet to unite their often distributed IT components in an effort to cooperatively store and use existing resources and services. This requires integrating disparate software systems, services, and information producing an application integration. *Service-oriented architectures* (SOAs), which utilize middleware as communication frameworks and partition component functions into logical, homogeneous modules for use by clients, have emerged to meet the demand for low cost, fault-tolerant, decoupled, interoperable systems. To leverage the internet for the greatest advantage to businesses, new computing paradigms such as *web services* have been developed using XML and SOAP as the basis for their standard.

With the advent of web services, SOAs have become increasingly web-based. SOAs have emerged that incorporate web services registration and dynamic service discovery within the confines of XML and SOAP [13, 19]. Web services-based SOAs claim to provide interoperable communication for legacy systems and commercial-off-the-shelf (COTS) components. However, this presupposes that components registering their services with the SOA manifest analogous communication properties and security expectations. Thus, integrating COTS components by describing their

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 84-93, 2004.

services in a Web Services Description Language (WSDL) document and registering it with a SOA does not solve the interoperability issues at stake.

Interoperability problems arise when communication between components and between components and an application is impeded. In order to provide transparent communication in an application integration, these problems must be resolved. It follows, due to the heterogeneous and encapsulated nature of COTS components, extra functionality that fulfills communication expectations is required to enable the integration of their services in a web-based SOA. These problems are further compounded by on-demand communication, a frequent paradigm in such systems. The correction of mismatches in service definitions for COTS components during dynamic communication can be especially daunting.

Traditional SOAs often provide some integration functionality as a centralized management layer. This layer does not take into account highly disparate data formats and communication strategies. Web services-based SOAs implement enablement and integration functionality as web services, promoting a high degree of decoupling such that services can be managed and destroyed correctly. However, there is no present differentiation in web services standards between computational services and those that manage service communication. In order for SOAs to become viable mechanisms for dynamic application integration these management services must be delineated.

In this paper, we identify common services necessary to enable COTS components to participate in a web-based SOA. We define these service types as *integration management services* because they allow the explicit management of the integration functionality that binds a component to a SOA. We use patterns to establish the inherent integration functionality in integration management services such that they not only support their definition, but they promote the identification of these services to facilitate dynamic application integrations using SOAs.

2 Motivating Example

A business constructs a SOA to automate profit tracking and liquidity assessment to aid upper management in making decisions concerning spending and acquisitions. For example, a user requests a profit analysis based on the company's market exposure, the size and expense of their shipping fleet, the performance of their stock, and their holdings and acquisitions, and receives an assessment of the company's current profitability. The SOA consists of legacy components that maintain company status, COTS assessment components, and COTS middleware.

Fig. 1 depicts the five main services of the components of the profit tracking system. *Risk* contributes client, portfolio, instrument, segment, and sector data. *Logistics* provides shipping, tracking, and accounting information. *Market History* supplies company stock performance information as well as the performance information of acquired companies and companies with which there is a strategic alliance. *Knowledge Manager* organizes and aggregates the incoming data from Risk, Logistics, and Market History. *Profitability Assessment* analyzes particular financial data (e.g., the data supplied by Risk, Logistics and Market History) to provide profit reports and a liquidity assessment. Risk, Logistics and Market History can be queried simultaneously, and the data that they supply can be buffered and forwarded appropriately via messages. When executing within the company's intranet, confidential or sensitive information, which the system utilizes, is protected. The architecture of the system is static.

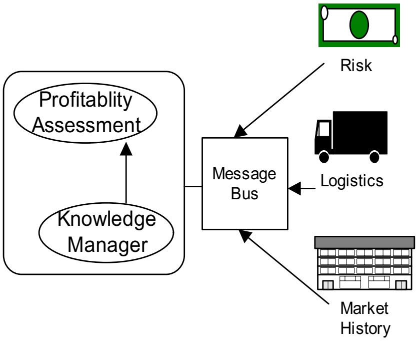


Fig. 1. The Profit Tracking System

3 Relevant Background

Distributed, integrated systems require correct, secure, and evolvable solutions as components often cannot provide these guarantees at their interface [6, 7]. Connectors have become increasingly important to COTS integration research. They form the basis for component interaction, and are the means by which COTS provide services in application integrations [2, 12, 14]. Complex connectors that bind disparate components in application integrations are often specified using design patterns [4, 5, 12, 16]. These enabling functions or processes are deployed "external" to the component to allow it to interact with other components and with a middleware platform or environment. These functions are generally in the form of translation and transformation, routing and decision making, functional calls and polling, and security mechanisms. Some examples of these connectors are as follows.

- *Integration elements* [12] provide a uniform description of integration functionality found in particular architecture and design patterns, e.g., adaptor, proxy, etc [5, 9].
- *Protocol transformation* [18] wraps components to resolve interoperability problems.
- *Distributor connectors* [14] direct data flow through brokering and routing decisions.

Web Services*,* which promise minimized integration efforts, are an emerging generation of distributed technologies. They can be described as autonomous, modular applications, which execute over the internet to perform a particular task, and comply with a specific technical format, i.e., WSDL [10, 13, 15, 19]. Web services frameworks offer platform-level integration, i.e., they use well-established protocols as a standard, to eliminate reliance on factors such as devices, operating systems, middleware solutions, and programming languages. Such technologies as Jini, JXTA, XML, and SOAP can be used to enable dynamic registration and discovery of web services in SOAs. However, they prescribe particular communication properties to the SOA, requiring all component services to communicate homogeneously in order to be integrated in the system [13]. At present, integration functionality that enables components, without the proper XML interface, to communicate as web services is not addressed or described in standard web services research.

To allow the use of web services outside the confines of XML, web-based services have been implemented to facilitate the incorporation of web service in established middleware frameworks such as CORBA, EJB, COM+ or .NET systems [15, 19]. Services such as these that enable components as web services in such middleware-based SOAs encompass interoperability issues and security [15]. However, at present these services are not classified as integration enablers, existing only as ad hoc solutions independent of the middleware design.

4 The Architecture of Integration Management Services

The designers of the profit tracking system outlined in Section 2 want to market it to other businesses as an on-demand service. The new system must accommodate subscribers from manufacturers, distributors, and portals. Its performance must be scaled to real-time while protecting the integrity and confidentiality of subscriber data. Based on these requirements, large-scale provisioning of the system's functionality excludes static integration of every subscriber's components. Performance would be prohibitively slow and the system would no longer be protected by intranet security, making subscriber data vulnerable.

A better approach allows the dynamic connection and disconnection of subscriber components. When a subscriber desires an assessment based on their financial exposure, market history, and real-time stock market information, their component services, along with New York Stock Exchange, Australian Stock Exchange, and NASDAQ real-time monitors will be "temporarily inserted" into the profit tracking system environment. These component services will have interfaces distinct from existing services, provoking interoperability problems. Because of such heterogeneity, additional integration functionality is necessary to enable these services in the SOA [3, 10, 19, 21]. Furthermore, the web-enabled system must be interoperable and fulfill the security requirements of the participating components and the original application. Thus, given that the existing SOA can be web-enabled, the developers still require technology for dynamic assessment followed by the instantiation of web-enabling *integration management services* for each of the components.

In the remainder of this section, we define the services of protocol translation, reliable messaging, security and synchronous/asynchronous conversion using established pattern languages as a foundation [5, 9]. We minimize the categories presented for clarity and space considerations, focusing on those most appropriate to the integration management service patterns. The services can stand alone or be composed depending on the degree to which the component communicate and the expectations of other components and the application. We use UML stereotypes for component, translator, controller, and extender following research definitions for architecture components and integration element connectors [4, 5, 12, 16]. We then return to the profit tracking system, and discuss how the services are deployed.

4.1 The Protocol Translation Service

Name: Protocol Translation
Problem: Messages and data that components communicate as part of their services are formatted differently than that of the SOA environment into which they are to be inserted.
Forces: Dissimilar protocols (e.g., SOAP, JMS, CORBA, and DCOM); Lost, garbled, or incomprehensible messages should not occur.
Context: A SOA has a different communication protocol, whether web-enabled or not, than the connecting component.
Solution: Implement a message translation service to manage how a component communicates messages/requests to a SOA [3, 18, 19]. This service adapts outgoing calls to the proper format for the SOA and adapts incoming calls to a component format.

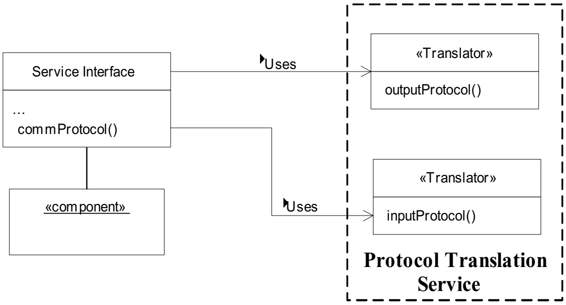


Fig. 2. The Protocol Translation Service

A protocol translation service, depicted in Fig. 2, is composed of one or more translation functions. Any requests made to the component's service interface are directed first through the incoming translator, so the request mirrors the communication protocol of that component. The outgoing protocol translation service uses the service interface to provide an intermediate format translation to that of the environment. The product of this translation is then forwarded.

4.2 The Reliable Messaging Service

Name: Reliable Messaging
Problem: The execution of components that expect delivery confirmation when they communicate messages could be suspended indefinitely, causing deadlock, if the SOA does not provide delivery guarantees.

Forces: Need to provide and require message/data delivery confirmation; Guaranteed message delivery.
Context: A SOA that asynchronously communicates its messages does not necessarily provide acknowledgements to requests.
Solution: Implement a service that simulates a direct handshake between the component and SOA environment [1, 6, 11]. As depicted in Fig. 3, this integration management service monitors the component's service interface to which it is bound for failure as well as for incoming and outgoing messages. If the component fails, the reliable messaging service retains all messages and polls the failed component until it comes back online. Upon restart, all communications are forwarded appropriately and a callback to the communicating component is issued as acknowledgement of its request.

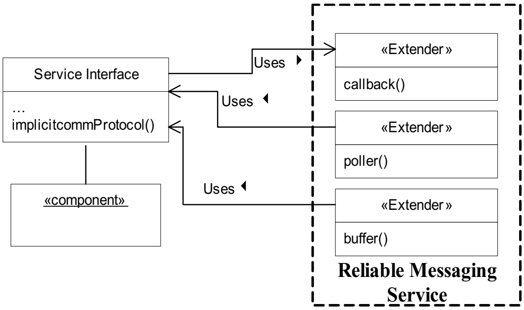


Fig. 3. The Reliable Messaging Service

4.3 The Security Service

Name: Security
Problem: Components are often not equipped with the security mechanisms necessary to communicate in a web-based environment. Generally, this is because they do not expect to communicate actively or in a distributed fashion.
Forces: Dynamic connections to a web-enabled SOA; Messages should be transferred securely to the SOA environment as a special form of communication.
Context: A SOA requires secure communication, authenticating and certifying users, as well as encrypting messages and data.
Solution: Implement a service that provides security mechanisms for a component to properly communicate in a web-enabled environment. This service fields all messages and requests to ensure they come from authorized participants and generates credentials or encrypts data such that outgoing communications are consistent with the expectations of the SOA [3, 17, 19]. It this way, communication conflicts such as inhibited rendezvous, and mismatched data formats are resolved [6]. The service requires a process to buffer requests and an authorization enabler to field incoming and outgoing messages (see Fig. 4), executing embedded security mechanisms such as authentication and certification. If the security service is being used to handle encryption, it must be coupled with a protocol translation service. Depending on the level of secu-

rity dictated by the application expectations, another buffer may house previously authorized components (depicted in the dashed box in Fig. 4).

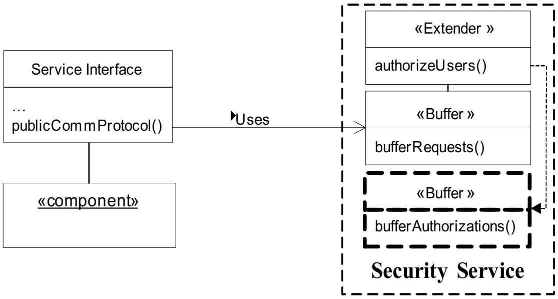


Fig. 4. The Security Service

4.4 The Synchronous/Asynchronous Conversion Service

Name: Synchronous/Asynchronous Conversion
Problem: A component communicates either by directly calling the SOA and expecting a direct response, or by broadcasting a request for insertion and being provided the facilities to communicate with the SOA. The interoperability problem stems from the SOA expecting the opposite type of communication.
Forces: Transparent message passing; Connectivity expectations; Dynamic communication and insertion.
Context: An SOA implemented using a middleware platform which supports the communication paradigms asynchronous messaging or request and reply, but not both.
Solution: Implement a service that changes the mode of message transfer for a component. The synchronous/asynchronous conversion service enables component to insert itself, for example, by adapting a component's implicit communications to explicit calls [3, 20]. Fig. 5 illustrates the synchronous/asynchronous conversion service as the composition of a buffer for housing both incoming or outgoing messages and a bi-directional router to direct and forward communications.

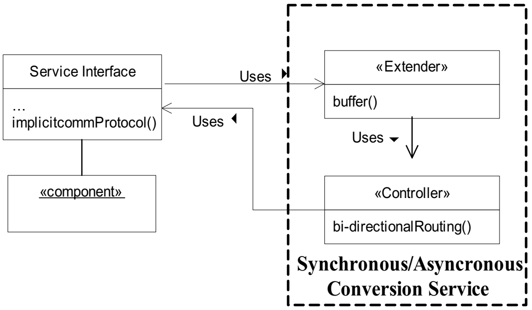


Fig. 5. The Synchronous/Asynchronous Conversion Service

Notice in Fig. 1 - 4, we partition the internal component functions from its external service interface. The interface functions, i.e., the functions directed through the integration management services of components, are identified through comparison of the architectural properties of the component to the expectations of the SOA environment [8]. Without first describing the component using uniform properties, it is difficult to assess what integration management services are needed for interaction enabling. Defining the integration management services in terms of patterns provides structure and easy reuse.

Returning to the profit tracking example, Fig. 6 depicts the system with three integration management services. Protocol Translation reformats data on the part of Risk, and Market History to mimic that of the established SOA. Security provides credentials to NYSE, ASX, NASDAQ, Risk and Market History, certifying their data, as the SOA requires guarantees of authenticity for sensitive information. Reliable messaging assists Risk and Market in forwarding their data by emulating direct communication and in supplying a message receipt to those components on the part of the system. They will then provide their information to the knowledge manager and disconnect.

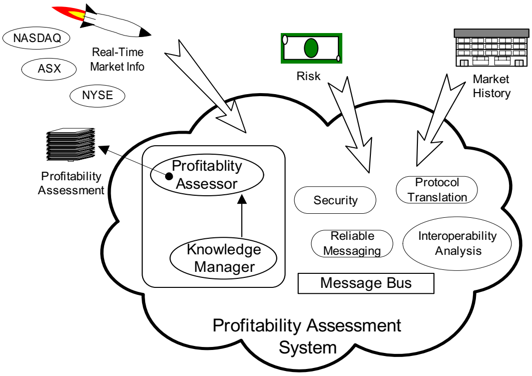


Fig. 6. The Profitability Assessor

5 Conclusions

Integration management services supply the technology to enable interoperability and make seamlessly available a component's services for use in web-enabled SOAs. Unlike a static design that centralizes its integration solution to form a single, encapsulated system, integration management services promote the high decoupling necessary to enable components as services in such architectures. However, the present definition of web services is too specialized and classification of web services outside of component computations is ambiguous and mired in contradiction. Thus, the differentiation of such enablement services from traditional web services is important to further promote use of web services SOAs for application integrations. Yet, the current research in web services does not presently offer a principled approach to such classification.

In this paper, we present common enablement services as integration management services and pattern these services. These patterns localize the integration functionality present in these services and define the architectural forces (e.g., the properties of the interface that cause interoperability problems requiring extra-functional solutions) that influence the enablement of components as services. Using this approach, enablement services can be distinguished from component services, offering the design guidance and uniformity needed for SOAs to become viable mechanisms for application integration.

Acknowledgements. This material is based upon work supported in part by AFOSR (F49620-98-1-0217) and NSF (CCR-9988320). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or the US government. The government has certain rights to this material.

References

1. Web Services Glossary:W3c Working Draft. W3C, http://www.w3.org/TR/ws-gloss/, May 14 (2003)
2. Allen, R. Garlan, D.: A Formal Basis for Architectural Connection. ACM Transactions on Software Engineering and Methodologies, 6 (3). 213-249
3. Bloomberg, J.: Web Services Management: Successfully Architecting the Future of Your Business. Zapthink, Waltham, MA, November (2002) 1-12
4. Bonura, D., Culmone, R. Merelli, E.: Patterns for Web Applications. 14th Int'l SEKE. Ischia, Italy July 15 -19 (2002) 739-746
5. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns. John Wiley & Sons, (1996)
6. Davis, L., Flagg, D., Gamble, R. Karatas, C.: Classifying Interoperability Conflicts. 2nd Int'l ICCBSS. Ottawa, Canada (2003)
7. Davis, L. Gamble, R. F.: Identifying Evolvability for Integration. 1st Int'l ICCBSS. Orlando, Florida (2002)
8. Davis, L. Gamble, R. F.: The Impact of Component Architectures on Interoperability. Journal of Systems and Software, 61 (1). (March 1) 31-45
9. Gamma, E., Helm, R., Johnson, R. Vlissides, J.: Design Patterns Elements of Reusable Object-Oriented Software. Addison-Wesley, (1995)
10. Gergic, J., Kleindienst, J., Despotopoulos, Y., Soldatos, J. Polymenakos, L.: An Approach to Lightweight Deployment of Web Services. 14th Int'l SEKE. Ischia, Italy July 15-19 (2002) 635-640
11. Ingham, D., Shrivastava, S. Panzieri, F.: Constructing Dependable Web Services. IEEE Internet Computing, 4 (1), January/February (2000) 25-33
12. Keshav, R. Gamble, R.: Towards a Taxonomy of Architecture Integration Strategies. ISAW - 3. 1-2, November (1998)
13. Kleijnen, S. Raju, S.: An Open Web Services Architecture. ACM Queue, 1 (1), March (2003) 38-46
14. Mehta, N., Medvidovic, N. Phadke, S.: Towards a Taxonomy of Software Connectors. 22nd International Conference on Software Engineering. (2000)
15. Pierce, M., Fox, G., Youn, C., Mock, S., Muller, K. Balsoy, O.: Interoperable Web Services for Computational Portals. IEEE/ACM SC2002 Conference. Baltimore, Maryland November 16 - 22 (2002)

16. Schmidt, D., Stal, M., Rohnert, H. Buschmann, F.: Pattern-Oriented Software Architecture, Volume 2, Patterns for Concurrent and Networked Objects. John Wiley & Sons, Inc., New York, NY (2000)
17. Spitznagel, B. Garlan, D.: A Compositional Approach for Constructing Connectors. WICSA'01. Amsterdam , The Netherlands (2001) August 28-31
18. Spitznagel, B. Garlan, D.: A Compositional Formalization of Connector Wrappers. 25th International Conference on Software Engineering. Portland, OR May 3-10 (2003) 374-384
19. Stal, M.: Web Services: Beyond Component-Based Computing. Communications of the ACM, 45 (10). (October) 71-76
20. Vinoski, S.: Where Is Middleware? IEEE Internet Computing, MARCH/APRIL (2002) 83-85
21. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J. Sheng, Q. Z.: Quality Driven Web Services Composition. 12th Int'l WWW Conference. Budapest, Hungary May 20-24 (2003) 411-421

Migrating Application Integrations

D. Flagg, R. Gamble, R. Baird, and W. Stewart

Department of Mathematical and Computer Sciences
The University of Tulsa
600 South College Avenue
Tulsa, OK 74104 USA
{flagg,gamble,robert-baird,william-stewart}@utulsa.edu

Abstract. The internal functionality of middleware is highly variable and thus, well-constructed integrations are difficult to perform without understanding the architectural style of the middleware and the adaptive connections needed make components in an application integration "middleware-aware." In this paper, we use IBM's WebSphere® MQ to implement two different architectural styles of integration: request/reply and publish/subscribe. The middleware supports both approaches by using different configurations of controlling, routing, and translating functionality within the connectors. By explicitly describing the component connectors attached to the middleware, we discuss the trade-offs that exist between *centralized solutions* in which the middleware is responsible for the majority of the integration functionality and *localized solutions*, in which application connectors are responsible for integration to the largest extent possible.

1 Introduction

Despite the wide array of COTS middleware, evolvable and dynamic application integration is still difficult. As middleware companies attempt to design the perfect integration solution, they must increasingly expand their product functionality. An unfortunate result of this "feature explosion" is that it enables middleware frameworks to be shoehorned into nearly any application integration if enough implementation effort is exerted. For example, IBM WebSphere® MQ can implement request/reply [1], publish/subscribe [1], and fire-and-forget paradigms. In Borland VisiBroker, the CORBA functionality can simulate a publish/subscribe system using their Publish/Subscribe Adapter. Thus, while the ability to implement an application integration is a certainty given most middleware solutions, it may not necessarily be a "good" implementation.

Many application integrations revolve around streamlining old business processes. It makes sense that, throughout the lifecycle of a distributed system, shifting requirements will necessitate that components be added, removed, and modified. More dynamic systems minimize the down-time required to implement system modifications and insertions. Dynamism is therefore an important quality that should be addressed in all application integration endeavors.

Research in software architecture and its place in application development has examined both evolvability [4, 5, 12, 16] and dynamism [2, 3, 8]. Evolvable architectures share similar qualities with dynamic architectures, such as the ability to determine how

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 94–103, 2004.

to best change established architectures, accommodate new component versions, and facilitate the addition and removal of components. A common thread in facilitating evolvability and dynamism is the concept of *architecture connectors.* Connectors are considered as first class objects with many possible incarnations. Each component communicates through the connector to which it is attached to transfer information to recipient components using one or more middleware technologies [9]. By singling out a specific architecture entity in this way, it is possible to examine the trade-offs among connector content, complexity, and knowledge of the environment.

When considering a particular application integration architecture style, there are many tradeoffs to be considered. These include how the middleware technology can best contribute to the application integration, what architecture style can be used to facilitate both performance and flexibility, and where the main pieces of the integration solution should reside – central to the middleware or housed local to the component connector to the middleware. In this paper, we address these issues by examining the core functionality of IBM WebSphere MQ and the impact of architecture style on integration, i.e., request/reply vs. publish/subscribe. We use as an example a distributed personal librarian system that progresses through a migration from a request/reply to publish/subscribe paradigm to detail specific aspects related to middleware, architecture style, and connector structure with respect to the migration.

2 Relevant Research

Software connectors are architecture abstractions that facilitate interaction among components. Connectors manifest themselves in software systems as access points for shared variables, table entities, buffers, procedure calls, remote procedure calls, network protocols, pipes, etc. [9]. Connectors are an important abstraction for application integration. There exists explicit notation for them as first class entities, and they are well-suited for distributed, heterogeneous message-based environments [6, 7, 10, 13].

Constructing connectors is a key architecture research area whose goal is to satisfy the need for specialized forms of interactions required to bridge component mismatches or to achieve extra-functional properties for performance and reliability [14]. In our context, wrappers are also considered to be connectors, because a wrapper can be new code inserted between component interfaces or infrastructure support between application level code and communication mechanisms. Wrappers are used to compensate for the mismatch problems of interacting, heterogeneous components by altering the perceived behavior of the component with respect to the other components in the system [15].

Reconfigurable architecture research focuses on placing systems in specific architectural styles, e.g., C2 [9] and Pit [11], that research has shown exhibit desired behaviors. “Lateral welding” and “horizontally slicing” are two research techniques that enable the use of middleware in the context of an architectural style. Both of these techniques consist of implementing a single conceptual software connector, named a “virtual connector,” using two or more actual connectors, that are linked across process or network boundaries through a given middleware technology [9]. When implementing an application integration, generally developers will not adhere to any style in particular, with the exception of one enforced by their chosen (or required) middleware product. Thus, forcing a particular application into a new integration style is often problematic.

3 Example Application Integration: ANUBIS

A Network Utility for Book Information Searching (ANUBIS) is a multi-user personal librarian system developed in-house. ANUBIS resolves ISBNs, organizes, catalogs and recommends books or other similar media for a distributed base of users. It is implemented using: *IBM WebSphere MQ v5.3* (middleware), *MySQL Production Release v4.0.12* (database), and *Amazon.com Web Services 2.0* (online data source). ANUBIS has key characteristics that are present in application integrations and can affect application migration: using COTS products, obtaining web services, enabling distributed communication across middleware, and using a custom GUI for the overall system.

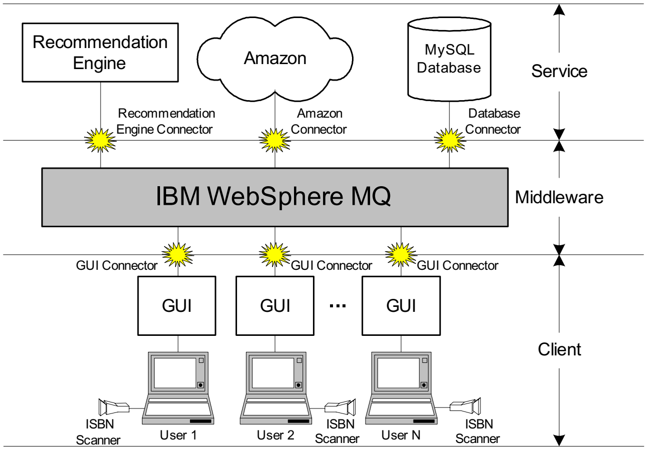


Fig. 1. ANUBIS Architecture

Using a bar code scanner (or keyboard), a user enters an ISBN. The media information for the ISBN is retrieved from the online data source. This information is then stored in the user's personal space within the database. Recommendations are made based on user information (i.e., stored media) and preferences to suggest media items that may interest the user using Amazon.com and the Recommendation Engine. Figure 1 depicts the implemented system, in which a recommendation engine, data source, and database provide services to each user (client), utilizing the middleware to enable the connections.

The system is implemented using two different architecture styles for middleware: request/reply and publish/subscribe. This allows for a straightforward comparison of connector and middleware complexity resulting from the two implementation styles. We use ANUBIS to demonstrate the change in connectors as middleware is migrated from one style to another, and the trade-offs that must be negotiated within this migration.

3.1 IBM WebSphere MQ – The Middleware

WebSphere® MQ is a queue-based messaging middleware wherein a transaction requires a producer to put a message on a queue and a consumer to pull it off. The system implementation involves components connecting to the middleware framework using MQClient libraries (the connectors in Figure 1). These are the Application Programming Interface (API) for the request/reply style and the Application Messaging Interface (AMI) for the publish/subscribe style.

Figure 2 depicts the two WebSphere MQ paradigms. Both utilize the MQ Queue Manager which provides expected communication services, such as location transparency and guaranteed message delivery. For the request/reply, only the queue manager is needed to govern the connections to the specific queues. However, the publish/subscribe style utilizes an additional service that overlays the MQ Queue Manager to provide subscription management. Since WebSphere MQ is inherently message queue based, this management takes the form of propagation functionality to ensure that a copy of each message is available for each component with a corresponding subscription.

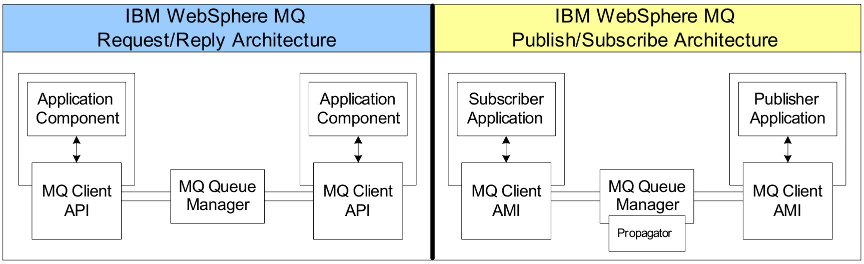


Fig. 2. Paradigm Dependent Middleware

3.2 ANUBIS Request/Reply Implementation

The overall behavior (as perceived by the end user) is identical across the styles. Only the connectors are impacted by the migration. Therefore, we will only discuss the distinction between WebSphere MQ request/reply and publish/subscribe within ANUBIS as it relates to how the connectors interact with each other and the middleware. Since ANUBIS is event driven, the most natural implementation is request/reply. This implementation uses at least two queues for each request/reply action, i.e., one for each service request and one for each reply. Since many components perform several services, many queues are required. For example, in Figure 1, the multi-threaded *Database Connector* watches seven incoming queues for service requests and puts replies on one of seven outgoing queues depending on the recipient and service performed.

Figure 3 depicts the typical interaction between a *Client Connector and Service Connector* within ANUBIS using request/reply through WebSphere MQ. A service is any component that provides functionality to other components. The *Service Connector* bridges the gap between *WebSphere® MQ* and the service using their respective APIs. The flow in Figure 3 is as follows. The *Client Connector* puts a message on a queue (*serviceRequestQueue*) and blocks on the *serviceReplyQueue*. The request queue is read exclusively by the *Service Connector* which pulls the message off of the queue, calls the service, forms an appropriate reply message based on the service response, and puts it on the *serviceReplyQueue*.

For performance reasons the system is implemented so the users share reply queues, requiring users to browse reply queues for messages where their user ID matches the message's *correlationID* field. Once the *Client Connector* has the message, it forwards the message payload to the user. Though not difficult to program, the rigidity of having pre-defined queues severely reduces the flexibility and dynamism of the architecture. For instance, the control logic that manages the queues is centrally housed in the middleware. In the event that the system must dynamically add/upgrade components or upgrade the middleware, even a minor change can have severe impacts.

3.3 ANUBIS Publish/Subscribe Implementation

In publish/subscribe, instead of having dedicated queues for each service request, (or multiple queues if a destination performs multiple services) there is simply one queue, which serves as the message bus. Therefore, topics are used to differentiate the messages. For example, a component wishing to receive messages of a certain subject would subscribe to the associated message topic (and conversely for publication). In Figure 4, the *Service Connector* subscribes to a topic (in this case *serviceRequest*) and blocks until a message of that topic is available. The *Client Connector* publishes a message to the *serviceRequest* topic and blocks on the topic of the reply (in this case a subtopic of *Users* that corresponds to the user ID of the client). The MQ Queue Manager creates copies of messages for all subscribed components, i.e., the number of copies equals the number of subscriptions, and notifies the connectors.

The publish/subscribe paradigm facilitates dynamism and evolution. This is mainly due to having the responsibility of the routing logic and brokering contained within the component connectors, i.e., the connectors contain the topic information relevant to the particular component. This relieves the computational burden from the middleware. Additionally, the action of publishing or subscribing can be done at run-time, creating more "intelligent" connectors that allow a more flexible control/data topology.

4 Migrating to More Adaptability

In this section, we discuss application integration migration from three viewpoints. The first viewpoint is the middleware, i.e., how does the functionality offered by the new middleware match that offered by the previous middleware. The second viewpoint is from the architectural style viewpoint. As some systems are best implemented in a certain style, it is important to consider the optimal architecture, based on design, implementation, security, and dynamism. The third viewpoint considers the ANUBIS connectors, and the impact of moving more responsibility for integration to the component endpoints.

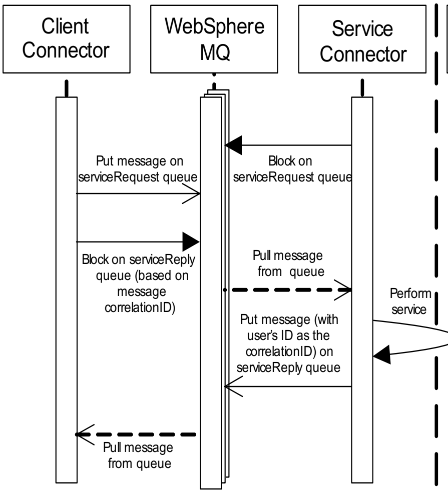

Client Connector
WebSphere MQ
Service Connector
Put message on serviceRequest queue
Block on serviceRequest queue
Block on serviceReply queue (based on message correlationID)
Pull message from queue
Perform service
Put message (with user's ID as the correlationID) on serviceReply queue
Pull message from queue

Fig. 3. Request/Reply Sequence

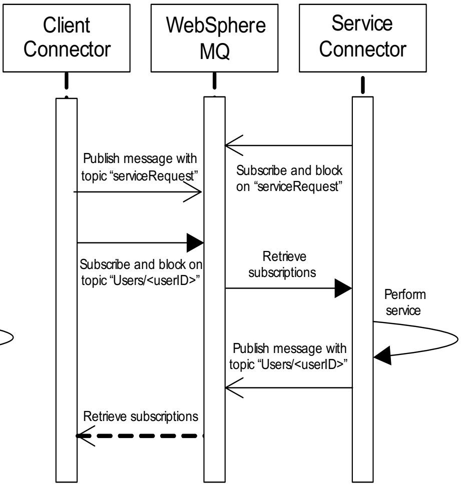

Client Connector
WebSphere MQ
Service Connector
Publish message with topic "serviceRequest"
Subscribe and block on "serviceRequest"
Subscribe and block on topic "Users/<userID>"
Retrieve subscriptions
Perform service
Publish message with topic "Users/<userID>"
Retrieve subscriptions

Fig. 4. Publish/Subscribe Sequence

4.1 Middleware Migration

In the request/reply implementation, WebSphere® MQ maintains a strong implicit control over message passing because it relies on static queues with predefined endpoints to transmit messages. This is implemented as a static router found within the Queue Manager in the request/reply architecture in Figure 2. Although, publish/subscribe lacks the direct, static routing by the multiple queues, it uses a *Propagator* (see Figure 2). The Propagator performs a similar function (matching requests to replies), except that it matches subscriptions to publications dynamically. It does this by managing subscription information (and monitoring the run-time creation and deletion of subscriptions) to ensure a copy of each published message exists for each subscribed component. This extends the scope of messages in the system, allowing dynamic multi-recipient communications.

The publish/subscribe implementation has less implicit control in a central location because it maintains only one message bus on which all types of messages (corresponding to subscribed topics) are passed. Therefore, brokers on the component connector, external to the middleware, determine what action to take based on the topic of the arriving messages (since they may subscribe to many topics). Migrating to the WebSphere® MQ publish/subscribe implementation does not require the complete replacement of connector functionality in the request/reply implementation. For instance, a buffer is carried over that serves as a storage facility for messages as they are passed through the system.

4.2 Architectural Style Migration

Since different architecture styles have different data and control flow expectations, it logically follows that there will be design trade-offs. Given that we migrate ANUBIS from a hierarchical style (request/reply) to an event-based style (publish/subscribe), we assess trade-offs directly from these two implementations. We specifically examine design complexity and change management.

Design. The design of a request/reply architecture is fairly simple. However, for a large system, the complexity can be quite overwhelming. There may be multiple, directed communication paths between interacting components, e.g., in ANUBIS each service requires a queue. Thus, new integration functionality focuses on managing and monitoring these queues, as well as the information that is on the queues. As the system evolves, this management can become cumbersome and error-prone.

Conversely, in publish/subscribe, the design focuses more upon what information a component provides *to* the system and what information it requires *from* the system. Thus, the information alone is controlled and monitored. The burden for assuring the component receives the proper information is the responsibility of the component. Because this is the main consideration required to enable proper communication, evolution and component upgrade are more locally managed.

Change Management. Within request/reply, the insertion and modification of components requires the middleware to be internally modified by adding the necessary queues. The connector also has to be modified to communicate directly to existing and new queues. This requires substantial static information that is difficult to use in a dynamic setting.

The publish/subscribe paradigm facilitates dynamism and change management quite well, because most of the routing logic and brokering can be contained within the connector. Within the publish/subscribe paradigm, contributing components can register with the system when they are available to participate. Of course, there is a semantic restriction that the component can utilize the available topics. The amount of allowable dynamism is dependent on where the responsibility of brokering the messages for the components lies. If it is local to the component's connector, then a high degree of dynamism is possible. However, if the middleware continues to maintain the same amount of control as in the request/reply paradigm, in which it performs all of the message management and directed delivery, the potential for dynamism is reduced.

4.3 ANUBIS Connector Migration

We identify four generic factors that are affected by the connector migration in ANUBIS. The first is the change in the client interface offered by the middleware given a particular implementation framework. The second is how events and message handling are directed and controlled. The third is the role the identity of the component plays in the application integration. The fourth is the manipulation of process control, such as where threading occurs and how it is managed. Though we use ANUBIS as an example, clearly these factors have applicability as localized connectors play a more significant role in application integration.

Interface Migration. The first consideration for system migration is the possibility that the client interface of the middleware may change. In ANUBIS, the request/reply interface uses the IBM Application Programming Interface (API), while publish/subscribe requires the Application Messaging Interface (AMI). With the migration, is the need to create a new translator in the component connector to convert transmitted objects into raw bytes as accepted by the AMI.

Control Migration. In request/reply, each service within a component has a dedicated, typed queue that anticipates specific requests. No logic is needed to determine what operation to perform because it explicitly correlates to the request queue. The transition from directed queues and their central control to a more flexible implementation requires the component connectors to shoulder more functional responsibility for controlling messages of interest. The increased functionality due to new brokers on the component connector leads to a greater flexibility for the application integration because these brokers can be more easily updated than the centralized integration functionality.

Identity Migration. Middleware style migration causes the component identity to change priority. This is due to the identity being highly dependent on the location of message and event control within the system. With the centralized control of request/reply, comes a lessening for the need for component identity, and for the connector to control its use. The centralized solution manages information direction, and predefines the destinations based on the expected purpose (or use) of the particular queues – which are tied directly to component identity.

By making the connectors more responsible for the directed communication, we separate the concerns of identity distinction from topic distinction, allowing the publish/subscribe connectors to know *who* they are as well as *what* they are interested in. This functionality is split between extending the integration functionality which handles the middleware connection to allow a subscription/publication interaction with the system, and the routing integration functionality which handles message dissemination of incoming messages based on the topic. Thus, when connectors receive messages, they should trigger the appropriate functionality within the component.

Process Migration. Though the connectors are multi-threaded in the request/reply implementation, it is the middleware that actually controls the threading. Because MQ allows threads to block (wait for messages) on queues, it makes sense to have one thread per queue rather than to wastefully have a dispatcher thread poll all the queues. The migration to publish/subscribe removes this control. However, multi-threading capability can still be maintained without decreasing performance if the local control over threading is increased. Publish/subscribe connectors utilize a broker to handle thread-based dispatching of messages using as a basis the topics the components request. In fact, it may be necessary to add brokering functionality beyond the thread controller, depending upon how the topic hierarchies are implemented. For instance, it may have multiple, outstanding subscriptions to deal with or only one subscription at a time.

5 Conclusion

Many application integrations face migration to new middleware, the incorporation of new software components, and the new demarcation of existing components (such as in service-oriented architectures). Furthermore, there is a need for more dynamism in application integration, in which components provide on-demand services. Proper design decisions made at the time of migration – such as where the seat of integration control should reside – can make the difference in whether the application will be allowed to continually evolve or stagnate in a certain implementation style.

In this paper, we examine the impact of moving from one middleware implementation style to another using the same product. We control the analysis by working with exactly the same components. We focus on the differences between a centralized integration solution in which component connectors have minimal functionality, because middleware retains the majority of the functionality, and a localized integration solution, in which component connectors take the major responsibility for making the component participate in the interaction. In general, we find that localized integration is more difficult to design and implement. This occurs as it is necessary to cover many scenarios and decompose a central solution into its smaller, local parts. However, it offers the high-degree of flexibility needed for application integrations to evolve, upgrade, and allow for the dynamic insertion and deletion of components.

Acknowledgements. This material is based upon work supported in part by AFOSR (F49620-98-1-0217) and NSF (CCR-9988320).

References

1. Websphere MQ Application Programming Guide. IBM, 3, October (2002)
2. Abate, P. Bernardo, M.: A Scalable Approach to the Design of Sw Architectures with Dynamically Created/Destroyed Components. Int'l Conf. on Software Engineering and Knowledge Engineering. Ischia, Italy (2002)
3. Allen, R., Douence, R. Garlan, D.: Specifying and Analyzing Dynamic Software Architectures. Conf. of Fundamental Approaches to Software Engineering. Lisbon, Portugal (1998)
4. Davis, L. Gamble, R. F.: Identifying Evolvability for Integration. 1st Int'l ICCBSS. Orlando, Florida (2002)
5. Davis, L., Payton, J. Gamble, R.: Toward Identifying the Impact of Cots Evolution on Integrated Systems. 2nd Int'l Workshop on the Successful Development of COTS. Limerick, Ireland (2000)
6. Hasler, K., Gambler, R., Frasier, K. Stiger, P.: Exploiting Inheritance in Modeling Architecture Abstractions. (1999)
7. Keshav, R. Gamble, R.: Towards a Taxonomy of Architecture Integration Strategies. ISAW - 3. (1998)
8. Magee, J. Kramer, J.: Dynamic Structure in Software Architecture. FSE - 4. San Francisco, CA (1996) 3-14
9. Medvidovic, N.: On the Role of Middleware in Architecture-Based Software Development. 14th Int'l SEKE. Ischia, Italy (2002) 299 - 306
10. Mehta, N., Medvidovic, N. Phadke, S.: Towards a Taxonomy of Software Connectors. 22nd Int'l Conf. on Software Engineering. (2000)
11. Mikic-Rakic, M., Mehta, N. Medvidovic, N.: Architectural Style Requirements for Self-Healing Systems. 1st Workshop on Self-Healing Systems (WOSS02). Charleston, SC November 18-19 (2002) 49-54
12. Oreizy, P., Medvidovic, N. Taylor, R.: Architecture-Based Runtime Software Evolution. 20th Int'l Conf. on Software Engineering. Kyoto, Japan (1998) 177-186

13. Payton, J., Gamble, R., Kimsen, S. Davis, L.: The Opportunity for Formal Models of Integration. 2nd Int'l Conf. on Information Reuse and Integration. (2000)
14. Spitznagel, B. Garlan, D.: A Compositional Approach for Constructing Connectors. WICSA'01. Amsterdam, the Netherlands (2001) 28-31
15. Spitznagel, B. Garlan, D.: A Compositional Formalization of Connector Wrappers. 25th Int'l Conf. on Software Engineering. Portland, OR (2003) 374-384
16. van der Hoek, A., Mikic-Rakic, M., Roshandel, R. Medvidovic, N.: Taming Architectural Evolution. ESEC/FSE - 9. Vienna, Austria (2001)

Web-Based COTS Component Evaluation

Franck Barbier

LIUPPA, Université de Pau
BP 1155
64013 Pau CEDEX, France
Franck.Barbier@univ-pau.fr

Abstract. The deep nature of COTS components is that they are shut software units for end users. In the process of reuse, test is a natural and primary step for obtaining information on component capabilities. Such an assessment and evaluation is however restricted to the single access and use of interfaces. To attenuate this, Built-In Test (BIT) components are introduced. These components enhance the means by which COTS components may be, in a broad sense, acquired from Internet. In this paper, we develop two strategies. Firstly, distinct BIT versions of original components are implemented. Secondly, BIT components are built from scratch in order to, for vendors, supply components that have customizable and innovative test and configuration features. In both cases, the assistance of a code generator allows the introduction of extra properties for viewing and handling BIT components in Web browsers. Trust results from this approach that demonstrates the necessity for supporting and instrumenting component procurement in general. A "true" component marketplace on Internet becomes then conceivable.

1 Introduction

Are COTS components somewhere else that on the Web? Reasonably they are not. There is however nowadays no formal organization of a component marketplace on Internet. Looking at featured websites, components are eclectic in forms (e.g. size, implementation language) and in natures (e.g. application domains, component infrastructure conformance). There are especially no relevant component repositories based on a unanimously agreed typology, no consensual prediction-enabled component specification language, no intelligent acquisition support that goes beyond the simple fact of searching (with currently strong limitations) and downloading components. Facing up such an odd and unstructured provisioning environment, a key point is the possibility to rigorously assess and evaluate components in order to make potential reusers confident with the products they intend to buy, lease or freely incorporate into their applications. In the absence of appropriate procurement techniques and tools, COTS component reuse rates remain low compared to all of the available software material.

In this paper, we advocate the delivering of BIT components that are components plus BIT code. We view this extra material as a means for functional and non functional component characteristic checking, and thus a step towards trustable components. Since the BIT technology has been presented in detail in [1], we here stress the

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 104–116, 2004.

Web-based manipulation of BIT components. Indeed, such components are activable, and thus testable on-site or remotely, by means of a Web browser. Furthermore, in addition to a testing interface, they also offer a configuration interface that permits (re)-configuration at runtime. A special thought occurs in the paper about the fact that COTS components are outsourced entities, and thus, have to carefully fulfill *local* runtime and deployment conditions [2].

We show in the paper that we together address prefabricated components and, from a vendor's point of view, the building of BIT-specific COTS components that aim to be offered on Internet. For pedagogical reasons, the technique and its associated tool are illustrated by means of the *java.util.Stack* elementary component. Since components are most of the time bigger, we also concisely discuss why our contribution is applicable for sizeable components that have numerous states, complex relationships between these states and elaborate request processing.

In Section 105, we walk through the issues relating to component acquisition, covering the procurement process and the pertinent idea of trusted component. Section 0 exposes our technical framework, namely how BIT components are constructed: A design technique with two alternatives and a CASE tool for reducing boring and recurrent tasks. In this section, we cope with the simplistic *Stack* component, showing how it is handled within a Web browser. A complementary discussion on bigger components with an example closes the paper.

2 Component Provisioning

The rapid growth of a COTS component market relies on an adequate spreading medium which is and will undoubtedly be Internet. In this section, we characterize what are the essential consumer expectations with regard to a component market in which the idea of "trusting components" might become concrete. We on purpose lay down the bases of a simple, rational and efficient procurement process.

2.1 Component Market, the Missing Link

Bass *et al.* in [3] analyze from a business perspective the existence and the development of a component market. They especially underline the idea of "brokerage market" in which companies provide software engineers with mechanisms for searching, selecting and globally acquiring components.

Today's practice and experience however show that component websites are not in general really endowed with appropriate tools as for instance dedicated component search engines or component comparison tools. Regarding the CBSE literature, component procurement frameworks, or better, component procurement methods, appear [4], [5], especially those focusing on selection in the spirit of requirements engineering [6]. Nevertheless, an important lack of component assessment and evaluation techniques and tools persists, notably assembly evaluation methods [2].

Components on the Web have various forms and natures. Open source software leads to access and reuse not only interfaces but insides of components. Otherwise, components can have binary or runtime forms (*.jar* in the Java world for instance), can be documented with well-known specification languages, can conform to infrastructure standards as CORBA or EJB. Components may also be supplied with gradual certification degrees that surely enhance their quality in general. In this context, the procurement process is more or less safe. For several reasons based on common sense, a very fine-grained component as the *java.util.Stack* coming from the Java API immediately creates confidence and no reuse barriers slow down its use. In contrast, sizeable components, domain components in transportation (e.g. air traffic control), in finance (e.g. General Ledger) or other domains that come from anonymous companies generate suspicion.

We then believe that the worldwide component market will reach an acceptable level of maturity and efficiency as soon as software engineers will find techniques and tools that will, in the worst case, attenuate and, in the best case, nullify such a suspicion.

2.2 Trustable Components

The simple idea here is that components are, in a pejorative sense, foreign entities. In other words, there is therefore a non negligible risk in incorporating them into our own applications. As claimed by Bader *et al.* in [7], trust is inversely proportional to risk. In that scope, they explain why there is a need for trust, and, on purpose, sketch a wrapping mechanism that equip components with usage contracts that are the support for an agreement between vendors and consumers. In the same spirit, we propose in [1] components with BIT functionality in order to instrument the way by which components may be tested, and thus declared appropriate, with respect to *in-situ* exploitation. This means that deployment contexts and customer-based specific usages may make a component unsatisfactory while, at development time or simply, for vendors, the same component may give a good level of satisfaction. Quality attributes are in fact relative in the sense that builders and reusers manipulate components under different angles.

The need for an assessment and evaluation method for systemically establishing a quantitative and/or qualitative trust degree is then obvious.

A component is trustworthy if and only if:

- It fulfills, in terms of computation and control, all the expected functional requirements. The use of its interfaces leads to an observable and intelligible behavior that in particular allows comparisons with competing components as well as upgrades in order to manage application evolution;
- It enables assembly assessment and evaluation based on collaboration/interaction behavior prediction. Components may be individually "correct" by construction while they may fail at integration time since there are no combination patterns including the said component, that readily and efficiently meet requirements;
- It fits quality of service expectations (reliability, performance, resource burden…) and more generally non functional requirements (e.g. security, norm respect). This level of quality of service together hinges on individuals and assemblies.

Trust stems from the fact that component services are necessary and sufficient for implementing all of the desired functions of software applications, but also from the fact that components are compositional in all other behaviors. Specifically, they may be integrated and still operate the same way in any environment, QoS features being manageable and above all controllable without high difficulty.

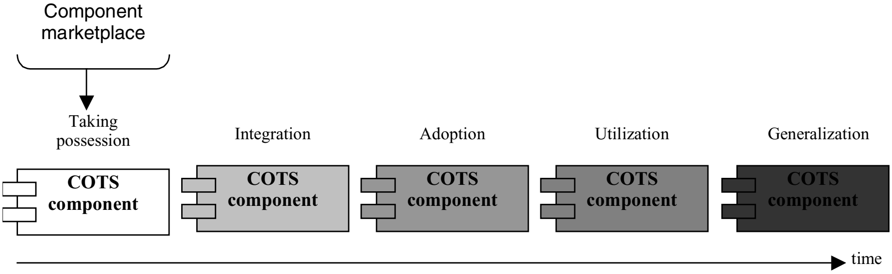


Fig. 1. Component appropriation cycle.

Test is the more naive and recognized mechanism for establishing trust. Contracts embodied by assertions in the Eiffel programming language [8] are greatly studied by the CBSE research community. COTS component are however in essence closed software units due to encapsulation and thus, black-box testing cannot be the panacea [9]. Trust implies the acceptance for suppliers, that purchasers benefit from a more or less important access/view relating to the inside of components. Limits are nevertheless strongly required: For instance, the respect of intellectual property, avoiding security violation or any misuse through direct alteration of private/protected properties.

In this paper, we present new insights into the approach presented in [1]. Making components trustable means the development of special *in-situ* test scenarios for reusers in order to maximize information on components. Their choice spectrum becomes then larger due to the increasing of all checking possibilities.

2.3 Acquisition Process

A picture of a synthetic procurement process with five phases appears in Fig. 1.

Taking possession covers component discovering, downloading, up to individual evaluation. Integration corresponds to any evaluation of collaboration potentiality with regard to in-house components or other external components (same supplier, distinct supplier) that are already adopted and thus used in applications. Adoption is the act of qualification, even certification when components are incorporated into highly critical systems, and leads to resultant actions such as leasing or buying. Utilization precedes generalization and minimizes dependency. In other words, users stress the selection of units of functionality but also pay attention to component substitutability in anticipating and preparing application implementation alternatives that might, for various reasons, put aside any previously selected components. In contrast, generalization precludes for inversing choices in the sense that the retained COTS components act as essential blocks in end users products.

The key issue of a Web-based component market is leveraging substitutability. One may expect that that the cycle in Fig. 1. has to be quick as possible. Fig. 2. sketches what should be the daily activity of an average component integrator, dealing with a couple of components (or BIT components) within a full day. At this time, component qualification, from identification to definitive selection is a complicated task. Substitution, mostly resulting from maintenance, is costly since proving that applications are not disturbed, even damaged, due to the replacement of components by others (upgrades or better units of functionality and/or quality of service), is often empirical and indisputably long.

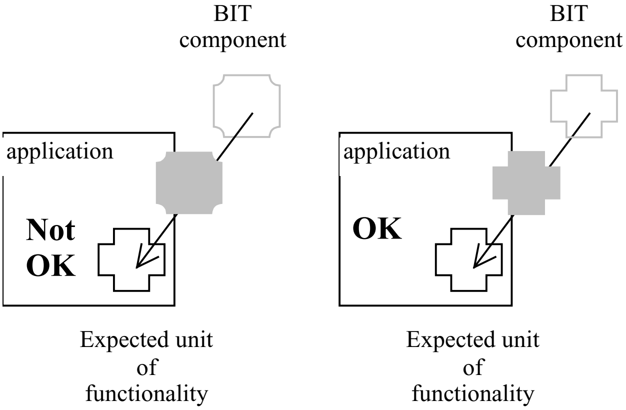

BIT component
BIT component
application
Not OK
application
OK
Expected unit of functionality
Expected unit of functionality

Fig. 2. Elimination (left hand side) versus selection (right hand side) of COTS components.

3 Technical Framework

A BIT component possesses a provided (a.k.a. functional) interface and a required interface (Fig. 3.) as stated in [10]. We add a testing interface and a configuration interface.

In Fig. 3., the purpose of the testing interface is to essentially recast the provided interface, to possibly make accessible components inside. Recasting means that the testing interface includes the same set of services as the provided interface but implementation varies for testing reasons. For instance, creating new variables in order to trace some paths of execution. The testing interface may also offer new operations to make visible hidden properties (in read and/or write modes), to establish *atomic* sequences of service calls and so on. As for the configuration interface, it is most of the time correlated to deployment and is rarely used within client requests. We consider a general-purpose *Management* entity in any distributed software system that operates as a preferential client of the configuration interface. In our approach, (re)-configuration means putting a server component into a defined state, to invoke one or more of its services, and then to verify that the returned result and final state are mutually consistent before accepting the component global delivery as “correct” [11]. Such actions are typically performed when a system of components is (re)-configured, and usually occur infrequently at one or two well-defined moments during a system’s running lifetime.

As further detailed below, we single out Harel’s Statecharts [12] as a backbone for component individual behavioral representations and for contract expressions, contracts being in essence dedicated to component interaction checking. Note that the

Statecharts modeling technique is part of the core of the Unified Modeling Language or UML [13], itself widely used in the world of CBSE for modeling components [14].

COTS components that pervade on the Web have however not all of the interface types appearing in Fig. 3. We thus imagine two different cases. The first one (Section 0) is a kind of reengineering in which a COTS component is transformed into a BIT component having by definition the material in Fig. 3. In the second case, BIT components are completely new software modules that comply with the BIT technology. Instead of building ordinary components, vendors may directly provide BIT components. In Section 0, we illustrate such a case by means of an Automated Teller Machine or ATM component in the finance domain. We expose the benefits of such an approach and raise some inherent drawbacks in conclusion.

We here emphasize a Java implementation of our technology, and more precisely we rely on the reflection capabilities of Java (*java.lang.reflect* package). Most of the component models, especially JavaBeans and EJB, are also built on the top of this package. This hopefully makes our approach credible. For Java COTS components coming from everywhere on Internet, we exercise introspection in order to automatically generate skeletons of BIT components. Code is augmented according to the technique described in Section 0. All of the approach depends upon the Java Management Extensions or JMX sub-library [15] that generalizes the notion of manageable component or *manageability* including that of configuration.

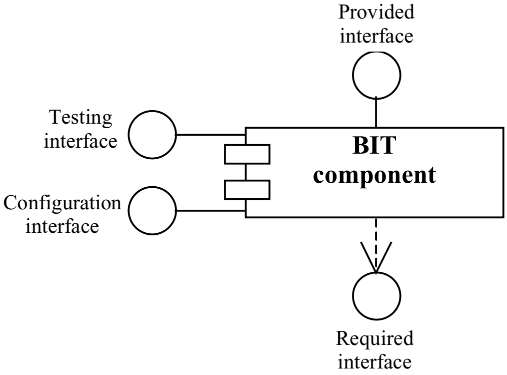


Fig. 3. Canonical organization (UML formalism) of a BIT component.

3.1 From a Component to a BIT Component

In the Java scope, a BIT class can be automatically generated by means of the BIT/J CASE tool [16] under the conditions that the original class is accessible through its source code (*.java*) *or* its "binary" form (*.class* or *.jar*) *or* is running and is attainable from a Java virtual machine, possibly remotely and under well-formalized security conditions. An interesting aspect of the *java.util.Stack* class is that it is pedagogical: That is a "true" COTS component in the sense that we did not develop it. Moreover, our technique goes beyond the simple fact of making testable all of the provided services of a class. As illustrated in Sections 0 and 0, we try, when possible, to "re-format" the inside of a component in order to have a more comprehensible view, allowing complex contract expressions at integration time especially.

3.1.1 Reengineering

In Fig. 4., an transformation path with optional directions is sketched for a COTS component.

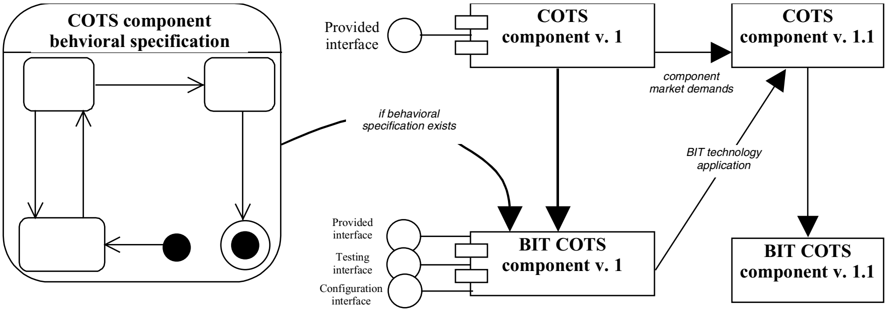


Fig. 4. BIT technology based on reengineering.

In Fig. 4., an assumption is made that COTS component providers have accurate behavioral specifications of their products (in our case Statecharts) or purchasers have to create these dynamic models, possibly with the help of designers. COTS component dynamic models such as that on the left hand side of Fig. 4. nowadays rarely exist, even if UML-based component testing becomes common [17]. Note that the BIT technology may nevertheless work *without* behavioral specifications of components. We however advocate the building of Statecharts to gain all of the advantages of BIT.

So, in the absence of specification, an abstruse set of services making up a COTS component's provided interface exists. Without formal intelligible models, the functionality offered by components is incomprehensible. Reengineering thus means, in most cases, the postponed modeling of component behaviors. This global task is sometimes unrealistic, even impossible. This may simply be caused by complexity, or the fact that components match with difficulty to state machines or to any similar formalism. In Fig. 4., starting from a COTS component v. 1, we construct a BIT COTS component v. 1. Next, new releases appear either due to market demands or BIT technology application.

3.1.2 Simple Component

Applying BIT to classical Java classes, leads to models as that in Fig. 5. This case study is such that the initial component (the Java *Stack* predefined class) has no state machine: It was *not* originally organized and delivered based on a comprehensive specification. We may thus abstract observable states according to what we want to test. In Fig. 5., for instance, one does not deal with a "Full" state since *Stack* does *a priori* not reach such a state. In contrast one distinguishes two sub-states belonging to a *Not empty* state. We have also an *Empty* state. Finally, transitions are that of the provided interface of *Stack*.

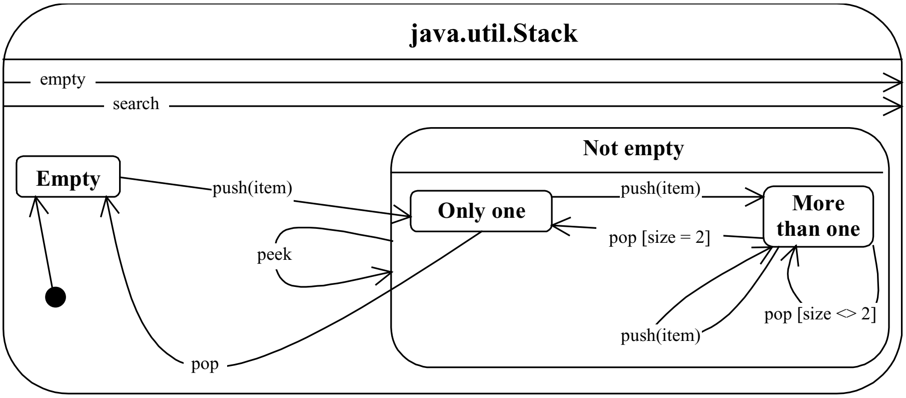
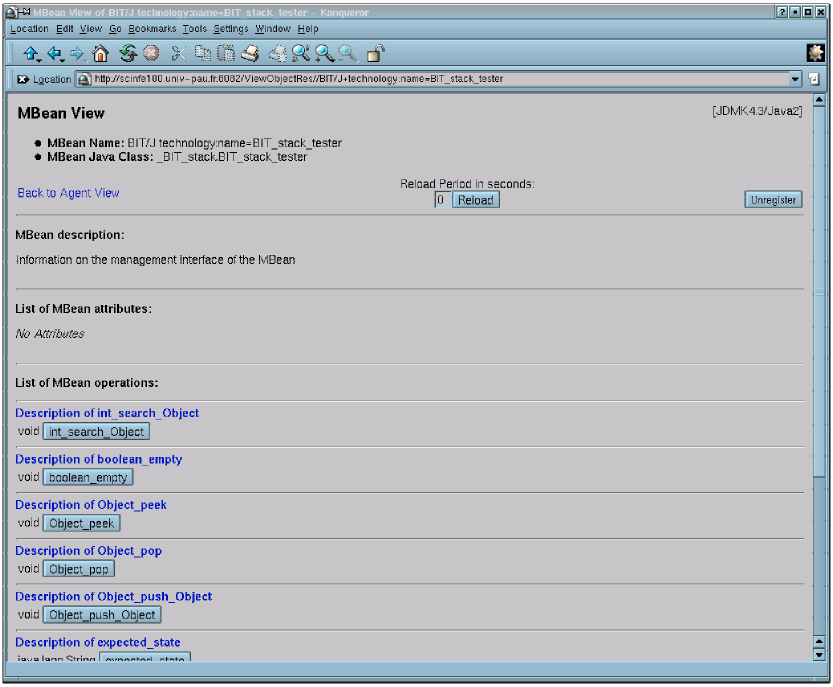


Fig. 5. Reengineering of the Java *Stack* COTS component.

We do not in this paper enter into much detail concerning the precise implementation technique of a BIT component since it is already published in [1] and [16]. The final result appears in Fig. 6. The operations signatures are transformed by the CASE tool as follows: *public Object push(Object item)* becomes *public void Object_push_Object()* in the BIT component tester which is the MBean (standing for Manageable Bean in JMX) viewed in the browser in Fig. 6. As explained in [1], the code in the *public Object push(Object item) throws Statechart_exception* method of the BIT component sustains large changes that fit the state machine in Fig. 5. as follows (case of the *push* function):

```
Object result = _stack.push(item);

_BIT_stack.fires(_Empty,_Only_one);

_BIT_stack.fires(_Only_one,_More_than_one);

_BIT_stack.fires(_More_than_one,_More_than_one);

_BIT_stack.used_up(); // state machine execution

return result;
```

Finally, JMX facilitates the remote execution of MBeans. Hence, clicking on buttons runs services in the browser of Fig. 6. (by lack of space, configuration services are hidden at the bottom of the window). Five facilities (see below) are always visible as buttons, and above all useful and interesting each time after a service of the testable interface is run. These facilities mainly serve as “trackers” of component behaviors:


MBean View of BIT/J technology:name=BIT_stack_tester - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Location
Reload
Unregister

http://scinfe100.univ-pau.fr:8082/ViewObjectRes//BIT/J+technology:name=BIT_stack_tester

MBean View [JDMK4.3/Java2]

- **MBean Name:** BIT/J technology:name=BIT_stack_tester
- **MBean Java Class:** _BIT_stack.BIT_stack_tester

Back to Agent View

Reload Period in seconds: 0

MBean description:

Information on the management interface of the MBean

List of MBean attributes:

No Attributes

List of MBean operations:

Description of int_search_Object
void int_search_Object

Description of boolean_empty
void boolean_empty

Description of Object_peek
void Object_peek

Description of Object_pop
void Object_pop

Description of Object_push_Object
void Object_push_Object

Description of expected_state
java.lang.String expected_state

Fig. 6. Web-based COTS assessment and evaluation.

```
public String interpretation()

public String expected_result()

public String result()

public String current_state()

public String expected_state()
```

(Re)-configuration may occur within services (next section) or via the browser. For instance, a special action that reinitializes the BIT stack state machine may be defined as follows:

```
public String reset()
```

It simply put the BIT stack in the *Empty* state.

3.2 New Technique for COTS Component Design

COTS component vendors have opportunity in hearing from and exploiting customers feedbacks. This is surely true when we consider component repairing. Equipping components with new functionality cannot however, in essence, be peculiar to customer needs, with the exception of large adhesion, which, from a marketing point of view, justifies customer-oriented evolution. How then to foster such a business interaction? How to capitalize on users experience? We here illustrate such perspectives in relation with large-scale domain-centric components, which are, from their birth to their death, built on the top of the BIT technology.

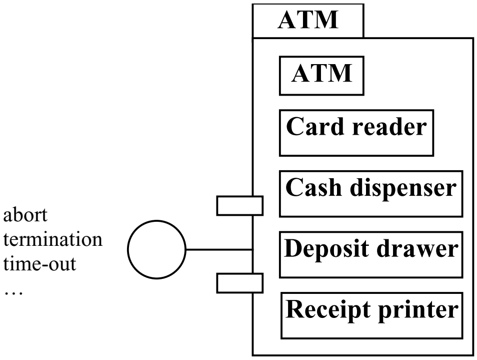


Fig. 7. ATM component.

Fig. 7. is an ATM component made up several common subparts: A card reader, a cash dispenser, a receipt printer, a deposit drawer and an ATM subcomponent embodying control and coordination services for all of the devices. A large consistent collaborative statechart may be offered in order to formalize how such an aggregate component may function. In [18], we give an example and explain how to combine components based on Statecharts as well as on the principle of full encapsulation: Subparts are unshared and participate in the implementation of the whole they belong to. Having then state machines of subcomponents and aggregates, the building and the delivering of a BIT COTS ATM component is immediate. We in fact strongly believe that providing executable images of behavioral models, as well as their systematic access through the Web, is a significant step in the search of high-confidence components. There is indeed now, no way for finding an easily understandable ATM component on Internet while it is recognized as a central element in many banking applications.

Otherwise, sophisticated implementation of control features are possible, even relevant. For instance, the Web browser will make visible the *abort* service that in particular checks a contract relating to the card reader. This methods tries a risky (re)-configuration that simulates a kind of re-initialization for the ATM: Going towards the *Start* state that enables a transaction start even though the card reader may not have recovered a consistent state. An interesting point here is that the re-configuration can be, instead, externalized (button(s) in the Web browser) allowing more generally many qualitative measurements of the tested aggregate component, but also subparts if they have been built with BIT.

```
public void abort() throws Statechart_exception {
   try {// event processing here based on the BIT/J
        // special statechart management mechanism,
        // see [1] or [16] for more details
      card_reader().card_to_be_ejected();
      …
      // post_condition
      check(card_reader().in_state(“Ejecting”);
      // impose that class Card_reader is built with
      // BIT
   }
   catch(BIT_contract_violation_exception bcve) {
      try {// possible dynamic re-configuration
         to_state("Start");
         // possible subpart re-configurations here:
         …
      }
      catch(BIT_state_exception bse) { … }
      catch(BIT_state_monitor_exception bsme) { … }
   }
}
```

4 Conclusion

The intuitive and next deep understanding of COTS components is recognized as a key challenge in the future since COTS components are external entities whose first uses raise a lot of distrust. Although, the Web is certainly one of the best distribution media, no advanced thought really occurs about how COTS components may be acquired from this medium. The idea is to have integrated test code in components that favors any postponed and/or *in-situ* assessment and evaluation. Furthermore, BIT components have a testable and configuration interfaces. Each service of these two interfaces is accessible and activable via a graphical user interface allowing genuine learning, qualitative measurement, and more generally an appropriate mechanism for leveraging trust. Information on component behaviors may in particular be captured based on the fact that Harel's Statecharts are used to obtain executable component specifications.

A drawback is however the fact that the BIT technology may discourage its use for component builders in the sense that there is a risk of losing their know-how, and more practically their intellectual material.

The overall paper's contribution is a support for component procurement on the Web. Regarding the technical approach, we stress Java components that, in using the reflection power of Java, are remodeled. More precisely, existing components are replaced by their BIT incarnations whose code is generated via a CASE tool. As for new components, they may be constructed by means of the BIT technology if vendors want to offer to potential customers, a new kind of safe and reliable way of acquisition.

A main perspective of the work presented in this paper is to go on studying sizeable COTS components. We need aggregate components used by clients components. Parts of these aggregates may have or may not have BIT material. In any case, testing the whole is rarely equivalent to testing all of its parts individually. We thus need a strategy by which we are able to determine where some BIT code is useful, and consequently, where it is not. This raises another consequential problems that are overheads (in terms of resource burden: BIT code is costly) and the concomitant access through together the "normal" provided *and* the "additional" testable interfaces. We recently observe during experimentations that this concurrent use may create typical perturbations, or errors, while an exclusive use of one of the two interfaces, is totally safe! We are thus currently thinking about an efficient deployment mechanism guarantying that the BIT code is "the best friend" of the functional code, i.e. formal arbitrations occur at runtime so that no failures depend upon BIT itself.

Acknowledgements. This work has been partially funded by the European Union within the Component+ (IST 1999-20162) IST project.

References

1. Barbier, F., Belloir, N., and Bruel, J.-M.: Incorporation of Test Functionality into Software Components, proceedings of The 2nd International Conference on COTS-Based Software Systems, Ottawa, Canada, Lecture Notes in Computer Science #2580, Springer, February 10-12, (2003) 25-35
2. Crnkovic, I., and Larsson, M.: Building Reliable Component-Based Software Systems, Artech House (2002)
3. Bass, L., Buhman, C., Comella-Dorda, S., Long, F., Robert, J., Seacord, R., and Wallnau K.: Volume I: Market Assessment of Component-Based Software Engineering, Carnegie Mellon University, Software Engineering Institute, TECHNICAL REPORT CMU/SEI-2000-TR-008, ESC-TR-2000-007 (2000)
4. Meyers, B., and Oberndorf, P.: Managing Software Acquisition – Open Systems and COTS Products, Addison-Wesley (2001)
5. Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., Muthig, D., Paech, B., Wust, J., and Zettel, J.: Component-Based Product Line Engineering with UML, Addison-Wesley (2002)
6. Maiden, N., Kim, H., and Ncube, C.: Rethinking Process Guidance for Selecting Software Components, proceedings of The 1st International Conference on COTS-Based Software Systems, Orlando, USA, Lecture Notes in Computer Science #2555, Springer, February 4-6, (2002) 151-164
7. Bader, A., Mingins, C., Bennett, D., and Ramakrishan, S.: Establishing Trust in COTS Components, proceedings of The 2nd International Conference on COTS-Based Software Systems, Ottawa, Canada, Lecture Notes in Computer Science #2580, Springer, February 10-12, (2003) 15-24
8. Meyer, B.: Object-Oriented Software Construction, Second Edition, Prentice Hall (1997)
9. Wallnau, K., Hissam, S., and Seacord, R.: Building Systems from Commercial Components, Addison-Wesley (2002)
10. Szyperski, C., Gruntz, D., and Murer, S.: Component Software – Beyond Object-Oriented Programming, Second Edition, Addison-Wesley (2002)
11. Groß, H.-G., Atkinson, C., and Barbier, F.: Component Integration Through Built-In Contract Testing in Component-Based Software Quality: Methods and Techniques, Lecture Notes in Computer Science #2693, Springer (2003)
12. Harel, D.: Statecharts: A Visual Formalism for Complex Systems, Science of Computer Programming, 8, (1987) 231-274
13. Object Management Group: OMG Unified Modeling Language Specification, version 1.5 (2003)
14. Crnkovic, I., Hnich, B., Jonsson, T., and Kiziltan, Z.: Specification, Implementation, and Deployment of Components, Communications of the ACM, 45(10), (2002) 35-40
15. SUN Microsystems: Java Management Extensions, Instrumentation and Agent Specification, version 1.2 (2002)
16. Belloir, N., Bruel, J.-M., Barbier, F.: BIT/J library – user's guide (2003)
17. Wu, Y., Chen, M.-H., and Offutt, J.: UML-Based Integration Testing for Component-Based Software, proceedings of The 2nd International Conference on COTS-Based Software Systems, Ottawa, Canada, Lecture Notes in Computer Science #2580, Springer, February 10-12, (2003) 251-260
18. Barbier, F.: Composability for Software Components: An Approach Based on the Whole-Part Theory, proceedings of The 8th IEEE International Conference on Engineering of Complex Computer Systems, Greenbelt, USA, IEEE Computer Society Press, December 2-4, (2002) 101-106

Software Fault-Tolerance with Off-the-Shelf SQL Servers

P. Popov[1], L. Strigini[1], A. Kostov[2], V. Mollov[2], and D. Selensky[2]

[1] Centre for Software Reliability, City University, London, UK
{ptp,strigini}@csr.city.ac.uk
[2] Department of Computing, Technical University, Plovdiv, Bulgaria
alex@obs.bg,vmollov@yahoo.com,selensky@bigfoot.com

Abstract. With off-the-shelf software, software fault tolerance is almost the only means available for assuring better dependability than the off-the-shelf software offers, without the much higher costs of bespoke development or extra V&V. We report our experience with an experimental setup we have developed with off-the-shelf SQL database servers. First, we describe the use of a protective wrapper to mask the effects of a bug in one of the servers, without depending on an adequate fix from the vendors. We then discuss how to combine the diverse off-the-shelf servers into a diverse modular redundant configuration (N-version software or N-self-checking software). A wrapper guarantees the consistency between the diverse replicas of the database, serving multiple clients, by restricting the concurrency between the client transactions We thus show that diverse modular redundancy with protective wrapping is a viable way of achieving fault-tolerance with even complex off-the-shelf components, like database servers.

1 Introduction

The audience of this conference is well aware of the pros and cons of using off-the-shelf (OTS) software components[1]. In this paper we focus on the dependability problems that OTS components pose to system integrators: their documentation is usually limited to well defined interfaces, and simple example applications demonstrating how the components can be integrated in a system. Component vendors rarely provide information about the quality and V&V procedures used. This creates problems for any integrator with stringent dependability requirements. At least in non-safety critical industry sectors, vendors often treat queries of the quality of the off-the-shelf components as unacceptable or even offensive [1]. System integrators are thus faced

[1] We use the term "components" in the generic engineering meaning of "pieces that are assembled to form a system, and are systems in their own right". "Components" may be anything ranging from software libraries, used to assemble applications, to complete applications that can be used as stand-alone systems. We consider together commercial-off-the-shelf (COTS) and non-commercial off-the-shelf, e.g. open-source, components: the difference is not significant in our discussion. Even when the source code is available, it may be impossible to make use of it – its size and complexity (and often poor documentation) may deny the system integrator the advantages usually taken for granted when the source code is available.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 117–126, 2004.

with the task of building systems out of components which cannot be trusted to be sufficiently dependable for the system's needs, and often are not.

As we argued elsewhere [2] fault-tolerance is often the only viable way of obtaining one's required dependability at the system level, given the use of OTS components. In this common scenario, the alternatives – improving the OTS components, performing additional V&V activities – are either impossible or infeasible without costs comparable to those of bespoke development. This situation may well change in the future, if customers with serious dependability requirements achieve more clout in their dealings with OTS component developers, but this possibility does not help system integrators who are in this kind of situation now.

Fault tolerance may take multiple forms, e.g., additional (possibly purpose-built but relatively simple) components performing protective wrapping, watchdog, monitoring, auditing functions, to detect undesired behaviour of the OTS components, prevent their producing serious consequences, and possibly effecting recovery of the components' states; or even full-fledged replication with diverse versions of the components. Such "diverse modular redundancy" seems desirable because it offers end-to-end protection via a fairly simple architecture, and protection against the identical faults that would be present in replicas within a non-diverse modular-redundant system. The cost of procuring two or even more OTS components (some of which may be free) would still be far less than that of developing one's own.

All these design solutions are well known. The questions, for the developers of a system using OTS components, are about the dependability gains, implementation difficulties and extra costs that they would bring for that specific system.

To study these issues, we have selected a category of widely used, fairly complex OTS components: SQL database servers. Faults in the currently available SQL servers are common. For evidence one can just look at the long list of bug fixes supplied by the vendors with every new release of their products. Further reliability improvement of SQL servers seems only possible if fault-tolerance through design diversity is employed [3]. Given the many available OTS SQL servers and the growing standardisation of their functionality (SQL 92, SQL 99), it seems reasonable to build a fault-tolerant SQL server from available OTS servers. We have developed an experimental testbed which implements a diverse-redundant SQL server by wrapping a redundant set of SQL servers, so that multiple users run their transactions concurrently on the wrapped SQL servers. We are running experiments to determine the dependability gains achieved through fault tolerance [4]. In this paper, we report on experience gained about the design aspects of building fault tolerance with these specific OTS components:

- regarding diverse modular redundancy, we consider *N-version programming* (NVP) and *N-version self-checking programming* (NSCP) – to use the terminology of [5]. In NVP, the system's output is formed by a vote on the replicated outputs. In NSCP, each diverse "version" is supposed to fail cleanly, so that anyone of the replicated outputs can be used as the system's output. Both solutions depend on guaranteed consistency between the states of the diverse replicas of the database. This problem of replica consistency, despite having been under scrutiny for a long time, is still far from being solved in general for database servers [6], [7];
- regarding protective wrapping, we have outlined elsewhere [8] the idea of protective wrapping for OTS components. Wrappers intercept both incorrect and potentially dangerous communications between OTS components and the rest of the

system, thus protecting them against each other's faults. For an OTS SQL server, the protective wrapper protects the clients against faults of the server, the server against faults of the clients, and also each client against the indirect effects of faults of the other clients.

In our design approach we assume no changes to the OTS SQL servers, since we do not have access to their internals. By necessity, therefore, our solutions are based on restricting the interaction between the clients and the SQL server(s).

2 The Experimental Environment for OTS SQL Servers

The testbed has been built in collaboration between the Centre for Software Reliability at City University, London, and the Technical University in Plovdiv, Bulgaria. It allows one to run various client applications concurrently against diverse SQL servers which use a significant sub-set of the entry-level SQL-92 language. The testbed contains a wrapper for the SQL servers, implemented as a DCOM component, accessed by the client applications. Fig. 1 shows its architecture.

The testbed was created to allow experiments with 3 functionally comparable OTS SQL servers, Oracle 8.0.5, MS SQL 7.0, Interbase 6.0. The servers can run under any operating system for which there are versions of the products used; we used Windows 2000 Professional edition for experiments with the three servers and several operating systems (Win2k, Win98 and RedHat Linux 6.0) for experiments with Interbase.

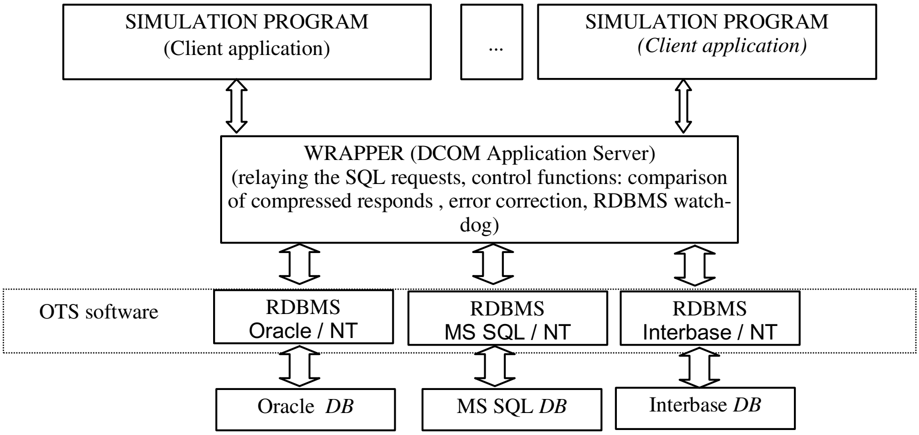


Fig. 1. Architecture of the testbed

We have experimented with between 1 and 100 clients and varying numbers of transactions per client, which include queries (SELECT) or modifications of the databases (INSERT, UPDATE, DELETE), "triggers" and "stored procedures". We have used two applications: i) a variation of a real-life warehouse application; ii) a simplified banking application, in which funds are being transferred between accounts under the invariant condition that the total amount of funds remains constant. This invariant allows a simple correctness check ("oracle") for whether the overall series of transactions is processed correctly by the server(s). With the "warehouse" client application, a comparison of the tables in the databases checks at predefined intervals whether the databases remain consistent, but no oracle exists to detect which of the servers has failed in case of disagreement. A third application is under development, based on the TPC-C benchmark [9].

The testbed allows different configuration parameters to be changed such as:

- the number of clients and of queries submitted by each in an experiment;
- the "demand profiles" of the clients, a probability distribution defined on the set of queries used. The query types and parameter values are chosen by the testbed, according to user-set probability distributions. A "Template Editor" tool exists for extending the set of transactions and setting the probability distributions, so that one can experiment with a wide range of loads on the servers;
- various modes of concurrency control between the clients:
  - *Free mode*, i.e. unrestricted access to servers by all clients. The level of isolation between the transactions provided by the servers is set to "serialisable", but no mechanism (e.g. atomic broadcast) is implemented in the testbed to control the order in which the queries are delivered to the individual servers and hence executed by the servers. The clients are multithreaded, with a separate thread talking to each of the servers.
  - *Bottleneck mode*, which imposes a very restrictive total order of the access by the clients to the server, with no concurrency between the clients. The threads representing the clients are synchronised (using critical sections) and the servers are supplied with only one transaction at a time. The next transaction (coming from any of the competing clients) is only initiated after the previous transaction is either committed or rolled back;
  - *WriteBottleneck mode*, in which the wrapper allows an arbitrary number of concurrent observing (i.e. read-only) transactions to be sent to the servers but no concurrency between the modifying transactions (which contain at least one INSERT, DELETE or UPDATE statement). A modifying transaction can only be started after the previous modifying transaction is completed (committed or rolled back).
- intervals for comparison of the tables in the databases and for "ping"-ing the servers to check whether they are still functioning.

For each experiment, a detailed log of events is recorded, including, e.g., all queries as sent, all exceptions raised, ping responses, results of database comparison, with timestamps for queries and responses.

3 Wrapping against Known Faults in a Server

A form of fault-tolerance is to deal explicitly with known faults. We give one example here.

With the Microsoft SQL v7.0 server, we observed that when the number of clients exceeded 20, the sharing of LOCKS between the competing threads created by the SQL server to serve its clients could cease to work properly. A peculiar situation could arise in which some clients acquired the locks they needed but remained in the "waiting" state, thus keeping all the other clients (trying to acquire the same locks) from continuing (this abnormal situation is not a deadlock, which the server would detect and handle by rolling back all competing transactions but one). The problem only occurs when the number of concurrent clients is large, and become more frequent as this number increased.

As we later found out, Microsoft reported the problem as due to a fault of the SQL server (Bug #56013, [10]).

A work-around is for the administrator - or for an application - to detect the situation and intervene by killing the thread that holds all the LOCKs but remains in a "sleeping" state (i.e. is in the root of the chain of blocked threads). However, manual intervention by the administrator is costly and may still allow large delays before being undertaken. Handling the problem explicitly in the client applications is only satisfactory if *all clients* handle the situation properly.

We have found another fully automated solution, which is relatively painless and can be incorporated in a wrapper, without changes to the legacy clients. It utilises a parameter, specific for MSSQL, LOCK_TIMEOUT, which can be explicitly set *for each query*. Its default value is 0, i.e. the blocked thread would wait for the needed lock forever. Setting it to non-zero value (we used 10 seconds) would make the server raise an exception "Lock request timeout period exceeded" when the set lock timeout expires. Now the client instead of waiting forever will get the exception and can roll the transaction back, while the locks are passed on to other clients. This solution is sufficient to resolve the occurrences of "bad blocking", at the cost of some number of transactions being rolled back. It can be improved if we include in the wrapper an exception handler for LOCK_TIMEOUTs, which would gradually increase the LOCK_TIMEOUT period, or just repeat the transaction after rolling it back, and thus make the resolution of the "bad blocking" condition completely transparent to the client. The cost of our simple solution, of course, is rolling back multiple transactions: not necessarily a high cost. The alternative - killing the thread at the top of the blocking chain - also has its cost. If a server thread is killed, the connection between the client and the server is lost and a new connection will have to be established, which is a more expensive operation than rolling back a few transactions.

It is worth pointing out that our letting the wrapper manipulate the MSSQL-specific *lock timeout* does not interfere with the setting of the *query timeout* (a different mechanism, available to the client applications with any SQL server). A complex query may take very long to complete even under light load (e.g. executing a complex query with sub-queries) and, therefore, setting a large query timeout for all queries is reasonable. During the execution of a query, multiple LOCKS can be exchanged between the server threads which compete for access to a shared resource. Without bug #56013, long query timeouts can co-exist with very short LOCK_TIMEOUT without any aborts.

With this approach of implementing work-arounds in wrappers, a *user organisation* can provide fault tolerance for bugs it discovers to be detrimental to the dependability of its installations, without waiting for the vendor to recognise the problem and issue a patch, which in any case may not completely eliminate the undesired behaviour. When known problems are left open by the vendor, the system integrator has the only choice of either introducing a protective wrapper or building a work-around in the client applications. The latter option may well be more efficient, but it is more cumbersome to manage and implement correctly: the fix and any subsequent upgrade to it must be replicated in all the client applications. Note that in our example, if some clients did not properly use the LOCK_TIMEOUT defence, they could prevent the other, "well-behaved" clients from accessing a shared resource - forever. The well-behaved clients would be the only ones to receive multiple "Lock request timeout period exceeded" exceptions until the blocking chain of non well-behaved clients is removed somehow, e.g. by timing out the respective queries, which may take long.

In summary, implementing a fix in the wrapper reduces dependence on fixes by the vendor, and it seems *always a better option* than implementing it in the client applications, so long as feasible and the performance penalty incurred at run-time is acceptable.

4 Diverse-Redundancy with SQL Servers Guaranteeing Consistency

With the testbed developed (Fig 1), we wish to answer two questions:

- are the off-the-shelf SQL servers sufficiently diverse in their failure behaviours that a diverse-redundant server would be significantly more reliable than the individual servers?
- to what extend can one build a software fault-tolerant server with OTS SQL servers without altering the internals of the servers?

Regarding the first question, preliminary work with the bug reports publicly available for two open-source SQL servers, PostgreSQL 7.0 (www.postgresql.org) and Interbase 6.0 (firebird.sourceforge.net), indicates encouraging results. We have demonstrated via manual testing that our setup can tolerate most of the bugs of the two SQL servers reported over one year. We plan to extend this work to the other SQL servers we experimented with, and to supplement it with statistical assessment of the reliability gains, via extensive automated testing under different testing profiles.

Regarding the second question, we achieve consistency between the diverse SQL servers using only synchronisation mechanisms implemented in a wrapper (Fig. 1). Achieving consistency between diverse SQL servers is difficult. All SQL servers must be guaranteed to execute the same serialisable transaction history –"1-copy serialisable execution" [11]. This is difficult for identical replicas [6] and even more so for diverse SQL servers [3]. Here we have an example of "eager replication" regarded by many as very difficult [6] and, therefore, rarely used. A recent survey of the replication mechanisms used in the leading commercial SQL servers can be found in [12]. The available solutions are for non-diverse replicas and are based on vendor specific replication mechanisms, optimised for high performance at the expense of consistency. The generic mechanisms proposed by various researchers, e.g. [13], [14], universally require access to the internals of the servers so that the replicas can achieve a consistent view on the order and the results of transaction processing. Since we use OTS SQL servers whose internals we cannot modify, none of these solutions is available to us.

Using the testbed in *Free mode* gave us plenty of examples in which the consistency between the databases was violated. Actually, this problem was exacerbated by the original implementation of our wrapper, which did not even attempt to deliver the queries within a transaction in the same order to the different SQL servers. It is worth checking whether implementing a mechanism which guarantees the deliveries of the queries to the servers in exactly the same order would have an impact on the evolution of the databases. In any case it would not avoid inconsistencies, since different servers use different optimisation strategies, and may alter the order of the execution of the concurrent queries, compared to the order in which they are delivered to the server, in different ways.

In summary, we need to constrain concurrency to guarantee consistent evolution of the diverse databases. What are the non-intrusive (i.e. not altering the internals of the SQL servers) options available? An obvious option is to eliminate concurrency completely, via the *Bottleneck mode* of operation of our wrapper: a single query at a time is executed, guaranteeing that all servers will execute the *same serialisable history (1-copy serialisability).* Provided the servers process the query correctly, the databases will stay consistent. With tests with the *Bottleneck mode* of operation, consistency was indeed preserved[2]. As a by-product of using this mode, deadlocks between clients are eliminated. However, the limitations imposed are so restrictive that this mode is hardly of any interest. SQL servers are designed to provide high throughput under a wide range of circumstances, inlcuding heavy load and thousands of concurrent transactions, benefits which are denied in the *Bottleneck mode*.

Our alternative is to use the *WriteBottleneck* mode, in which many read transactions can be executed concurrently together with *at most one write transaction*. The reason for implementing this is that in most real-life applications of databases, most transactions are *observing transactions* (i.e. SELECT queries). This is particularly true for most web applications. In the extreme case of some large web-based databases, the only on-line operations are SELECTs, while updates are only run off-line. In the *WriteBottleneck* mode, the *changes occur in the same order* for all databases, guaranteeing the consistency of the changes across the servers. The transaction histories on the servers may only differ in the order of their read transactions. This may lead to inconsistent results returned by the read transactions: voting on the outputs from the servers may produce mismatches even if the servers work correctly. Voting is, thus, no longer a trustworthy error-detection mechanism. Without voting, this solution is only fully fault-tolerant so long as the database servers have a fail-silent of crash-fail semantics [15] - which is, on the other hand, still assumed by most developers of database applications.

The *WriteBottleneck mode* also allows two more standard tricks for improving the throughput of the read transactions:

i) the wrapper can return to the client the first among the redundant results from a SELECT query: diversity will bring some performance improvement if different servers perform best on different queries;
ii) the wrapper can perform load balancing for read transactions by forwarding queries to only some servers, and thus reduce the load on the individual servers and increase overall system throughput. This may compensate for the delays on write transactions due to the *WriteBottleneck mode*.

The *WriteBottleneck mode* eliminates, as a by-product, all problems caused by concurrent execution of write operations by the servers. A recent study reports that SQL servers employing snapshot isolation, e.g. Oracle, have trouble handling the “write skew” problem [16]. This problem, first described in [17], is as follows. Suppose that X and Y are data items representing bank balances for a married couple, with the constraint that X+Y>0 (the bank permits that either account be overdrawn as long as

[2] We always explicitly tested the effects of any solution we designed. This experimental confirmation is important, even if the solution, as specified, can be proven to have a desired property, and even if the solution is correctly implemented in the wrapper, because its operation relies on the functioning of the SQL servers - whose internal details the integrator cannot verify - and thus may violate that property.

the sum of the account balances remains positive). Snapshot isolation is reported to have a problem with two transactions which concurrently attempt to withdraw from the two accounts, X and Y. It is possible to commit both transactions and leave the accounts with the constraint X+Y>0 violated. The *WriteBottleneck mode* clearly eliminates the problem. SQL servers using snapshot isolation will work properly if accessed via a wrapper in *WriteBottleneck mode* of operation. If this mode of operation is acceptable, there is no need for changes of the client applications as proposed in [16] to guarantee that a set of sufficient conditions for serialisability are met.

The expectation to see the diverse databases stay consistent under the *WriteBottleneck mode* has also been confirmed by testing with a few millions of transactions on the three diverse SQL servers of Fig 1.

To conclude, the *WriteBottleneck mode* is less restrictive than the *Bottleneck mode* and has *some practical relevance*, since its performance penalty may be negligible. Predictably, in our tests the performance of the testbed (measured by the time to execute a set of transactions) was better in *Free* than in *Bottleneck mode*. However, with a light load of write queries the *WriteBottleneck* and the *Free modes* are comparable. Of course, as the load of write queries increases the performance under the *WriteBottleneck* decreases, approaching the performance of the *Bottleneck mode*.

The *Bottleneck mode* has the advantage of allowing voting for error masking, at the cost of severe performance limitations. The *WriteBottleneck mode* reduces the latter disadvantage at the cost of making voting either unusable or more expensive (e.g., if discrepancies are rare enough they can be handled by aborting and retrying the transactions involved).

5 Related Work

Replicated databases are common, but most designs are not suitable for diverse redundancy. We have cited in the previous section some of the solutions proposed. Recent surveys exist of the mechanisms for eager replication of databases [7], and for the replication mechanisms – mainly lazy replication – implemented in various SQL servers [12]. The Pronto protocol [13] attempts to reduce the negative effects of lazy replication using ideas typical for eager replication. One of its selling points is that it can be used with off-the-shelf SQL servers, but it is unclear whether this includes diverse servers. A potential problem is the need to broadcast the SQL statement from the primary to the replicas. We have observed that the syntax of the SQL statements varies between commercial SQL servers, even for a standardised set of statements. This would create a problem with diverse SQL servers and would require translating the statements into the SQL “dialects” spoken by the diverse SQL servers involved in the replication, which inevitably will slow the replication down.

In software development, the idea of wrapping at various levels has been exploited as a means of making a particular function compatible with the component framework used, e.g. COM (DCOM), Java RMI, CORBA, etc.

Protective wrapping has received some attention recently [18], [19]. In [20] we described an approach in which protective wrappers are seen not only as a means of tolerating *known bugs* but also as a means of correcting *suspicious system behaviour*.

The ideas proposed in [2] are also enjoying some popularity for the purpose of security (intrusion tolerance). For instance, HACQIT (Hierarchical Adaptive Control of Quality of service for Intrusion Tolerance) [21], uses diverse off-the-shelf web-servers to detect failures (including maliciously caused ones, like defacement of web content) and initiate recovery. [22] proposes an architecture with multiple, diverse, COTS application servers. The Cactus [23] and SITAR [24] architectures support diversity among application modules to enhance survivability.

6 Conclusions

Software fault tolerance, in the form of checker/monitor components (protective wrapping) or diverse modular redundancy (N-version or N-self-checking systems) recommends itself as a cost-effective solution for improving the dependability of off-the-shelf software [2]. However, it may be objected that it is only feasible with simple software components, and is thus of very limited applicability. Our experience gives some evidence against this pessimistic view. We showed that simple wrapping techniques allow reasonable protection against design faults of the OTS SQL servers or their clients. In the case of the diverse-redundant configuration, by accepting some loss of efficiency we achieve a fault-tolerant SQL server at the moderate cost of multiple OTS SQL servers. The constraints imposed to guarantee consistency do limit performance, but this penalty may still often be minor compared to that of either using an undependable database or producing a highly trustworthy server. We have not yet estimated the dependability improvement produced by the diverse-redundant configuration, but we have evidence that it tolerates most of the known faults of the SQL servers used, which is a promising first piece of evidence. We intend to measure the potential dependability gains on long runs of test cases under different loads.

There are several possible extensions of our research since our implementation of a software fault-tolerant SQL server is only a demonstration prototype. Our goal in building it was to measure the *potential dependability gains* offered by diversity, not to build a full-fledged fault-tolerant server. For instance, we have not implemented an efficient solution for recovery of a database that is found to be corrupted. Scalability too, a very important aspect of connectivity with SQL servers, is not paid adequate attention, yet.

Acknowledgement. This work was supported in part by the DOTS (Diversity with Off-The Shelf components) project funded by the Engineering and Physical Sciences Research Council of the United Kingdom. Authors would like to thank all colleagues from the Centres for Software Reliability at City University and Newcastle Upon Tyne, UK, involved in the DOTS project and in particular Dr Alexander Romanovsky for the helpful comments on an earlier draft of this paper.

References

[1] ECUA, "3rd European COTS User Working Group (ECUA) Workshop," in *Panel with Industrial Collaborators*. Copenhagen, Denmark, 2002.

[2] P. Popov, L. Strigini, and A. Romanovsky, "Diversity for off-the-Shelf Components," presented at International Conference on Dependable Systems and Networks (DSN 2000) - Fast Abstracts supplement, New York, NY, USA, 2000.

[3] J. Gray, "FT101: Talk at UC Berkeley on Fault-Tolerance", 2000, pp. 62 slides, http://research.microsoft.com/~Gray/talks/UCBerkeley_Gray_FT_Avialiability_talk.ppt.

[4] P. Popov and L. Strigini, "Diversity with Off-The-Shelf Components: A Study with SQL Database Servers," presented at International Conference on Dependable Systems and Networks (DSN 2003) - Fast Abstracts supplement, 2003.

[5] J. C. Laprie, J. Arlat, C. Beounes, and K. Kanoun, "Definition and Analysis of Hardware-and-Software Fault-Tolerant Architectures," *IEEE Computer*, vol. 23, pp. 39-51, 1990.

[6] J. Gray, P. Helland, D. Shasha, and P. O'Neil, "The Dangers of Replication and a solution," presented at ACM SIGMOD International Conference on Management of Data, Montreal, Canada, 1996.

[7] M. Weismann, F. Pedone, and A. Schiper, "Database Replication Techniques: a Three Parameter Classification," presented at 19th IEEE Symposium on Reliable Distributed Systems (SRDS'00), Nurnberg, Germany, 2000.

[8] P. Popov, L. Strigini, S. Riddle, and A. Romanovsky, "Protective Wrapping of OTS Components," presented at 4th ICSE Workshop on Component-Based Software Engineering: Component Certification and System Prediction, Toronto, 2001.

[9] TPC, "TPC-C, An On-Line Transaction Processing Benchmark, v. 5.," 2002.

[10] Microsoft, "MS SQL 7.0, BUG #: 56013, FIX: Lock Conversion Processing Does Not Properly Wakeup Lock Waiter", http://support.microsoft.com/default.aspx?scid=kb;EN-US;236955, 2002.

[11] A. Bernstain, V. Hadzilacos, and , and N. Goodman, *Concurrency Control and Recovery in Database Systems*. Reading, Mass.: Addison-Wesley, 1987.

[12] A. Vaysburd, "Faul Tolerance in Three-Tier Applications: Focusing on the Database Tier," presented at 18th IEEE Symposium on Reliable Distributed Systems (SRDS'99), Lausanne, Switzerland, 1999.

[13] F. Pedone and S. Frolund, "Pronto: A Fast Failover Protocol for Off-the-shelf Commercial Databases," presented at 19th IEEE Symposium on Reliable Distributed Systems (SRDS'00), Nurnberg, Germany, 2000.

[14] F. Pedone, R. Guerraoui, and A. Schiper, "Transaction Reordering in Replicated Databases," presented at 16th IEEE Symposium on Reliable Distributed Systems (SRDS'97), Durham, NC, 1997.

[15] R. D. Schlichting and F. B. Schneider, "Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems," *ACM Transactions on Computing Systems*, vol. 1, pp. 222-238, 1983.

[16] A. Fekete, D. Liarokapis, E. O'Neil, P. O'Neil, and D. Shasha, "Making Snapshots Isolation Serializable," 2000, pp. 16.

[17] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil, "A Critique of ANSI SQL Isolation Levels," presented at SIGMOD Internationa Conference on Management of Data, 1995.

[18] M. C. Mont, A. Baldwin, Y. Beres, K. Harrison, M. Sadler, and S. Shiu, "Towards Diversity of COTS Software Applications: Reducing Risks of Widespread Faults and Attacks," HP Laboratories, Bristol, UK HPL-2002-178, 2002.

[19] A. Romanovsky, "Exception Handling in Component-Based System Development," presented at COMPSAC'01, Chicago, IL, 2001.

[20] P. Popov, L. Strigini, S. Riddle, and A. Romanovsky, "On Systematic Design of Protectors for Employing OTS Items," presented at 27th Euromicro Conference, Workshop on Component-Based Software Engineering, Warsaw, Poland, 2001.

[21] J. Reynolds, J. Just, E. Lawson, L. Clough, R. Maglich, and K. Levitt, "The Design and Implementation of an Intrusion Tolerant System," presented at International Conference on Dependable Systems and Networks (DSN 2002), Washington, D.C., USA, 2002.

[22] A. Valdes, M. Almgren, S. Cheung, Y. Deswarte, B. Dutertre, J. Levy, H. Saidi, V. Stavridou, and T. E. Uribe, "An Adaptive Intrusion-Tolerant Server Architecture," 1999.

[23] M. A. Hiltunen, R. D. Schlichting, C. A. Ugarte, and G. T. Wong, "Survivability through Customization and Adaptability: The Cactus Approach," presented at DARPA Information Survivability Conference & Exposition, 2000.

[24] F. Wang, F. Gong, C. Sargor, K. Goseva-Popstojanova, K. Trivedi, and F. Jou, "SITAR: A Scalable Intrusion-Tolerant Architecture for Distributed Services," presented at IEEE Workshop on Information Assurance and Security, West Point, NY, U.S.A, 2001.

ImpACT: An Alternative to Technology Readiness Levels for Commercial-Off-The-Shelf (COTS) Software

James D. Smith II

Carnegie Mellon Software Engineering Institute
4301 Wilson Boulevard, Suite 902, Arlington, VA 22203
jds@sei.cmu.edu

Abstract. The use of Technology Readiness Levels (TRLs) as a tool in assessing acquisition and development program risk has steadily increased over the past several years. There is considerable evidence to support the utility of using TRLs as part of a risk assessment, but there are some difficulties in using TRLs with software, especially Commercial-Off-The-Shelf (COTS) software technology and products. These difficulties take several forms, including "blurring-together" various aspects of COTS technology/product readiness; the absence of some important aspects of readiness; COTS product "decay;" and no mechanism to account for changes in the relative importance of the contributors to technology/product readiness over time. This paper briefly examines these issues, and proposes an alternate methodology—ImpACT—for assessing COTS software technology and product readiness which considers these factors.

1 Introduction

Technology Readiness Levels (TRLs) have been used within the National Aeronautics and Space Administration (NASA), as part of an overall risk assessment process, since the late 1980s. By the early 1990s, TRLs were routinely used within NASA to support technology maturity assessments and comparisons of maturity between different technologies [1, 2].

Within the United States Department of Defense (DoD), there has been considerable interest in using TRLs as part of risk assessments for entire systems, including hardware and software. Current DoD guidance requires technology readiness assessments prior to entering System Development and Demonstration; TRLs are one approach to meeting this requirement [3]. The Air Force Research Lab has adapted the NASA TRLs for use in assessing the readiness of critical technologies for incorporation into weapon systems, and the Army Communications Electronics Command (CECOM) has developed a draft set of TRLs to support software technology management [4, 5].

Several sources cite the difficulties in applying TRLs to assess the readiness of software-based technologies and products [5, 6]. Some of the characteristics of TRLs that affect their use in assessing COTS software technology and product readiness are discussed in more detail in the following sections.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 127–136, 2004.

1.1 "Blurring" Together Multiple Aspects of Readiness

One of the difficulties with using TRLs in readiness assessments involving COTS software technology and products is the way in which TRL definitions combine, or "blur" several different aspects of readiness. Because different aspects of maturity may be more-or-less important within a particular program context, this makes it difficult to understand the contributions of individual readiness components to the overall product or technology readiness. Thus, the implications of these individual aspects to the overall readiness of a product or technology—and by extension, to the overall risk assessment—are difficult to discern.

As an example, in their draft TRLs for software, CECOM defines TRL7 as

> *Represents a major step up from TRL6, requiring demonstration of an actual system prototype in an operational environment… Algorithms run on processor of the operational system and are integrated with actual external entities. Software support structure is in place. Software releases are in distinct versions. Frequency and severity of software deficiency reports do not significantly degrade functionality or performance. VV&A complete. [5].*

Note how this definition combine different aspects of technology readiness (e.g., functionality, maintainability, and reliability), making it all but impossible to understand how the contributions of any one facet of readiness affects the overall product or technology maturity.

1.2 Product/Technology Criticality

Just as importantly, TRLs leave out such considerations as the degree to which the technology is critical to the overall success of the system (including how difficult it would be to replace it, or assume some fall-back posture, should the technology in question prove unacceptable), or the suitability of the technology in question to its intended use within the system.

Some programs have attempted to deal with this effect by correction factors to adjust the TRL of a given technology for, say, the criticality of that technology to the success of the system, or the technical complexity of the technology [7].

1.3 COTS Software Product "Decay"

TRLs are used to gauge technology or product readiness growth within a specific context; this information is then used to inform judgments about the maturity of a given product or technology for use in a particular operational environment. Therefore, a product assessed as being at TRL 9—the highest level—is viewed as being at low risk to the system. The fact that a product or technology is subject to "decay," (e.g., replacement by a newer release, or product of unproven maturity, or simply retired without replacement), is not accounted for in the existing TRL definitions.

1.4 Time-Varying Contributions to Readiness

Another problem with existing TRL definitions is that for different phases of an acquisition, or different lifecycle phases within a development, different technology readiness levels may be appropriate *for different reasons*. For example, in a "proof of concept" demonstration, it is reasonable to use relatively immature, untested technologies as part of the demonstration. On the other hand, for a program in post-deployment sustainment, use of "tried and true" technologies and products is prudent.

Various programs have attempted to deal with this by applying some form of "correction factor" to a raw TRL. One approach used is to adjust a TRL downward by some amount if a particular technology or product comprises more than some percentage of the functionality of the system [7]. Another technique used is to "normalize" technology maturity to the relevant environment for the different lifecycle phases of an acquisition or development (e.g., for a laboratory "benchtop" test, a product or technology with a TRL of 3 or 4 may be acceptable) [4, 5]. Neither of these approaches addresses the issue of how different aspects of maturity contribute—in varying degree—to defining the context in which to understand technology or product maturity during different phases of system acquisition and development.

2 An Alternative Approach

The previous section outlines some of the challenges encountered in using TRLs to assess programmatic and technical risks for systems that include COTS software products. The remainder of this paper will propose an alternative approach that addresses these issues.

2.1 Background

This work grew out of an earlier SEI effort to develop a framework for evaluating COTS technology maturity. A key contribution of this effort was the recognition that technology readiness—especially for COTS software products and technology— is not something that can be represented by a single number. Rather, technology readiness is the result of several factors within a particular context, similar—in concept—to the definition of the Modeling and Simulation Technology Readiness Levels (M&S TRLs) proposed by the Department of Energy, which defined modeling and simulation technology readiness as a function of "correctness," "usability," and "relevance" [8].

In this initial framework, the readiness attributes were:

- When needed (and impact to system if not available)
- Off-the-shelf (verification)
- When projected to be ready (and level of investment needed to bring to that point)
- Projected date of obsolescence (and impact to system)

To use this within a program, software products or technologies would be evaluated against each of the attributes, and assigned values (red/yellow/green) according to the degree to which each product satisfied the conditions described by the attributes. As an example, for the "Off-the-shelf" criterion, a rating of "Green" would mean that the product was in broad commercial use, "Yellow" would indicate that the product had been demonstrated in a relevant environment, and "Red" would signify that your system would represent the first use of the product in question. In addition, a product was rated "High," "Medium," or "Low" based on its importance to the system.

2.2 ImpACT: A Multi-attribute Readiness Description

This initial attribute set afforded greater insight into technology maturity than that obtained from TRLs, but there were difficulties in using it to evaluate COTS technology and product readiness. First, these attributes had the same non-orthogonality problem as TRLs. For example, the "when needed (and impact to system if not available)" attribute combined elements of both time and criticality into a single rating. Similarly, there was a blurring of concerns in each of the initial attributes, resulting in little quantifiable improvement in understanding software technology and product readiness than that provided by TRLs.

This experience led to a realization that a different set of maturity attributes was necessary. From our previous experience with COTS-based systems, and after an extensive (though by no means exhaustive) review of the available literature, four main factors emerged as key elements in addressing the software product/technology maturity "problem":

1. The criticality of the software technology or product to the system. To what degree is the technology or product essential for the successful operation of the system, as well as the amount of disruption to the system should the product or technology not work, or not be available? This attribute reflects how closely the system architecture and/or implementation is tied to a particular technology or product, and is an indication of the potential for technology or vendor "lock," where it is extremely difficult—or impossible—to substitute a different technology or product within a system.
2. The availability of the software technology or product. Is this a commercially available product in widespread use in relevant domains? Or, is it a home-grown engineering tool with no commercial support?
3. The degree of "fit" or "misfit" of the software technology or product and the system. Does it have the necessary functionality? If not, how critical is the missing functionality? Are there unneeded capabilities? Do these introduce other complications (e.g., vulnerabilities, undesired modes of operation, etc.)?
4. How well the product or technology lifespan matches the needs of the system. Will a product or technology be ready when needed? Will it still be supported by its developer after incorporation into the system, or will it become an "orphan"?

These factors were then refined into a revised attribute set—ImpACT—defined as:

- ***Imp**ortance* – Criticality to the system; difficulty of effecting a work-around if the technology or product doesn't work (or isn't available)
- ***A**vailability* – The degree to which the product or technology is commercially available
- ***C**apability* – The functional fit (or misfit) between the product or technology and the requirements of the system

- *Timeframe* – A measure of how the lifecycle of the product or technology matches the lifecycle for the system. Will it be available when needed? Over the life of the system?

This attribute set appears to address the orthogonality and criticality issues, as well as the issue of "software decay" by providing a measure of not only when the technology or product will become available, but also when it will cease to be viable in a given context. Definitions of each of these attributes are provided in the Appendix. The following sections will discuss an approach to reason about and quantify the time-varying contributions of the individual components of product and technology readiness.

2.3 Acquisition and Development Maturity Growth

To better understand how different types of acquisitions, or different phases in a system development, are affected by the different attributes of technology or product maturity, a picture may be useful. In their paper, Hanakawa, *et al*, model the growth of knowledge during software development, and show how that growth can be represented by the family of sigmoid (s-shaped) curves, as shown in
Fig. 1 [9]. Extending this model to a software-intensive system acquisition or development, and equating "knowledge" with some measure of maturity (such as requirements satisfaction or technical performance measure improvement), some generalizations can be made. For example, a typical acquisition or development program will mature slowly during initial concept exploration and technology development, until some critical point is reached (e.g., fundamental science is understood, algorithms validated, etc.) at which time the rate of progress increases. As a program moves towards greater maturity, and most—though probably not all—requirements are satisfied, progress tapers off. In Hanakawa's model, the exact shape of this curve is dependent on the statistical distribution of tasks (e.g., requirements to be satisfied, program milestones, etc.) and their degrees of difficulty, the knowledge/competence of the organization to perform these tasks, and the rate at which knowledge accumulates by performing these tasks. Thus, every acquisition or development will result in a unique "maturity profile."

In Fig. 2, typical DoD acquisition phases and milestones are overlaid on a representative maturity profile. Within each of these phases, the slope and curvature (concave or convex) of the curve reflects the changing rate of maturity growth.

This maturity profile provides some insight into how the relative importance of the individual components ImpACT can change during the course of the program. For example, during a technology demonstration (i.e., ATD, ACTD), the fact that a software technology or product is projected to become unsupported sometime during the demonstration is probably less significant than the same situation during the later stages of development, or during post-deployment sustainment. The key to this approach is that, while the absolute values of the individual contributors to product or technology maturity cannot be defined, it is possible to articulate the importance of any one aspect (e.g., "Capability") relative to another, using "fuzzy" definitions like "as important as," or "much less important than" for each phase in a program's life cycle.

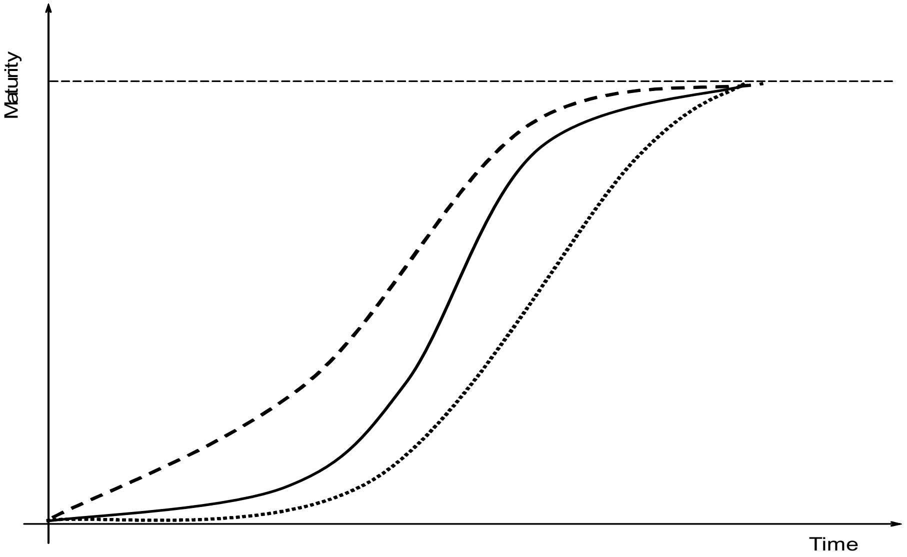


Fig. 1. Characteristic maturity growth curves, or "maturity profiles"

2.4 Reasoning Framework

Since ImpACT is defined by multiple criteria, it seems reasonable to use a technique like Saaty's Analytic Hierarchy Process (AHP) to assess the results [10]. AHP defines a process for evaluating multiple criteria, using a hierarchical structure (i.e., goal, attributes and sub-attributes, and alternatives) and pair-wise comparisons to determine the alternative that best satisfies the desired goal. The use of relative rankings, such as "*x* is much more important than *y*" or "*x* has roughly the same importance as *y*," works well in the context of software-intensive system acquisition and development where absolute criteria don't exist, but relative rankings *can* be defined with some de- gree of confidence. This approach also lends itself to examining the effects of time-varying affects of the contributions to maturity of each of the ImpACT criteria.

As a simple illustration of using AHP, consider an evaluation of two COTS products ("A" and "B") being considered for incorporation into a system. During this particular phase of the system's development, the criticality of the product to the system (Imp) is considered slightly more important than the ability of the product to satisfy the full range of requirements (C). Both of these attributes are significantly more important than the commercial availability of the product (A), and the temporal availability of the product with respect to the system (T) is of still less concern. Expressed in a pair-wise comparison matrix, this becomes:

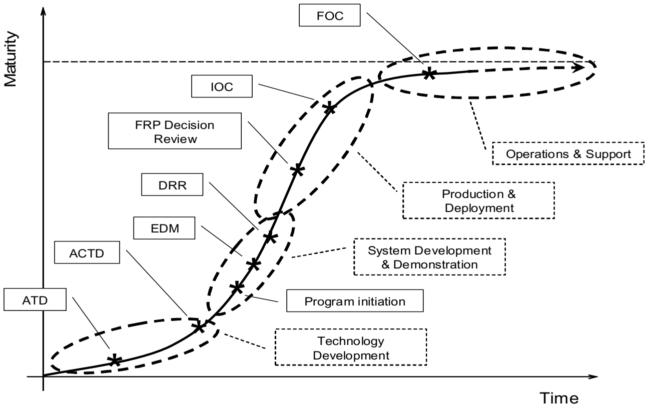


Fig. 2. Representative DoD acquisition/development program phases and milestones overlaid on a typical growth curve

$$\begin{array}{c|cccc} & I & A & C & T \\ \hline I & 1 & 7 & 2 & 9 \\ A & 1/7 & 1 & 1/3 & 1 \\ C & 1/2 & 3 & 1 & 7 \\ T & 1/9 & 1 & 1/7 & 1 \end{array} \tag{1}$$

The two products were evaluated against the ImpACT criteria, with the result:

$$A_{IACT} = [0,2,2,1] \qquad B_{IACT} = [2,0,3,2] \tag{2}$$

Looking at the Importance criterion (I), it was determined that for this specific development, during this life cycle phase, the mapping from ImpACT criteria to AHP relative rankings was as follows:

$$\begin{array}{rl} \text{ImpACT } 0 = & \text{AHP } 1 \\ 1 = & 2 \\ 2 = & 3 \end{array} \tag{3}$$

Thus, the PCM for products A and B with respect to the Importance criterion is:

$$\begin{array}{c|cc} & A & B \\ \hline A & 1 & 3 \\ B & 1/3 & 1 \end{array} \tag{4}$$

By a similar process, the PCMs for both products with respect to the remaining ImpACT criteria. Applying the AHP method results in weighted scores for the two products as shown:

$$\begin{array}{c}\text{Product A} = 0.68 \\ \text{Product B} = 0.32\end{array} \tag{5}$$

So, for this example, Product A's technology readiness appears to be significantly greater than that for Product B, given the relative importance of the readiness criteria (i.e., ImpACT) and the determination of the relative weighting of the values within the criteria.

While AHP provides a method to reason about the contributions of various attributes to satisfying a desired goal, neither AHP nor ImpACT define how the relative rankings of the criteria are derived. Just as the CMM® framework leaves the definition of appropriate processes to the implementing organization, ImpACT leaves the criteria evaluation definitions to the developing or acquiring organization. Within the context of any particular development or acquisition, one approach may be more suitable than another. For instance, there are several possible means of determining the degree of match/mismatch between a candidate product or technology and the desired capabilities; examples include the "Risk-Misfit" approach, described by Wallnau, Hissam, and Seacord in *Building Systems from Commercial Components*, or the "Gap Analysis" methodology described by Ncube and Dean [11, 12].

3 Conclusions

While there is a growing body of evidence that using TRLs as part of an overall risk assessment can lead to an improved understanding of the technological and programmatic risks in a system development or acquisition, there are several difficulties in applying "traditional" TRLs to the evaluation of software technologies. This is especially true for COTS software technologies and products, where TRLs don't provide any meaningful discrimination between mature, commercially-available technologies or products, nor do they take into account the inevitable decay which all software—especially COTS software—experiences. Finally, the existing TRL framework lacks any explicit mechanism to deal with the time-varying effects of the various contributors to technology and product readiness.

The ImpACT methodology provides an alternative to TRLs for COTS software products and technologies which directly addresses each of these shortcomings in a manner which can be tailored to an individual development or acquisition organization, for any software-intensive system.

References

1. Eisman, M., Gonzales, D.: Life Cycle Cost Assessments for Military Transatmospheric Vehicles. <www.rand.org/publications/MR/MR893/> (1997)
2. Mankins, J.: Technology Readiness Levels – A White Paper. <http://advtech.jsc.nasa.gov/ downloads/TRLs.pdf> (1995)

3. Department of Defense.: Operation of the Defense Acquisition System). <http://dod5000.dau.mil/DOCS/DoDI%205000.2-signed%20(May%2012,%202003).doc> (2003)
4. General Accounting Office.: Better Management of Technology Development Can Improve Weapon System Outcome. <http://www.gao.gov/archive/1999/ns99162.pdf> (1999)
5. Graettinger, C., Garcia, S., Siviy, J., Schenk, R., Syckle, P.: Using the Technology Readiness Levels Scale to Support Technology Management in the DoD's ATD/STO Environment. <http://www.sei.cmu.edu/publications/documents/02.reports/02sr027.html> (2002)
6. Graettinger, C., Garcia, S., Ferguson, J.: TRL Corollaries for Practice-Based Technologies. <http://www.acq.osd.mil/sis/Conference%20Presentations/TRL%20Corollaries%20for%20Practice%20Based%20Technologies.pdf> (2003)
7. Wong, B.: NASA Cost Symposium – Multivariate Instrument Cost Model-TRL (MICM-TRL). <http://ipao.larc.nasa.gov/symposium/MICM-TRL-Wong.pdf> (2000)
8. Department of Energy.: Modeling and Simulation Technologies Future Combat System Workshop. <http://www.amso.army.mil/topic/fcs/feb-conf/overview.ppt> (2000)
9. Hanakawa, N., Morisaki, S., Matsumoto, K.: A Learning Curve Based Simulation Model for Software Development. <http://ieeexplore.ieee.org/iel4/5475/14745/00671388.pdf?isNumber=14745&prod=CNF&arnumber=671388&arSt=350&ared=359&arAuthor=Hanakawa%2C+N.%3B+Morisaki%2C+S.%3B+Matsumoto%2C+K.%3B> (1998)
10. Saaty, T.: The Analytic Hierarchy Process. McGraw-Hill, New York (1980)
11. Wallnau, K., Hissam, S., Seacord, R.: Building Systems from Commercial Components. Addison-Wesley, Boston (2002)
12. Ncube, C., Dean, J.: The Limitations of Current Decision-Making Techniques. In: Dean, J., Gravel, A. (eds.): COTS-Based Software Systems. Lecture Notes in Computer Science, Vol. 2255. Springer-Verlag, Berlin Heidelberg New York (2002) 176-187

Appendix: ImpACT Attribute Definitions

Table 1. ImpACT Attribute Definitions and Values

Value	***Attribute***			
	Importance	***Availability***	***Capability***	***Timeframe***
0	At least one suitable alternative can be “plugged-in” with minimal tailoring	Widespread commercial use; “shrink-wrap”	“Perfect” fit between component capabilities and system needs	Available over projected life of the system
1	At least one suitable alternative available; reintegration required, with minimal software changes required	Limited commercial use	All necessary functionality present, but some minor “fit” issues	Available when needed, but anticipated that it will be replaced/retired during the system lifetime
2	Moderate reintegration effort required, with pervasive software changes	Public testing (release candidate, beta)	Deficiencies in one or more second- or third-tier functionality; work-arounds available	Available when needed; end-of-life with replacement announced
3	Significant architectural and/or implementation changes required, but extent limited to a single portion/aspect of the system	Private testing (alpha, beta)	Deficiencies in one or more second- or third-tier functionality; no work-arounds available	Available when needed; end-of-life without known replacement announced
4	Significant, pervasive architectural and/or implementation changes required - consider refactoring	“Opportunity-ware”	Significant deficiencies in one or more critical functional areas; work-arounds can provide required functionality with degraded capability	No longer available by “need date,” but end-of-life with replacement announced; not available by need date, but alternate product (e.g., prior version) available
5	No workaround available; “back to the drawing board”	“Vapor-ware”	One or more critical functions missing; unsuitable for use	No longer available by “need date,” and end-of-life without replacement announced; not available by need date, and no alternate product available

COTS-Based Systems – Twelve Lessons Learned about Maintenance

Donald J. Reifer [1], Victor R. Basili [2], Barry W. Boehm [1], and Betsy Clark [3]

[1] Computer Science Department, University of Southern California,
Los Angeles, CA 90089, USA
dreifer@earthlink.net, boehm@sunset.usc.edu

[2] A.V. Williams Building, Room 4111, College Park,
MD 20742, USA
basili@cs.umd.edu

[3] Software Metrics Inc, USA
Betsy@Software-Metrics.com

Abstract. This paper presents the twelve most significant lessons the CeBASE community has learned across a wide variety of projects, domains, and organizations about COTS-Based Systems (CBS) maintenance. Because many of the lessons identified are not intuitive, the source and implications of the lesson are discussed as well within the context of maintenance model for CBS.

1 Introduction

One of the major thrust areas for the Center for Empirically-based Software Engineering (CeBASE) is its COTS-Based Systems (CBS) initiative. The CBS initiative was started because practitioners need empirical data published to base their decisions upon. CeBASE is addressing this problem by capturing lessons learned and empirical data and making it available via its web site (www.cebase.org) and continued publications. In May 2001, we published our first paper which highlighted our CBS "Top "10 list [2] and initial findings relative to maintenance [4]. In this paper, we focus our attention on CBS maintenance activities and lessons learned to shore up the little knowledge that exists about this mysterious domain.

CeBASE was organized to support software organizations in answering the key questions about what models, tools and techniques to use to produce high-quality software on schedule and within budget limitations throughout the software life cycle. Its focus is not limited to the development phases of the life cycle. CeBASE accumulates empirical models in order to provide validated guidelines for making such selections, recommending areas for research, and supporting software engineering education. CeBASE's primary objective is to transform software engineering from a fad-based practice to an engineering-based discipline in which development processes are selected based on what is known about their effects on products, through

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 137–145, 2004.

synthesis, derivation, organization, and dissemination of empirical knowledge on software development and evolution phenomenology. The lessons learned identified in this paper are directed towards achieving these goals.

2 COTS and the Maintenance Process

There is a great deal of confusion over the activities that software organizations perform during software maintenance. Figure 1 tries to how COTS renewal and refresh activities fit as part of the work performed which is organized into the following two often competing groupings:

Sustaining Engineering: Those activities conducted during the maintenance phase of the software life cycle to sustain software operations and support activities after the software system goes operational. Typical sustaining engineering activities include:

- Applications repairs
- Configuration management
- Distribution management
- Facility updates and management
- Quality control
- Security administration
- System administration
- System software repairs
- User support and training
- Unscheduled software repairs

The following separable COTS activities are conducted as part of these activities:

- Continued COTS package evaluation
- COTS vendor liaison (problem reporting and repair and update evaluation and influence)
- COTS package patch management and repair
- Interim response to COTS surprises (e.g., vendor default)

New Version Release: Those activities conducted during the maintenance phase of the software life cycle to generate new versions that incorporate new user-defined and prioritized features and functions (enhancements), scheduled repairs (fixes to patches) and perfective changes (typically to improve performance or address resource utilization shortfalls). Typical new version release activities include:

- Requirements engineering (for the new version)
- Version development
- Version alpha testing
- Version beta testing
- Version acceptance testing
- Version documentation
- Version management
- User support and training (for the new version)

The following separable COTS activities are conducted as part of these activities:

- COTS package update synchronization (with release cycle)
- COTS package repair and/or replacement (COTS refresh)
- COTS package tailoring and enhancements (COTS renewal)
- COTS package integration and testing (including retesting of changed configurations)
- COTS test script development (for use during alpha, beta and acceptance testing)

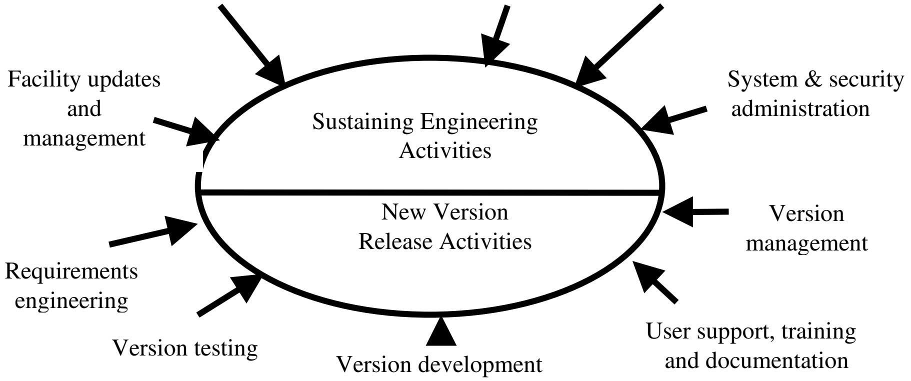

Facility updates and management
Sustaining Engineering Activities
System & security administration
New Version Release Activities
Version management
Requirements engineering
Version testing
Version development
User support, training and documentation

Fig. 1. Software Maintenance Activities

The most interesting observation made based on updates made during the last fifteen years to our original maintenance study [3] was that most software staffs remain workforce-limited during the operations and support phase of the software life cycle. This means that management does not acquire the staff needed to handle the required workload to support both the new version release and sustaining engineering tasks. Instead, they are told to figure out what workload they could accommodate with an existing, fixed workforce whose skills and experience were typically leaner than the staff assigned to projects developing new software. In contrast, development teams are most often tasked to figure out how many people they will need to develop software with a given functionality per an often constrained schedule.

Typically, the workload is equally distributed in most software shops between new version release and sustaining engineering activities. However, when conflicts arise for staff, many managers resort to load-balancing. For example, when sustaining engineering activities requires a disproportionate number of people to install and train the staff in a new operating system (i.e., moving from Linux to Windows or vice versa), management will often either extend the promised new version delivery date or cut back on its delivered features and/or functionality.

3 COTS Maintenance Lessons Learned

Within the context of the maintenance process shown in Figure 1, we present our lessons learned relative to COTS maintenance. These findings extend work previously reported in the literature and available at the Center for Empirically Based Software Engineering's portal site (www.cebase.org). For each lesson learned, we summarize our findings, provide a source for our observations, and suggest what it means relative to how we could improve the way that we manage the maintenance of COTS products.

- **Lesson 1** - The refresh and renewal process for COTS-based systems (CBS) needs to be defined a priori and managed so COTS package updates can be synchronized with each other and the organization's release and business cycles. If they aren't, updates may occur sporadically during the maintenance part of the cycle and the risk of technology obsolescence may increase dramatically. The magnitude of the problem increases exponentially with the number of independent COTS products to be synchronized (see Lesson 5).

  Source: Survey of 34 COTS-based systems as part of a recent Air Force Scientific Advisory Board study looking at management of COTS software within weapon systems (SAB-TR-99-03, April 2000). Surveys of FAA and NASA CBS maintenance projects.

  Implications: Currently, few COTS software life cycle models address maintenance processes for CBS. Guidance is needed to define the activities that take place during the refresh and renewal process. Criteria for making decisions relative to when to incorporate updates within releases also need to be defined along with their associated risks and business implications. Preferably, the refresh frequency is tied to operational mission cycles (annual new-product rollout, biannual operator retraining).

- **Lesson 2** – COTS software capability and quality evaluation needs to be managed as a continuing task during the maintenance phase.

  Source: Most publications that discuss CBS processes advocate that a market watch function should be established as well as a process to evaluate specific COTS products (see National Research Council of Canada, Software Engineering Institute and University of Southern California (USC) research reports).

  Implications: Most COTS software studies recommend that firms (1) establish a market watch function to keep track of where their packages are heading and (2) an evaluation activity to continuously assess options. A market watch looks at the marketplace as a whole, monitoring the health and viability of a specific vendor as well as what the competitors are coming out with. The COTS evaluation activity provides you with a detailed assessment of package capabilities, quality issues and future options. It typically involves conducting some form of operational demonstration.

- **Lesson 3** – The cost to maintain CBS often equals or exceeds that of developing custom software. Maintenance in this context involves updating CBS with new releases, modifying wrappers and glue code and incorporating fixes/repairs into the system. Costs average 10 percent of the development cost per year over a ten year life for the system. Although releases occur every year, technology refreshes for COTS aren't synchronized as they occur every two years. Defect rates per release for CBS are poorer than for custom-built software, averaging 10 to 40 percent higher.

  Source: Recent study by RCI across 3 large firms looking at the cost of COTS software across 16 systems, some of which employ over 40 different packages.

  Implications: Even though firms can save time and effort during development using CBS, they should evaluate the total life cycle cost of options prior to making commitments. Such analysis could identify risks that negate many of the advantages that CBS brings to the table. For example, firms must coordinate glue code updates along with package improvements. Research at USC has found that a line of glue code costs, on average, three times the cost of a line of custom code to develop and maintain. Thus, the effort during maintenance can get quite expensive. In situations where COTS functionality is relatively small and the CBS has a long life, custom solutions may work out to be cheaper than COTS alternatives. On a related note, project managers that were interviewed as part of our research said that, unlike custom systems, CBS need a continual stream of funding throughout their life cycle. Such funding is required to keep up with the marketplace that is dynamic and recognizes the fact that vendors are continually releasing new versions. Funding was an issue with several projects we spoke with because maintenance budgets get cut often. The opinion of those we spoke with is that maintenance budget cuts hurt a CBS more than a custom system because they are more able to delay maintenance in the latter if necessary.

- **Lesson 4** – The most significant variables that influence the cost of CBS maintenance include the following (in priority order):
  - Number of COTS packages that need to be synchronized within a release
  - Technology refresh and renewal cycle times
  - Maintenance workload for glue code and wrapper updates
  - Maintenance workload to reconfigure packages
  - Market watch and product evaluation workload during maintenance
  - Maintenance workload to update databases
  - Maintenance workload to migrate to new standards

  Where maintenance workload represents the amount of effort software engineers expend to handle the task at hand.

Source: Recent study by RCI across three large firms looking at the maintenance cost of COTS software across sixteen systems, some of which employ over 40 different packages. Parameters identified using a survey that asked those responsible for maintenance for insight. Parameters are listed in impact order with number of packages requiring synchronization being twice as sensitive as the need to migrate to new standards.

Implications: Cost models like USC's COCOTS (see http://sunset.usc.edu for information) need to be updated to encompass the full CBS life cycle. Currently, they focus on estimating the costs associated with evaluating, adapting and deploying COTS software packages during development and maintenance. In the future, such models need to incorporate additional variables like the last three bullets on our list to permit those assessing life cycle costs to estimate the full cost of the maintenance portion of the CBS life cycle.

- **Lesson 5** – Maintenance complexity (and costs) will increase exponentially as the number of independent COTS packages integrated into a system increases.

  Source –Initial results of study of twenty projects by the COCOTS team at USC [1].

  Implication - Projects should understand the maintenance implications of integrating large numbers of COTS products into a system. In addition to the effort involved in the initial integration, they need to consider that each product will evolve in its own way, according to different timetables, at the discretion of the vendors. You will have to expend considerable effort to handle the continuing evolution of these products (e.g., understanding the impact of an upgrade on the rest of the system, making changes to glue code). An important COTS evaluation criterion is the COTS vendor's track record on stability and predictability.

- **Lesson 6** – Significant time and effort must be spent up-front analyzing the impact of version updates and new releases (even when the decision is made not to incorporate the updates).

  Source: Initial results of study of 20 projects by the COCOTS team at USC suggest that analysis efforts during maintenance directed towards updates can tax the organization severely. This is particularly true for safety-critical systems.

  Implications: Maintenance modeling needs to assume that there are fixed and variable costs for CBS. Fixed costs are those associated with market watch and continued product evaluation. Variable costs are a function of the work performed to incorporate updates and fixes, changes and optimizations into the impending release. Workload balancing is needed to optimize the workload performed by the fixed staff as part of this process.

- **Lesson 7** – Flexible CBS software licensing practices lead to improved performance, reliability and expandability.

  Source: Best acquisition practices surveys done in 2000 and 2001 by RCI for the Army (see www.reifer.com for paper on innovative licensing).

  Implications: Partnering instead of conflict management was identified as the preferred approach to licensing. Shared goals lead to products with improved "goodness of fit" and "functionality" for the buyer. Leveraging relationships to achieve shared goals is strongly desired. Innovative contracting under such arrangements lead to deep volume discounts and priority service/bug fixes. Traditional approaches to licensing where contracts govern instead of relationships lead to distrust and poor results.

- **Lesson 8 –** Wrappers can be effectively used to protect a CBS from unintended negative impacts of version upgrades.

  Source: Projects interviewed for the COCOTS database. One project successfully used wrappers for information hiding so that different versions of COTS products (or different products) can be swapped without impacting the rest of the system.

  Implication: CBS architectures should accommodate COTS changes throughout the system life cycle.

- **Lesson 9 –** You may have to re-tailor COTS components with new releases to accommodate new features and functionality (including interfaces and external links). The effort involved in re-tailoring often proves to be non-trivial because new interfaces and links need to be supported along with the old ones.

  Source: In most cases, you will have to write new scripts to address these features and functionality. In addition, test scripts will have to be changed to incorporate new scenarios. All of these changes take time and effort that needs to be factored into your maintenance workload. If the number of scripts is large, the nominal maintenance cost of 10 percent per year will have to be increased proportionately.

  Implication: CBS architectures should accommodate COTS changes throughout the system life cycle.

- **Lesson 10** – The Achilles' heel of most COTS projects is the interface to legacy systems. They fail over and over again.

Source: The Center for Empirically Based Software Engineering lessons learned repository captured as a result of an e-workshop held on the topic of COTS maintenance.

Table 1. Post Delivery Error Rates by Application Domain

Application Domain	Number Projects	Error Range (Errors/KSLOC)	Normative Error Rate (Errors/KSLOC)	Notes
Automation	55	1 to 5	2	Factory automation
Banking	30	2 to 7	3	Loan processing, ATM
Command & Control	45	0.5 to 3	1	Command centers
Data Processing	35	1 to 10	4	DB-intensive systems
Environment/Tools	75	2 to 9	4	CASE, compilers, etc.
Military -All	125	0.2 to 3	< 1.0	See subcategories
▪ Airborne	40	0.2 to 1	0.5	Embedded sensors
▪ Ground	52	0.5 to 2	0.7	Combat center
▪ Missile	15	0.3 to 1	0.5	GNC system
▪ Space	18	0.2 to 0.7	0.4	Attitude control system
Scientific	35	0.9 to 3	1.2	Seismic processing
Telecommunications	50	1 to 6	2	Digital switches
Test	35	2 to 8	4	Test equipment, devices
Trainers/Simulations	25	1 to 7	3	Virtual reality simulator
Web Business	65	3 to 10	7	Client/server sites
Other	25	1 to 8	4	All others

Implication: Extra care must be taken when interfacing COTS to legacy systems. Packages with well defined interfaces should be selected especially when legacy system interface definitions are not well-defined.

- **Lesson 11 –** Out-sourced CBS applications that don't require refreshed COTS components in their contracts for delivered applications often have to live with obsolete COTS products.

  Source: Initial results of study of twenty projects by the USC COCOTS team. One delivery had 46% of its 120 COTS products having unsupported releases.

Implication: When contracting for CBS development, make sure to plan for and specify the refresh of COTS components prior to acceptance.

- **Lesson 12 –** When the error rates with COTS packages equal to or exceed those being experienced with other software, it is time to consider replacing the package.

  Source: Recent study by RCI across 3 large firms looking at the cost of COTS software across 16 systems, some of which employ over 40 different packages. Trend data shows that COTS package quality goes down more quickly than the norm when error rates rise above expectations. Because COTS packages are subjected to broader testing than most custom software, quality expectations for fielded, mature systems are higher than the norm.

  Implication: When sustaining COTS-based systems, maintain options that enable you to replace packages when their quality deteriorates. As shown in Table 1, norms for error rates are application domain dependent and range from 0.4 to 5 errors/KSLOC post-delivery (after one year of operational use).

4 Future Directions

To make better decisions relative to CBS, we need more empirical knowledge. To gain this knowledge, we need to better understand the life cycle processes people use when harnessing COTS packages. The initial findings reported above are but the first step in our attempts to capture this empirical knowledge and make it generally available. As part of our CeBASE work, we plan to continue to collect data and investigate the phenomenology of COTS-based systems. We will continue to publish our results as we gather and make sense of it. This work complements the more general results available at our CeBASE web site at http://cebase.org/cbs and the SEI web site at http://www.sei.cmu.edu/cbs.

References

1. C. Abts, "CBS Functional Density-A Heuristic for Better CBS Design," Proceedings, ICCBSS 2002, pp.1-9.
2. Victor R. Basili and Barry Boehm, "COTS-Based Systems Top 10 List," Computer, IEEE Computer Society May 2001, pp. 91-93.
3. Donald J. Reifer, "A New Approach to Software Operations and Support Modeling," Proceedings of the 3rd COCOMO User's Group Meeting, Software Engineering Institute, November 1987.
4. Donald J. Reifer, Victor R. Basili, Barry W. Boehm and Betsy Clark, "COTS-Based Systems – Nine Significant Findings Relative to Maintenance," Software, IEEE Computer Society, September/October 2003.

A Wish List for Requirements Engineering for COTS-Based Information Systems

Vito Perrone

HOC - Hypermedia Open Center
Department of Electronics and Information, Politecnico di Milano
Via Ponzio 34/5 20133 Milano (Italy)
perrone@elet.polimi.it

Abstract. This paper summarizes the main achievements of a research whose main goal was to investigate the current state-of-art in the field of requirements engineering for COTS-based systems. For this purpose, we have reviewed the most relevant contributions in this field over the last 10 years have been considered. After analyzing these research contributions, we defined a scenario composed by a number of punctual but relevant contributions and a number of methodological approaches coping with the requirements definitions for such systems. Finally, on the basis of this scenario, a *Wish List* of desirable features of a hypothetical approach has been defined and compared against the current situation. This list may act as an empirical means for evaluating new approaches addressing RE for COTS-based systems, and bases its foundations on the current needs pointed out by the major experts in this field.

1 Introduction

Many factors contribute to the success of an Information System (IS). Among them, a primary measure is certainly the degree to which it meets the purpose for which it has been thought (say *acceptance degree*). *Software systems Requirements Engineering* (RE), broadly speaking, is the process of discovering that purpose, by identifying stakeholders and their needs, and by documenting them in a form that is appropriate for analysis, communication, and subsequent implementation. Over the past decade, a huge amount of interest in methods and techniques dealing with RE has grown in the research community, leading to the definition of several well-affirmed approaches.

On the other hand, the goals of developing systems in a better, faster, and cheaper way, continue to drive software engineering practitioners and researchers to investigate software engineering methodologies [1]. To face these market drivers, the current practice of IS development consists of adopting commercial packages usually called COTS (commercial-off-the shelf components).

In the light of the above considerations, this paper addresses the issue of how to combine well-known benefits coming from applying RE approaches with the need of using COTS packages in building up a new IS? Indeed, using COTS packages re-

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 146–158, 2004.

quires systematic approaches that are able to consider the COTS option from the early stages of the building process and in particular to come up with requirements that make feasible such an option. Unfortunately, RE approaches used for developing systems from scratch may be, and usually are, inadequate to front the development using COTS. Various factors affect the development of COTS based systems: COTS are those the market offer; they are sold "like-that" sometime like a black-box; their code is often not available; they have been though as general purpose solution in a specific domain; and so forth.

Starting from these considerations, this research aims at investigating the effects of this trend on requirements engineering practice. In particular, we try to collect the most important results achieved in the RE research community, putting them in the form of a *wish list* of features a "complete" approach should embody.

2 Using COTS in Building up an Information System

After analyzing the most relevant contributions in RE field from one hand, and for developing COTS base systems on the other, we have been able to make a picture of the current situation in this sector. The situation can be described as composed by a number of punctual contributions and a small number of complex approaches providing a means to cope with the main issues in building a new COTS-based IS.

A first, partially surprising result of the investigation is that most of the approaches take into consideration the problem of treating COTS only operate at design level. This means that a considerable part of the research efforts dealing with COTS neglects the influence of using COTS during the requirements engineering activities, supposing that any is performed.

A second relevant result is that most of the activities performed throughout the requirements engineering process for such systems, if compared with the ones performed for a system to be built from scratch, are strongly influenced by a sort of *shift of the paradigm* according to the system is built. In the next paragraph a better explanation of this point is provided.

2.1 Paradigm Shifting

Using COTS components to build up a system implies a fundamental shift of the paradigm used to build a system from scratch and starting from its requirements. In a *custom system*[1], the process of building a system follows a paradigm in which, ideally speaking, requirements define the system to be. For such systems, the development process consists, in a broad vision, of the steps shown in fig. 1

[1] A system where a customer produces a set of requirements for hardware/software of the system and a contractor develops and delivers that system.

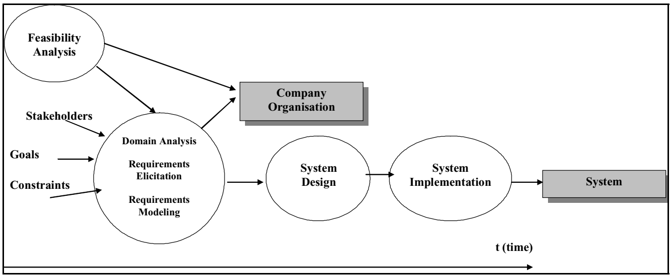


Fig. 1. Typical life cycle of a software system development

The *Feasibility Analysis* phase can be seen as an early domain and requirements analysis, in which the organization has to decide whether the system makes sense and which are the main directions to be followed in order to meet its own business objectives. For a custom system, if we consider the process as a whole looking at the sequentially-ness of the above mentioned phases, regardless the intrinsic iterative nature of the overall process, we can say that the Feasibility Analysis defines the large-grain structure of the organization the system will support. Then the RE process (as a whole) defines the thin-grain structure of the organization. In this light, system design can be considered as a function (ideally) of the RE output. This is because the system design and implementation are founded on this output. This is evidently an ideal situation, based on the assumption that designers completely understand the requirements specification and the implementation team is able to fulfill the design specification. Moreover in this picture we neglect the iterative nature of the process because the main aspect we want to point out is that the organization (by means of contractors) develops the system from scratch keeping the control of all or most of the phases: collect and define requirements; identify the architecture that satisfies the requirements; design individual subsystems in detail in order to fit within the architecture; code, test, and debug modules to meet the specified requirements; integrate sets of modules and subsystems into the complete system.

In COTS based systems the fundaments of the above paradigm need to be revised. In particular, some sequences and dependencies among phases of the overall process can give out. In this research we focus on the RE part showing as using COTS impacts this crucial phase of the overall process by collecting a number of insights coming from the research community.

2.2 Defining COTS Products and COTS-Based Systems

The definitions of COTS and COTS related concepts which can be found out in literature are usually very broad and cover a large variety of products. As a result, researchers and practitioners use the same word with different meanings. Some of these definitions are discussed in [2]. In order to avoid misunderstandings, here we provide a definition both for COTS products and COTS-based systems so that the reader can refer to these to understand the following sections. The definitions are quite general and include most of the others, making the paper consistent with most of the existing COTS definitions.

COTS Product Definition

A COTS product can be considered as one that:

- ❑ Can be sold, leased, or licensed to the general public (different license forms)
- ❑ It is offered "like-that" by a vendor trying to profit from it (or by a community trying to benefit from its usage by an increasing number of users)
- ❑ Intellectual property rights are retained by the vendor
- ❑ Identical copies are sold to a wide number of customers
- ❑ Access to source code as well as internal documentation is usually unavailable
- ❑ Complete and correct behavioral specifications are not available
- ❑ Periodical releases, with unpredictable evolutions and modifications, may be proposed by vendors
- ❑ Sometime, customers are provided with a number of predefined extension capabilities and personalization hock-points, but eventual modifications are not more under the vendor's responsibility

COTS-Based System Definition

The definition of COTS-based system we use, is the one proposed by Carney in [3] where he takes the point of view of the delivered system, instead of the parts: he identifies three types of COTS systems as a function of the number of COTS used and their influence on the final system: *turnkey systems* are built around a (suite of) commercial product(s); *intermediate systems* are built around one COTS but integrate other components; *integrated systems* are built by integrating several COTS, all on the same level of importance. In our investigation we refer to turnkey systems.

3 A Wish List

In this section, we attempt to summarize, in the form of a *wish list*, the desired features that an **approach** addressing requirements engineering for COTS-based systems should include or support. The proposed *wishes* are organized in four categories:

- ❑ Organizational-Management issues: aspects concerning the impact of using COTS on the organizational structure of a company. Techniques operating here may point out organizational changes that should be performed before other activities.
- ❑ RE Process issues: the impact of the COTS solution on the overall RE process. Typically this may involve an alteration both of the normal activities performed in the RE process for custom systems and addition of specific activity as well as process life cycle alteration, and so forth.

- ❑ Requirements contents issues: using a COTS solution can influence requirements acquisition, modeling and analysis. Here we refer to desired/required modifications of the way requirements are acquired, of which contents should be emphasized and which underplayed, how requirements should be specified and integrated with other specific information.
- ❑ COTS packages (or components) selection: selection of COTS package must be performed in coordination with the RE activities [4] and even more should be considered as part of the whole RE process. But which features the selection criteria and techniques should embody? How to integrate selection with RE and which factors should be considered?

3.1 Organizational-Management Issues

W1.1: Evaluating the opportunity of adopting a COTS solution (Management Issue). Before opting for a COTS-based solution, a complete approach should consider whether it is worth to follow this strategy or it is not feasible. COTS solutions are usually adopted to contain costs and to hit faster the market. Unfortunately the complexity of the system and the kind of market can make this choice worst than developing a system from scratch. There are several key management decisions to be considered before to opt for COTS software. In [5] a number of issues to be addressed are reported. Summarizing, most of them bring up the need for a systematic approach addressing these issues in the early stage of the system definition.

W1.2: Defining the business architecture and the software supply chain before or together to the software development process. This draws upon the considerations reported in [6]. Starting by these considerations, it can be argued as a particular attention should be put in defining an appropriate supply chain taking into account the business strategy. The approach should explicitly consider this factor since the very early stage of the development.

W1.3: Organization structure more flexible and open. In an approach for building custom systems, the feasibility analysis dictates the organizational structure of a company. The RE process and the following activities of the development life cycle, rely on this structure with some minor adjustments. In the case of COTS-based systems this structure can be strongly influenced by issues coming from the COTS market and products availability. For this reason, a suitable approach should consider a possible revision of the above structure, forcing its definition to be considered as part of the overall process. [6]

3.2 Requirements Engineering Process Issues

W2.1: Packages selection as an integrated and parallel activity since early requirements acquisition. It is widely accepted that COTS procurement activities must be interleaved with other traditional RE activities. Traditional approaches fail to support effectively these activities which should be both iterative and concurrent [7]. There are critical relationships among technology and product selection, requirement specification, and architecture definition. If the architecture is defined just to fulfill the requirements and then COTS products are selected, there will be only a few products (when any is available) that in some way fit within the chosen architecture. Pragmatically, three essential elements (requirements, architecture, and product selection) must be worked in parallel with constant trade-offs among them. Traditional approaches tend to confine the result of the RE process (as a whole) to the architecture definition while COTS product and suppliers are evaluated to better fit the defined architecture and to stay within the budget limits. Moreover, the evaluation is often performed through a trivial questionnaire sent to the potential supplies.

W2.2: Iterative life cycle process. Nowadays ISs are strongly interactive. For such systems it's proven the requirements engineering process must be iterative. Furthermore, for COTS-based systems, this need is felt twice since the intrinsic nature of the selection procedure requires an iterative process allowing a progressive reduction of candidate packages [7] [8].

W2.3: Process guidance and techniques integration. A high desired aspect of the approach is providing process guidance, even more due the actual lacking, in this direction, of current approaches [7] [8] [1]. Well-known techniques, already operating in the various facets of the RE field, may be included instead of re-thinking to new ones. The approach should enable the procurement team selects the most suitable techniques among various options, and organizes these in an iterative process. Some of these techniques should be used in the requirements activities, whilst others should perform a selection among the candidate packages.

W2.4: Integration of multi-criteria decision making techniques. The selection process must be systematic and well defined. Activities performed in this phase have to be interleaved within the overall iterative process. On the other hand, the techniques adopted in this phase have to be explicitly defined in order to record the rationale used in the decision of chosen COTS products. There is a range of techniques that can be used. Card sorts, for example, are simple to use and can acquire requirements that discriminate between products [10]. Two other approaches, among the nowadays most used, are AHP (Analytic Hierarchy Process) and WSM (Weighted Scoring Method), but they seem to work properly as quantitative criteria but to be inadequate to support qualitative reasoning. A further challenge is to keep the rationale of decisions made over the overall life cycle not only during the evaluation process [11].

W2.5: Integration of techniques to specify products evaluation to be used in matching requirements changes and evolution. One of the main problems is to extract from the commercial product description the information needed to make a selection against the acquired requirements. Once this effort has been performed we should prevent its wasting. These techniques should allow the procurement team records the rationale for product-requirement compliance to better react to requirements modification, addition and evolution. An example of these techniques can be found in [12]. Furthermore, the recorded rationale should refer to decisions made over the development life cycle not only during the evaluation process.

W2.6 Representing and storing the acquired selection knowledge. A considerable portion of the overall effort required to select among the available COTS solutions is due to the lack of an adequate knowledge of the specific market. For this reason, the approach should provide a technique first to represent and then to store this knowledge in order to be reused in future projects [1].

W2.7 Support Tool. A suitable tool should support the overall process. This tool should provide the procurement team with a support for all the above listed issues, including a feature for customising the process on the basis of the specific domain, team composition, time constraints, available techniques, and so forth.

3.3 Requirements Contents Issues

W3.1: Requirements Adaptation and balancing against COTS features. In a custom system, requirements are envisioned by the stakeholders' goals and on the basis of a set of specified constraints. In a COTS system, requirements can not neglect the availability of COTS products on the market, in that an available market may not exist to fulfill some requirements. This leads to requirements adaptation taking into account the knowledge of the existing market acquired little by little. Requirements elicitation and specification should support these adaptations [13].

W3.2: A more flexible requirements acquisition and specification. In COTS-based development, requirements statements need to be more flexible and less specific [14] [6]; otherwise it may be very hard to find out an appropriate package: products selection would be too strict or the amount of product modification would be so large that the COTS solution becomes not that worth anymore [15].

W3.3: Requirements specification should be structured in that tests cases can be easily performed since by the very early stages of the iterative process. Differently from a bespoke system in which test cases are used mostly to validate the developed application against the design in the latter stage of the product life cycle, for this kind of systems test cases can be used to select among the candidate components. Since the selection start by the very early interactions of the process, requirements are required to be well structured for test cases since there early phases [1]. A possible way to meet this desired facet of the RE process is to acquire requirements using use cases and scenarios techniques that make the requirements more amenable to test cases generation. For example in [16] an approach for decomposing goals into tasks which achieve these goals is proposed through generation of use cases that are equivalence classes of task scripts, and scenarios that are equivalence classes of use cases.

W3.4: Support of market evolution against requirements specification. One of the main issues in dealing with requirements for COTS-based systems is the impact of the evolution of used packages over the market [5]. This evolution impacts the development and maintenance of the system, therefore requirements should consider, for example, the supplier updating police as an additional selection criteria. If neglected, changes in COTS releases, competitive threats, stakeholders, reorganizations, and price structures may make requirements increasingly obsolete.

W3.5: Support of communication between the bespoke (custom) and COTS parts. A modern IS is made up both bespoke parts and COTS parts. Due their different nature, requirements giving rise to COTS parts and requirements giving rise to custom parts should be distinguished. On the other hand they are definitively related each other. These relations should be specified explicitly [5].

W3.6: Explicit consideration of the unused parts of each COTS-package. COTS package are more general purpose than requirements they answer [5]. The unused part can not be simply ignored because it may impact some functionalities and aspect of the system to-be. Requirements should handle the question of how to treat this unused part. This information will be of particular importance during the testing phase [17].

W3.7: Distinction between requirements used to select COTS packages and requirements not helping the selection. There are some requirements which in general are provided by all or most of the available package and other requirements which are very specific for the needs of a specific IS. In [9] this distinction has been pointed out calling them respectively "core" and "peripheral" requirements. Since one of the most important concerns of the RE for such systems is to define the procurement criteria, it's evident as for this task the former should be ignored, while the latter should be emphasized and acquired in more detail [18]. Anyway the approach should provide a way to discern between these requirements categories.

W3.8: Detailing Non-Functional Requirements for components selection. Since end-users are not in a position to specify functional requirements or to control the process of component development, there is no need for detailed functional requirements. As moving the focus from functional to non-functional requirements a number of topics, which should be addressed by the approach, come into light, as reported in [19].

3.4 COTS Packages Selection

W4.1: Direct consideration of adaptation costs for packages selection. Although glue-code development usually accounts for less than half of the total effort for the development of the COTS-based System software, the effort per line of glue code averages about three times the effort per line of custom applications code [5]. This consideration lead to adding the adaptation costs in techniques used to select packages. The distance between a package as sold and as ready to be integrated into the system should be a driver of the procurement process.

W4.2: Performing the decision of either buying or developing. For some parts of the system, adopting a COTS package isn't always the best choice. Such a decision should be evaluated during the COTS selection activity [5].

W4.3: Consideration of contract aspects for packages selection. Non-development costs, such as licensing fees, are significant and the procurement process must optimize them. SEI identifies three significant CBS activity areas: vendor relationships, license administration, training and cultural transition [20]. All these costs can significantly impact the worth-ness of a COTS solution instead of another one.

W4.4: Using some kind of weighted metrics to evaluate package compliance against functional and non-functional requirements. Lack of such metrics makes very hard and ineffective the product selection activity. Fit criteria should be expressed in terms of logical expressions or quantifiable tests to undergo commercial requirements standards [8]. A possible technique is repertory grid analysis [21], in which stakeholders are asked for attributes applicable to a set of entities and values for cells in an entity-attribute matrix. These metrics may weight a number of factors as costs, supplier credibility, contract forms, volatility of the packages on the market, required adaptation effort, adheres to current product standards, integration level, communication required against other packages, and so forth. Therefore, the approach should define a distance or ratio scale to be used for obtaining criteria scores in evaluating different COTS products.

W4.5: Stakeholders involvement in the product evaluation. Techniques used to select components among the possible ones should directly involve stakeholders [8] to further elicit requirements or to assess those already acquired reaching a deeper detail level.

W4.6: Minimization of independent COTS products. COTS-based system development and post deployment efforts can scale as high as the square of the number of independently developed COTS products targeted for integration, because integrating n COTS products involves potentially n(n − 1)/2 interfaces . The conventional wisdom in the use of COTS components is the more of the system that can be built using COTS components, the better. Beyond a certain point, however, an increase in the number of COTS components in a system may actually reduce the system's overall economic life span rather than increase it [22]. Taking in consideration these observations, the selection criteria should aim, among other things, to minimize the number of packages and vendors that are going to build the system.

W4.7: Support Tool. A suitable tool should support the selection activity. This tool should provide the procurements team with a support to all the above listed issues, that is, it should include a metric system, consider package costs and contract aspects, allow strategy definitions, allow an iterative process, store the acquired knowledge.

4 An Empirical Evaluation of Current Approaches

In this section we show the results obtained by matching some of the existing approaches against the wishes previously listed. The selected approaches are: RUP (Rational Unified Process) [24], OTSO (Off-The-Shell Option method) [17], MBASE (Model Based Architecting and Software Engineering) [25], CAP (COTS Acquisition Process) [26], PORE (Procurement Oriented Requirements Engineering) [4,18], CARE (COTS Aware Requirements Engineering) [1]. Information allowing this comparative study have been acquired from the literature currently available about every approach.

For each combination wish/approach we give an evaluation of how much this wish is satisfied by that approach, rating by *Poor* (either the satisfaction level is not clear or there are just some ideas concerning the wish topic), *Sufficient* (the approach takes into consideration this wish but just partially), *Complete* (the approach seams satisfying that wish completely).

The previous table can be explored both by row and by column drawing out some considerations about, respectively, the satisfaction level of a specific wish throughout all approaches, and the response of specific approaches in terms of the recognized wishes. In the following, some examples of this kind of analysis are reported:

Exploring by column:

- ❑ The RUP approach does not generally accomplish most of the desired wishes. Mostly it is due to the intrinsic nature of this approach, because, even if RUP provides support to include COTS components in a system, it operates at the logical design level of the system. From the table, it can be claimed as RUP totally satisfies the wish of having a support tool (that is Rational Rose™) and the generic characteristics of a modern process, but is weak when dealing with specific RE issues.
- ❑ CAP is particularly weak in dealing with the integration of the requirements acquisition to the COTS selection activity. This is obvious since it starts by an already acquired requirements base to select suitable COTS package. This aspect, combined with some other lacks, makes this approach incomplete.
- ❑ By comparing the last two columns each other and against the rest, it can be noticed as the last two approaches appear to be a bit ahead since they satisfy several wishes. Furthermore, it can be noticed as CARE strengthens PORE, since it is generally stronger in all wishes, as the same authors asserted "The CARE approach draws upon the good available ideas in current RE methodologies including RUP, MBASE, and PORE" [1].

Table 1. Satisfaction Level of wishes in analysed methods

	RUP	OTSO	MBASE	CAP	PORE	CARE
W1.1	Poor	Poor	Poor	Poor	Sufficient	Poor
W1.2	Poor	Poor	Poor	Poor	Poor	Sufficient
W1.3	Poor	Poor	Poor	Poor	Poor	Poor
W2.1	Poor	Complete	Poor	Poor	Sufficient	Complete
W2.2	Poor	Poor	Poor	Poor	Poor	Poor
W2.3	Sufficient	Sufficient	Poor	Sufficient	Complete	Complete
W2.4	Poor	Complete	Sufficient	Sufficient	Complete	Complete
W2.5	Poor	Sufficient	Sufficient	Sufficient	Sufficient	Sufficient
W2.6	Poor	Poor	Sufficient	Sufficient	Complete	Complete
W2.7	Complete	Poor	Poor	Poor	Poor	Poor
W3.1	Poor	Sufficient	Poor	Poor	Poor	Complete
W3.2	Poor	Poor	Poor	Poor	Complete	Complete
W3.3	Poor	Poor	Poor	Poor	Complete	Complete
W3.4	Poor	Sufficient	Poor	Poor	Sufficient	Sufficient
W3.5	Complete	Poor	Poor	Sufficient	Poor	Sufficient
W3.6	Poor	Poor	Poor	Poor	Poor	Poor
W3.7	Poor	Sufficient	Poor	Poor	Complete	Complete
W3.8	Poor	Sufficient	poor	Poor	Poor	Sufficient
W4.1	Poor	Sufficient	poor	Poor	Poor	Sufficient
W4.2	Poor	Sufficient	poor	Poor	Poor	Sufficient
W4.3	Poor	Complete	poor	Complete	Sufficient	Sufficient
W4.4	Poor	Complete	poor	Complete	Poor	Sufficient
W4.5	Poor	Poor	poor	Poor	Complete	Complete
W4.6	Poor	Poor	Poor	Poor	Poor	Poor
W4.7	Complete	Poor	Poor	Poor	Poor	Poor

Exploring by row:

❑ It appears evident as all wishes dealing with the first group, that is Organizational-Management issues are generally neglected.

❑ The approaches have a very variable behavior in respect of *selection technique* wishes, some ones are stronger for some wishes and some others are better with other wishes. By this consideration, inspecting all the approaches and extracting respective strengths, some enhancements could be performed.

❑ By looking the row belonging to W3.6 it can be argued as all approaches ignore the unused parts of used COTS packages, although in [23] it's claimed as this behavior may lead to unexpected consequences in the final system.

5 Concluding Remarks and Call for Research

The inspection of the research literature concerning COTS-based systems and in particular RE for such systems, brought us to draw the current situation in this field that we can describe as composed by a number of punctual contributions as well as complex approaches composed by a number of sub-activities in charge of defining the system to be. A first interesting consideration, raised out by this analysis, is that a considerable part of the existing approaches neglects the requirements problem, providing just some features to specify COTS components in designing the system. A further contribution of this research has been to determine the impact of using COTS packages on the requirements engineering activities and process. Finally, the main contribution of this research has been to recognize and describe, in the form of a wish list, a number of desired characteristics of a suitable approach in charge of defining a system including a more or less considerable part composed by COTS packages. Moreover, the existing approaches have been reconsidered against this list, allowing a recognition of what has been already accomplished and what is desirable for future researches.

In particular, this can be translated in some concise call-for-research:

Call1: The research has definitively shown as the UML community is lacking of an approach considering COTS since the requirements engineering phase. This is clear by examining the unique contribution treating COTS with UML, that is [24], where COTS are considered only during the design phase of the system.

Call3: The selection strategy, embodying all the aspects described in paragraph 3.4, should be definable and customizable so as to adapt the method to the specific application case.

Call4: All the existing approaches show an evident lack in supporting the requirements activities for such systems by means of tools. This is recognized as a main factor that tends to enlarge the already existing gab between the research community and the industrial sector. This is because tools usually allow a reduction of the exploitation time and attract people that otherwise should perform a number of activity manually.

References

1. L. Chung and K. Cooper: "A Knowledge-Based COTS-Aware Requirements Engineering Approach". In proceedings of 4th Int. Conf. on SEKE'02, ACM Press. July 15-19, 2002, Ischia, Italy.
2. Morisio M., Torchiano M.: "Definition and classification of COTS: a proposal". In Proceedings ICCBSS, Orlando (FL) February 4-6, 2002.
3. D. Carney, F. Long: "What Do You Mean by COTS?". IEEE Software, March/April 2000, pp. 83-86
4. C. Ncube and N. Maiden: "Guiding parallel requirements acquisition and COTS software selection". In proceedings of the IEEE International Symposium on Requirements Engineering 1999.
5. Jeffrey Voas: COTS Software: "The Economical Choice?". IEEE Software, 15(2):16-19, Mar 1998.
6. Farbey, B, & Finkelstein, A.: "Software Acquisition: a business strategy analysis". In proceedings of Requirements Engineering 2001 (RE01).
7. L.Brownsword, D.Carney, T.Oberndorf: "The Opportunities and Complexities of Applying Commercial-Off-the-Shelf Components". SEI Interactive, 6/98, 1998.
8. N. Maiden and C. Ncube: "Acquiring COTS Software Selection Requirements". IEEE Software, Volume 15 Issue 2, March-April 1998, pp. 46-56
9. Finkelstein, A., Spanoudakis, G., and M. Ryan: "Software Package Requirements & Procurement". In proceedings of the 8th Int. Workshop on Software Specification & Design,IEEE CS Press, 1996.
10. N.A.M. Maiden and G. Rugg: "ACRE: Selecting Methods for Requirements Acquisition". Software Eng. J., Vol. 11, No. 3, 1996, pp. 183-192
11. Carina Alves, Anthony Finkelstein: "Challenges in COTS decision-making: a goal-driven requirements engineering perspective". In proceedings of SEKE 2002: 789-794
12. T.P. Moran, J.M. Carroll: "Design Rationale: Concepts, Techniques, and Use". Lawrence Erlbaum Assoc., Hillsdale,N.J., 1996
13. C. Alves and A. Finkelstein: "Negotiating Requirements for COTS-Based Systems". In proceedings of 8th Int. Workshop on Requirements Engineering: Foundation for Software Quality, in conjunction with RE'02.
14. D. Carney: "COTS Evaluation in the Real World". SEI Interactive December 1998.
15. D. Carney: "Requirements and COTS-Based Systems: A Thorny Question Indeed". SEI Int. June '99.
16. I. Graham: Task Scripts: "Use Cases and Scenarios in O-O Analysis". O-O Systems 3, 1996, pp.123-142
17. Kontio, J.: OTSO: "A Systematic Process for Reusable Software Component Selection". TR, Dec. 1995.
18. N. Maiden, C. Ncube, and A. Moore: "Lessons learned during requirements acquisition for COTS systems". Communications of the ACM, Vol. 40, no. 12, 1997
19. L. Beus-Dukic: "Non-functional requirements for COTS software components". In proc. of ICSE '00
20. Basili V. R. and Boehm B.: "COTS-Based systems Top 10 List". IEEE Computer, vol. 24,5 May '01.
21. Shaw M.L. and Gaines B.R.: "Requirements Acquisition". Software Engineering Journal, 1996, 11 (3)
22. Tumuluri S., Raja S., and Cooper K.: "Commercial off-the-Shelf (COTS) Software Engineering Methodologies: A Comparative Study". T.R. UTDCS-24-01, December 2001

23. M. Feather: "Language Support for the Specification and Development of Composite Systems". ACM Trans. on Programming Languages and Systems 9(2), Apr. 87, 198-234.
24. G. Booch, J. Rumbaugh, and I. Jacobson: "The Unified Modeling Language User Guide". Addison Wesley Longman, Inc., USA, 1999.
25. Boehm, B: "Requirements that handle IKIWISI, COTS, and Rapid Change". IEEE Computer, Volume: 33 Issue: 7, July 2000.
26. Ochs, M.A. et al.: "A Method for Efficient Measurement-based COTS Assessment ad Selection – Method Description and Evaluation Results". In proc. IEEE 7th International Software Metrics Symposium.

From System Requirements to COTS Evaluation Criteria

Grace A. Lewis and Edwin J. Morris

Software Engineering Institute, COTS-Based Systems Initiative
4500 Fifth Ave.
Pittsburgh, PA 15213
U.S.A.
{glewis,ejm}@sei.cmu.edu

Abstract. Perhaps the most common question asked by organizations new to constructing COTS-based systems is how to choose the right product. In particular, they want to know how, starting from system requirements, an efficient and effective set of criteria for COTS selection can be developed. This paper focuses on recommendations and techniques for transforming a set of requirements into a set of product evaluation criteria that capture the facts and standards by which the fitness of COTS products can be judged.

1 Introduction

The starting point for virtually all software development efforts is a set of requirements that represent at some level the expectations of those that have a stake in the system. If the organization is considering the use of COTS products as system components, it is natural and appropriate to look first to these requirements as a basis for evaluating and selecting the right products. For some organizations, an initial focus on system requirements leads them to believe that generating the actual criteria for COTS evaluation from system requirements is trivial—each requirement is directly transformed into a criterion. However, our experience suggests that this simple transformation is not likely to achieve the desired result—selection of an appropriate COTS product—because:

- System requirements are normally written at an abstract level in order to allow sufficient flexibility for choosing multiple technical solutions. Unfortunately, criteria derived directly from such requirements are too abstract to serve as a way of evaluating products. For example, a requirement such as "The system shall be easy to use" is too abstract to judge if a product is or is not easy to use.
- System requirements are stated in terms of needs, whereas criteria should be stated in terms of capabilities to satisfy those needs. For example, a requirement such as "Information transfer shall be protected from unauthorized access" might be transformed into a criterion such as "Support for secure sockets or equivalent security mechanism", which is the expected concrete capability.
- Criteria should be obviously quantifiable whereas system requirements are often stated in a way such that the manner in which they are measurable is not immedi-

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 159-168, 2004.

ately obvious. Criteria should include the way in which the expected capability is to be determined in order to facilitate the evaluation process.

- System requirements tend to be incomplete, hardly ever stating every expectation placed on the COTS product. Often qualities of the component other than required functionality are overlooked. For example, a system requirement regarding the level of effort necessary to apply appropriate tailoring to new releases of a product is rare.

The focus of this paper is on recommendations and techniques for transforming a set of requirements into a set of product evaluation criteria that can be applied in a COTS product evaluation and selection process. A description of an evaluation process is available in work performed by the Software Engineering Institute and the National Research Council Canada [1].

2 Defining Evaluation Requirements for COTS Products

Because the requirements placed on a COTS product are not just a subset of the system requirements, the first step in defining evaluation criteria is to establish *evaluation requirements*. Due to the fact that COTS products rarely align exactly with anticipated system functions (some products combine functions, use different vocabularies, etc.), the mapping between product functions and system functional requirements may not be obvious or straightforward. In addition, COTS products add an entirely new class of concerns regarding licenses, testing, rapid deployment, and control of the content of upgrades that are often not addressed by system requirements.

Examples of legitimate evaluation requirements that are not always addressed by system requirements include:

- Architecture/Interface constraints: COTS product decisions are often constrained by other decisions that have already been made. These constraints become evaluation requirements when choosing a COTS product. For example, if a decision has been made to use a certain middleware mechanism, it makes little sense to buy a product that is clearly going to conflict with this technology.
- Programmatic Constraints: Time, money, available expertise, and many other programmatic factors may be sources of evaluation requirements that are not captured in system functional requirements. For example, availability of trained staff adept at using a product can be a useful criterion when choosing products, but is not likely to appear in system requirements.
- Operational Environment: Not all aspects of the operational environment are included as system requirements. For example, information about the organization that will perform maintenance on the system is frequently omitted. Thus, the operational environment may be a source of additional evaluation requirements.
- Stakeholder expectations: People place additional expectations on products that are not clear from system requirements. For example, users may have a strong preference about the style of the user interface to a product. These expectations are often not captured in system requirements, in part because it is assumed that the user interface can be tailored to expectations. However, the user interface of a COTS product is not as flexible as that provided by custom code. Therefore, it is more important to determine the acceptability of the interface.

2.1 Sources of Additional Evaluation Requirements

Several sources are available that provide evaluation requirements that are not system specific[1]. In general, such evaluation requirements will not address the full range of system-specific expectations for COTS products. There are multiple sources for this type of evaluation requirements, such as:

- Product feature checklists
- Organizational checklists
- Previous evaluations for other systems
- Marketplace

Product feature checklists are a standard fare for product comparisons. While they are only as reliable as the source, they are widely used—and misused—in COTS software evaluation. The basic capability provided by a product feature checklist is an itemized list of the features of a class of products. This list is then used as a basis for comparing different products. In some market segments, organizations have formed a niche market by identifying a set of generally preferred features and characteristics, and evaluating products against this set. The most notable example of this approach is *Consumer Reports* [2], which monthly evaluates consumer items and provides recommendations for preferred products. However, in the COTS software domain, the context in which the software will be used is not nearly as consistent as the various contexts for which *Consumer Reports* makes recommendations. Thus, an effective *Consumer Reports* for the COTS product marketplace is not likely in the near term. Some firms, however, regularly assess products that make up particular market segments; examples of these are Gartner [3] and Ovum [4]. In addition, trade magazines often publish product feature checklists. One should guard against accepting at face value product feature checklists that appear in trade magazines. These are sometimes produced by vendors specifically to present their product in best light or to highlight some feature that distinguishes their product from those of competitors. It is also important to keep in mind that no generic list can reflect all of the requirements that need to be considered—a generic list will include some features that are irrelevant for the system and other important features will be missed. These lists should not be used as the only evaluation requirements for COTS products.

Other sources of "reusable" evaluation requirements are organizational checklists that represent a consistent way to insure that corporate interests are addressed in the evaluation and selection of COTS products. Organization checklists may maintain relevant information about corporate policies, preferred architectures and expected engineering practices and can provide some uniformity and predictability in selecting products.

In general, corporate need for such checklists is not driven solely by technical considerations: many other concerns are reflected as well, such as legal considerations (e.g., "due diligence" in managing financial assets) and issues of scale (many organizations, many projects, many evaluators). Table 1 identifies the categories of criteria covered in an organizational checklist built by a large aerospace contractor. In addi-

[1] ISO 14598 includes the concept of "evaluation modules" that are reusable packages containing requirements and other data about an evaluation. In theory, these modules can be reused in different settings. This approach is similar to the reuse of evaluation requirements described here.

tion to the requirements covered in these categories, the actual checklist provides a 70-step process for evaluation and selection.

Table 1. Example of an organization checklist

Organizational Checklist for COTS Products
Compatibility with other COTS products in use
Adaptability, flexibility, reliability, maintainability
Impact on system integrity
Impact on system integration
Vendor support
Training
Documentation
Licenses

Evaluations that the members of the team (or the wider organization) have performed for similar purposes are also sources of evaluation requirements. While these requirements are likely to reflect some of the context-specific nature required, it is a common mistake to consider them equally appropriate for the new evaluation context as they were for the old. This mistake is reflected in statements such as: "We looked at Product X before and it was pretty good. It will be good for the new system also". A variant of this problem is represented by selecting "pre-approved" products from a product list without further analysis. In essence, such choices indicate that the requirements for COTS products in "this" system are sufficiently close to that of a "benchmark" system and therefore no product evaluation process is warranted—a very bold and risky position.

Finally, a source for evaluation requirements is the marketplace itself. Risk-Driven Generation [5] is a technique that looks at what the COTS marketplace can offer and focuses on risk introduced by or reduced by product and vendor features[2]. This approach derives a set of potential evaluation requirements from the features of products rather than from system requirements. This set of requirements is pruned by analyzing the risk to the system should the product provide or not provide a particular feature. The steps involved in Risk-Driven Generation are straightforward:

1. Identify interesting product features
2. Assert risk to the system of a product not exhibiting a feature
3. Categorize and quantify risk (optional)
4. Identify mitigations (optional)

The overall effect is that expectations regarding system needs are "adjusted" to agree with product capabilities. By focusing on risk, the approach also helps to combat "gold plating" of requirements and "featuritis". If little or no risk results from absence of a product feature, then the feature is not required of products under consid-

[2] The Risk Driven Generation approach identified in [5] refers to criteria. However, the use of the term *criteria* in the document is consistent with our use of the term *requirement*.

eration. Note that this approach may also generate negative requirements—features that the product should *not* have.

2.2 Classes of Evaluation Requirements

A common problem when evaluating COTS products is to treat the majority of requirements as providing no leeway for negotiation. Stakeholders are the common source for this problem because they are most familiar and comfortable with the processes, user interfaces, and capabilities provided by existing systems, and have high expectations that the new system will provide all of these plus the latest “wiz-bang” technologies. This is almost inevitably a mistake because virtually all COTS products are compromises suited to multiple organizations and will do things differently than an individual organization. In the most severe case, this approach will render all COTS products unacceptable, since none can meet the expectations and specifications[3] desired by users. If the organization is committed to using COTS products, it should strive for achieving sufficient flexibility in requirements. In reality, identified evaluation requirements will likely fall within a spectrum between two general classes or categories:

- *Hard requirements* that are mandatory—if the product does not meet these requirements, then the product or the system must be augmented in some way or the product is unsuitable. The augmentation could take the form of custom code that handles functionality the COTS product does not provide or the selection of another product that provides the missing capability.
- *Negotiable requirements* that are flexible—if a product does not demonstrate the preferred characteristic, then it is not automatically excluded. The options in this case are to adjust the requirement to some degree or to augment the product or the other components in the system to address the requirement.

We expect that the majority of requirements fall somewhere towards the center of a spectrum between totally inflexible and completely flexible, where there is some room for negotiation but in the end the requirement (or some version of it) must somehow be met. COTS products often achieve similar results through somewhat different means than existing, custom-built systems. In many cases, the alternate implementation expressed by the COTS product is acceptable. This has even been shown in several cases involving requirements that were thought to be controlled by legislative mandate (laws). In these cases, the driving force behind the requirement was the "traditional" approach of manual processes or legacy systems, rather than the actual letter of the law that later proved to be sufficiently flexible to allow alternate implementations. Several state and federal organizations have suggested to the SEI that many laws have more leeway than people who have grown accustomed to current procedures tend to believe. In all cases, the organization should carefully research any perceived legislative mandates to determine whether the letter of the law (rather than some traditional interpretation) is the limiting factor.

[3] In some cases, where organizations are dead set against the use of COTS products, this approach is often intentional.

2.3 Errors Compiling Evaluation Requirements

There are two types of errors that can occur when compiling evaluation requirements: errors of inclusion and errors of exclusion.

Errors of inclusion are caused when non-applicable requirements are included in the evaluation requirement set. Errors of inclusion have two unfortunate effects. First, they tend to reduce the total amount of time and effort that is focused on the evaluation of the product against necessary and important requirements. Thus, the organization may focus its evaluation efforts on determining whether a product supports a feature of questionable value, while spending less effort determining whether a product's vendor can appropriately support the product in a certain context. Second, errors of inclusion can eliminate suitable COTS products simply because they do not support the unnecessary and unimportant "requirement". Perhaps the best example of an error of inclusion is the tendency to want every "cool" capability—even if the capability does not add any appreciable value in the present context. A good technique to combat this tendency is to consider the risk to the system mission should the feature be absent or provided in a different manner. If there is no risk or it is minimal, then the requirement is unnecessary.

Errors of exclusion are the opposite and involve omitting requirements that are crucial to the fitness of the product and vendor within the system context. The main danger of errors of exclusion is that they can lead to the selection of an unsuitable product. For example, neglecting to consider the vendor's ability to support a product within a certain context may lead to the selection of an unsupportable product. Errors of exclusion are often caused by insufficient understanding and an oversimplification of the problem. An iterative approach to building COTS-based systems can help mitigate this risk. As understanding about the problem grows, it is almost inevitable that requirements that were initially overlooked will be identified.

3 Defining Evaluation Criteria

The second step in defining criteria for product evaluation is to take the evaluation requirements and convert them into evaluation criteria. A good criterion for evaluation and selection of a COTS product consists of two elements:

1. A clearly measurable statement of the capability necessary to satisfy a need—called a capability statement
2. A means for assessing and assigning a value to the product's level of compliance with the capability— called a quantification method.

A good criterion needs both the capability statement and a quantification method. However, in some cases the details of the quantification method may not be fully known when the criterion is first defined. In this case, these details must be filled in as they become known. Table 2 contains an example of a fully defined criterion for the evaluation of vendor support.

Table 2. Example of a requirement, capability statement, and quantification method

Requirement
- The COTS vendor shall provide support for the COTS product.

Capability statement
- COTS vendor support shall include
  - 24x7 help desk
  - On-site installation/training support
  - On-line error reporting

Quantification method
- A product support survey will be provided to potential COTS vendors. Support claims will be verified by exercising help facilities where possible and by contacting current product users to determine the quality of vendor support"

Some additional characteristics of a good criterion include:

- Assessable: If data cannot be gathered to support the quantification method selected, then the criterion is not good. For example, "quality of engineering" is not a good criterion if there is no way to gather data indicating whether the product is well architected, designed, and implemented.
- Discriminating: If all COTS products display a required characteristic, then the characteristic is not useful in identifying which products are superior. For example, the presence of a graphical user interface will not normally discriminate between modern word processors.
- Non-overlapping: If criteria overlap, then the associated product characteristics can be factored into deliberations multiple times. This can lead to misleading results and possibly wasted effort.
- Significant: If a criterion is not valuable in the context of the system then it should not be used. For example, the long-term stability of a vendor is irrelevant if the product will only be used as a short-term or interim solution.

3.1 Tips for Generating Good Criteria

Perhaps the most common approach to generating criteria is through decomposition of higher level expectations into more detailed criteria. Starting with a list of high-level expectations (e.g. ISO standard on product quality [6]), important expectations are decomposed into increasingly refined expectations, eventually resulting in something like a capability statement for a detailed criterion. At this point, a quantification method is added, completing the criterion.

Goal Question Metric [7] provides a technique to develop both a hierarchical decomposition of requirements into capability statements and associated quantification methods. With GQM, the user develops a triad containing:

- The goal, which is to fulfill an evaluation requirement.
- The question, which is a capability statement.

- The metric, which is the standard of measurement, determined either by direct measurement or decomposition into other GQM triads.

GQM has been adopted by a number of product evaluation techniques, including PORE [8] and OTSO [9].

3.2 Defining Quantification Methods

Quantification in the software domain is never easy, particularly when measurements use very different scales which only indirectly get at the salient characteristics, or involve products with which we have limited familiarity. However, the goal for quantification remains to produce fair, unbiased results, and to frame those results in a way that supports our ability to reason about them.

Sources of ideas about quantification methods abound. To quantify whether a COTS product achieves a functional capability, simple manipulation of an individual product feature may be adequate. However, to determine whether the feature will be useful in the way intended for the system, quantification may involve the execution and scoring of performance on a number of scenarios that reflect the actual system. For non-functional (quality) attributes, texts on software metrics [10] provide a number of ways to measure characteristics like reliability and quality. For example, the *COTS Acquisition Process* (CAP) created by Siemens provides an approach for decomposing requirements expressed as quality attributes and generating capability statements and quantification methods [11]. Quality attributes were identified from ISO 9126 and through literature review and interviews with experts.

To achieve the *harder* goal of producing fair, unbiased results, techniques must be found to consistently produce measurements that accurately reflect the characteristics of the product and vendor, and to integrate these results into a comprehensible picture. Results from measurement of COTS products are either qualitative (e.g.., poor, fair, good) or quantitative (e.g., cost, maximum entries in a database). Qualitative measurements normally reflect different perceptions of multiple evaluators on subjective rating scales. Strong guidelines for assigning ratings must be enforced to achieve the goal. Quantitative measurements have an aura of concrete and scientific about them, but can be systematically or randomly biased or otherwise invalidated. For example, a single run of a benchmark may involve slight variance in the timing of input data that can significantly alter the performance measured. This could lead to a false impression that one product is superior to a second, when in the vast majority of cases, the second is actually superior. Consistent test environments, sufficiently accurate tests harnesses and measurement scales, and multiple measurements are required to meet the fair/unbiased goal.

In order to integrate the various measurements into a comprehensible picture, the data from execution of the multiple tests must be converted to a consistent scale. This process is called “normalization”. Data is normalized by phrasing all quantification methods in terms of the same scale (e.g., performance of a certain level is considered poor, another level fair, and another level good) or by converting measured scores to the consistent scale.

A serious mistake some make in the normalization process involves converting ordinal data (only the order is significant) to interval data (the interval between units is equivalent). Converting the ordinal scale “poor, fair, good, excellent” to “1, 2, 3, 4” may lead one to believe that excellent is twice as good as fair. However, this is not true for ordinal data and trying to introduce operations using ordinal data (like determining the mean) is mathematically unsupportable.

4 Establishing Priorities

The third and final step in defining evaluation criteria is to prioritize the defined criteria. Normally, some characteristics of a COTS product or vendor are more important than others. In order to reflect this relative importance, priorities must be assigned to criteria to allow reasoning about products that have different strengths and weaknesses.

Priorities are themselves not simple, and in reality may reflect a number of different concerns. Some of these concerns are:

- How relevant is a criterion vs. other criteria?
- How expensive is it to provide the capability in a different way than with the product?
- What are the risks to the system if a particular criterion is not met?

A common way of expressing priority is by assigning weights to the criteria. Weights express the relative importance of normalized values of different criteria, answering questions such as, "How many units of this criterion would we trade for a unit of this other criterion?"

A variety of techniques can be used to assign weights to criteria. All involve making a judgment based on understanding of the system:

- Unstructured weighting: one or more people determine weights based on their experience and common understanding of the system. This is probably the most popular method, but not necessarily the best.
- Delphi technique: individuals use their own approach for deriving initial weights—the Delphi technique helps the team converge on a single weight [12] [13]. This is a popular method for gaining consensus in a team.
- AHP Pairwise comparison: the judgment is performed by comparing pairs instead of whole sets of criteria [14]. Criteria are ordered in pairs, and the team agrees on the relative importance of the criteria in each pair. An AHP tool can be used to compute a weight for each criterion from the total set of pairwise comparisons.

Once the criteria are prioritized, data from the different products can be collected and looked at in light of these criteria. A measurement plan will have to be established so that data for all criteria can be gathered efficiently.

5 Summary and Conclusions

One of the important steps in COTS product evaluation is the generation of evaluation criteria, that is, the facts and standards by which the fitness of a product will be judged. System requirements provide a starting point but rarely identify the complex set of evaluation criteria. A way to generate criteria from system requirements is to first determine the evaluation requirements, then define criteria from these requirements, and finally prioritize the criteria. Once criteria are defined, products can be looked at in light of these well-defined criteria and evaluated for suitability within the system context.

Several techniques were suggested in this paper and many more are available. To get started in your organization you should tailor the process by selecting techniques that make sense for the type of system you are building and the expectations for the components to be selected. You can expect to learn from every evaluation you perform. With time you will learn more about the process, the criteria, and how to evaluate products against the criteria.

References

1. Comella-Dorda, S., Dean, J., Harper, E., Lewis, G., Morris, E. & Oberndorf, P. *A Process for COTS Software Product Evaluation.* (CMU/SEI-2003-TR-017/ESC-TR-2003-017). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, October, 2003.
2. Consumer Reports. Information available online: http://www.consumerreports.org/.
3. Gartner Group. Information available online: http://www.thegartnergroup.com/.
4. Ovum. Information available online: http://www.ovum.com/.
5. Wallnau, Kurt; Hissam, Scott; & Seacord, Robert. Building Systems from Commercial Components. New York, NY: Addison-Wesley, 2001.
6. International Organization for Standardization (ISO). ISO/IEC 9126:1991 – Information Technology – Software Product Evaluation – Quality Characteristics and Guidelines for Their Use. Geneva, Switzerland: ISO/IEC, 1991.
7. Briand, L.; Morasca, S.; & Basili, V.R. Goal-Driven Definition of Product Metrics Based on Properties. (CS-TR-3346, UMIACS-TR-94-106) University of Maryland, Computer Science Technical Report, December 1994.
8. Ncube, C. & Maiden, N. A. M. PORE: Procurement Oriented Requirements Engineering Method for the Component-Based Systems Engineering Development Paradigm. Proceedings of the 2nd International Workshop on Component-Based Software Engineering (held in conjunction with ICSE '99). Los Angeles, CA: 1999.
9. Kontio, J. A Case Study in Applying a Systematic Method for COTS Selection. Proceedings of the International Conference on Software Engineering. Berlin, Germany: 1996.
10. Fenton, N.E. Software Metrics: A Rigorous Approach. Chapman & Hall, London, 1991.
11. Ochs, M., Pfahl, D., Chrobok-Diening, G. & Nothhelfer-Kolb, B. *A COTS Acquisition Process: Definition and Application Experience.* IESE-Report No. 008.00/E Version 1.0. Kaiserslautern, Germany: Fraunhofer Institut Experimentelles Software Engineering. February 2000.
12. Dalkey, N.C. & Helmer, O. "An Experimental Application of the Delphi Method to the User of Experts." Management Science, 9, 3 (April 1963) 458-467.
13. Linstone, H.A. & Turoff, M. The Delphi Method: Techniques and Application. New York, NY: Addison-Wesley, 1975.
14. Saaty, T.L. The Analytic Hierarchy Process. Mc-Graw Hill. New York, 1990.

Empirical Analysis of COTS Activity Effort Sequences

Dan Port[1] and Ye Yang[2]

[1] University of Hawaii, Honolulu
Hawaii, USA
dport@hawaii.edu

[2] Center for Software Engineering, University of Southern California, Los Angeles
90089 California, USA
yey@cse.usc.edu

Abstract. Empirical data has revealed that COTS based application (CBA) development lifecycles are unique and differ from traditional software development processes. Each project will vary considerably in the particular amount of effort expended on COTS assessment, COTS tailoring, and COTS glue-code development. As such, there are wide variations in cost/schedule/quality factors, risk items, and project decision process profiles. Previous work has described these variations and provided a composable COTS decision framework that models how such variations emerge within the COTS application development process. We expand on this work by elaborating the sequence in which COTS assessment (A), tailoring (T), glue-code (G), and custom development (C) activities are performed. These sequences provide a "genetic code" for a CBA development project and are useful in characterizing the uniqueness of a CBA development lifecycle and enabling tactical decision support such as identifying high-risk development patterns, effort and schedule planning, options valuation. Other applications include analyzing the effects of risks on COTS activity and validating the composable COTS decision framework motioned above. We present CBA activity effort sequences from 9 USC e-service projects and their relationship to reported project risks. In addition, we present an initial set of "anticipated" and "un-anticipated" CBA effort sequence patterns. Anticipated sequences are patterns that have been observed within our case studies and are rationalized with respect to COTS risk-reduction on a project. One would expect to see such patterns on a well-managed, successful CBA development project. Unanticipated sequences are patterns that have not been observed, or when they have been observed, have correlated strongly with high risk and perhaps project failure.

1 Introduction

The activities conducted while developing COTS based systems substantially differ from those conducted for non-COTS systems (David Carney's "COTS Spot" articles highlight this phenomenon frequently [9]). This has also been empirically verified in the authors previous works on COTS based applications (CBA) – appli-

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 169–182, 2004.

cations developed where at least 30% of the end-user functionality is provided by COTS products, and at least 10 % of the development effort is devoted to COTS considerations [5,6]. In these works, data collected from 19 COCOTS [1] projects and 21 e-services projects developed at the University of Southern California indicated that there are three distinct types of COTS engineering activities – assessment, tailoring, and glue-code development. We furthered this research by describing a recursive and re-entrant CBA process decision framework within these three activities based on the risk driven spiral model [3] abstracted from the CBA projects. The CBA process decision framework consists of dominant decisions and activities within CBA development to enable developers to "compose" a COTS development process specifically tailored for their project. Throughout this paper, we shall refer to major COTS related activities activity areas - assessment, tailoring, and glue-code - with the notations A, T, G respectively. Furthermore, if the project objectives, constraints, and priorities (OC&Ps) cannot be satisfied wholly with COTS, the project may perform some custom development (C) in addition to a COTS solution. The project can enter or re-enter any of these activities during the project development cycle. These activities may not be sequential. Two activities such as tailoring and glue-code may be implemented in parallel or perhaps not at all within any particular development cycle.

Many of the successful USC CBA e-service projects had made use of the CBA process decision framework. Owing to this, we observed the duration and the sequence in which each project performed A, T, G, C activities. Each project has a distinct pattern with respect to the overall effort expended in particular COTS activities [5], and Remarkably, the sequence in which COTS activities are performed [6]. In this paper, we investigate the possibility that a CBA project's A, T, G, C activity sequence may serve as a "genetic code" for analyzing and characterizing its generally unique COTS development process. CBA development is fraught with complex and difficult to manage risks, and an analysis and characterization of CBA activity sequences may serve to:

- Identify and avoid high risk development patterns
- Aid in COTS effort planning, monitoring, and control [1]
- Help explore COTS development options and rationalize COTS decision making
- Provide evidence to further validate the composable CBA decision framework [6]
- Help illuminate the COTS risks and risk management within the A,T,G COTS development activities

This paper provides an analysis of these characterizations with respect to risks that we have identified in our experience of implementing CBA projects at USC. In the next section, we begin with an overview of issues in COTS development process.

The rest of the paper is organized as follows. In section 3 we summarize the salient points of the CBA process decision framework and how it relates to the A,T,G,C activity sequences. Section 4 elaborates the activity sequence and the data collected from the USC e-services projects. Section 5 presents some lifecycle characterizations. In Section 6, sequence patterns are explored while Section 7 considers the risks associated with particular sequence element. Lastly, Section 8 presents some conclusions and thoughts for future direction with this work.

2 Overview of Previous and Related Work

Traditional software development process such as the waterfall model requires most if not all requirements to be identified before proceeding to the design, code and testing phases. However, in CBA projects it is extremely risky to specify upfront, detailed requirements for the software system. For example: if the project development team specifies detailed requirements before investigating the COTS market, the project risks losing the option of using a good COTS package even though it may satisfy the main objectives of the requirements but not the specific detailed requirements. This may result in not finding any COTS products that will satisfy all requirements, even though there may be many that satisfy the critical or important objectives of the system.

To alleviate this problem researchers have proposed various mechanisms of developing CBA's. Some have proposed a generic process for CBA development [2] while some others propose to use a separate process, based on the type of CBA project, to develop and implement CBAs [1, 4], Process frameworks such as the spiral model [3] and the SEI Evolutionary process for integrating COTS based systems (EPIC) [2] provide a suitably flexible and concurrent framework for development and implementation CBA projects. These processes however provide guidelines at a much more abstract level than what is required for development and implementation of CBA. In that, these processes often fail to address the specific decision set for navigating through the option space in the development and implementation of CBA's.

We believe that within the CBA domain there is a large variation in development approaches (e.g. turnkey, adaptation, integration) for which a single generic CBA process is unable to provide adequate development guidance. In the past our attempts at an inclusive generic CBA process have met with limited success. In our own attempts in utilizing a single generic CBA process we noticed numerous projects succumbing to serious effort allocation pitfalls. To this end we introduced guidelines for early classification and continuous monitoring of a CBA project along with development guidance based on the type of CBA project [5, 6]. These classifications are useful for strategic and tactical development planning as they provide an overview of the potential risks, characteristics, and development effort priorities. An early identification and classification of the type of CBA project can help a project development team clarify their development process and manage risks, especially risks due to COTS effort allocation which are a major source of CBA project instabilities [7].

3 COTS Based Applications Process Decision Framework [6]

In observations of e-service projects over a seven year period, the authors have seen a pronounced increase in the fraction of CBA projects [5]. This resulted in increased conflicts within the UML-based [8] development process used within the project. This has led to a good deal of confusion, frustrating re-work, risky decisions, unsatisfactory products, and unhappy stakeholders. A notable example of this re-work occurred within one of the authors' "USC Collaborative Services" project in which the developers scrapped (after much expended effort) their process-mandated UML based design models and substituted extensive and detailed assessments and comparisons of several COTS packages, each of which covered most or all of the desired capabilities. In analyzing this problem, we found that the ways that "successful" projects handled their individual assessment, tailoring, and glue code activities exhibited considerable similarity at the process element level. We also found that these process elements fit into a recursive and reentrant decision framework accommodating concurrent CBA activities and frequent go-backs based on new and evolving Objectives Constraints and Priorities (OC&P) and COTS considerations.

Figure 1 presents the dominant decisions and activities within CBA development as abstracted from our observations and analysis of USC e-services projects and CSE-affiliate projects. This represents the overall CBA decision framework that composes the assessment, the tailoring, glue-code, and custom code development process active ties within the overall development cycle. The CBA process is undertaken by "walking" a path from "start" to "Non-CBA Activities" that connects (via arrows) activities as indicated by boxes and decisions that are indicated by ovals. Activities result in information that is passed on as input to either another activity or used to make a decision. Information follows the path that best describes the activity or decision output. Only one labeled path may be taken at any given time for any particular walk; however it is possible to perform multiple activities simultaneously (e.g. developing custom application code and glue code, multiple developers assessing or tailoring).

4 COTS Activity Sequences

In our six years of iteratively defining, developing, gathering project data for, we identified three primary sources of project effort due to CBA development considerations and further evidenced in [5], which shows the distribution of COTS activities within the COTS related effort. For the purpose of this paper, we define these activities as:

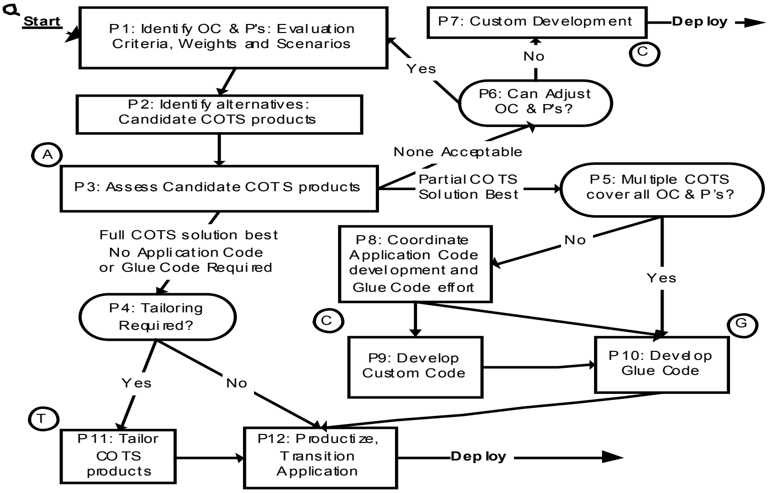

Start
P1: Identify OC & P's: Evaluation Criteria, Weights and Scenarios
P2: Identify alternatives: Candidate COTS products
A
P3: Assess Candidate COTS products
Full COTS solution best No Application Code or Glue Code Required
P4: Tailoring Required?
Yes
No
T
P11: Tailor COTS products
P12: Productize, Transition Application
Deploy
P7: Custom Development
Deploy
C
No
Yes
P6: Can Adjust OC & P's?
None Acceptable
Partial COTS Solution Best
P5: Multiple COTS cover all OC & P's?
No
Yes
P8: Coordinate Application Code development and Glue Code effort
C
G
P9: Develop Custom Code
P10: Develop Glue Code

Fig. 1. The CBA Decision Framework

- COTS Assessment is the activity whereby COTS products are evaluated and selected as viable components for a user application.
- COTS Tailoring is the activity whereby COTS software products are configured for use in a specific context. This definition is similar to the SEI definition of "tailoring" [10].
- COTS Glue Code development and integration is the activity whereby code is designed, developed, and used to ensure that COTS products satisfactorily interoperate in support of the user application.

The small circles with letters A, T, G, C in Figure 1 respectively indicate the areas of assessment, tailoring, glue code, and custom code development process activity within the CBA decision framework. Table 1 summarizes the results with respect to these activity areas from applying the process CBA decision framework within various e-services projects. We observed that not all projects follow similar activity sequences [6]. For example: Assessment and tailoring were the dominant activity for project 2 in the elaboration spiral, while for project 3 during the elaboration

Table 1. USC e-Service CBA Sequences

No	Effort Sequences			
	Incp.	Elab.	Cons.	Tran.
1	A	AC	ATG	C
2	A	AT	A	A
3	A	(TG)A	G	G
4	A	A(TG)	A(TG)	G
5	AT	AT	T	T
6	A	T	TG	G
7	AT	T	T	T
8	AT	(AA) TG	(TGC)	G
9	A	AT	TG	G

phase the team implemented in parallel tailoring and glue-code activities, and on finding the COTS unsuitable for their need this team implemented the assessment activity in the next spiral. Such differences in the activity sequences resulted in a specific set of charac-teristics and risks reported by the projects. Based on these reports, we have abstracted some common characteristics, patterns, and associated risks, which we shall present in the following sections.

4.1 Representation

To reconstitute the activity sequences, we used an effort reporting system that has been utilized by all the projects such as those listed in Table 1 (along with many others not listed there). The representation of a COTS activity effort sequence is the particular time-ordered sequence of letters A, T, G, and/or C as reported by the system. Parentheses are used when two or more activities were conducted in parallel. For example, (TG) would represent simultaneous tailoring and glue code. For the purpose of this paper, all tailoring and glue code effort expended exclusively for assessing COTS packages are considered assessment activities.

Table 1 shows a sampling of the A, T, G, C sequences for some of the CBA projects in terms of Inception, Elaboration, Construction, and Transition phases [8]. The sequences of activities are time ordered from left to right.

4.2 Data Collection and Validation

Within the USC e-services projects we collected data from three major data sources. These sources include weekly effort report, the weekly progress report and the weekly risk report. The weekly effort is a form of quantitative data where the team is required to report both COTS and non-COTS effort its members spent during the course of a week. The weekly progress report highlights the tasks that the project team completed in the prior week, and the tasks that it plans to complete in the next week. The weekly risk report consists of the top ten risks that the team is facing in the project. Both the weekly progress and risk reports are qualitative. Having both quantitative and qualitative reports helps determine the correctness and consistency of the team reports, and for our present purposes, discern notable patterns (or lack of expected patterns).

This data however has certain limitations in lieu of our research. These limitations have been listed below:

- It is not possible to differentiate atomic A, T, G, or group A, T, G inside a particular development phase. For example, T is presumed to be performed on a single COTS package, but if the project needed to tailor a set of COTS products, there is no way to correctly reconstruct and interpret the corresponding sequence of tailoring activities between the products and subsequent assessment. The representation is simply the aggregate effort for all the products.

- All projects were under a very tight 24-week schedule. These schedule constraints might significantly affect COTS considerations and decision making in the middle or even near the end of the project. For example, re-assessment caused by late requirements change or COTS volatility might eat up available schedule so much that the project cannot be completed in time. Thus, re-assessment activity becomes more costly than in non-schedule constrained projects.

5 Phase Characteristics of COTS Activity Sequences

The CBA activity sequences were defined and exemplified briefly in Section 4. In this section, we elaborate on the particular decisions made within the CBA decision framework that the activity sequence letters represent. We aim to illustrate and compare the overall similarities and differences between CBA projects. For example, continued assessment activity in Elaboration is common (projects 1,2,3,4,5,8,9), but some of this can be attributed to re-assessment resulting from insufficient earlier assessment (projects 1,2,3,5). This sometimes led to further, often more costly and risky assessment activity within Construction when forewarnings of significant COTS changes went undetected (Project No. 1, 2, 4). Such cascading events are typically not directly addressed by the current literature on CBS processes. This section concludes with a summary and overview of typical specific kinds of COTS activities within each development phase as observed within the USC e-services CBA projects analyzed.

5.1 Case Studies of the USC E-Services CBA Projects

The following table provides an elaboration of the CBA project sequences in the USC e-services CBA projects previously listed in Table 1. The dialog indicated is the result of actual, specific in process application of the CBA decision framework in Figure 1 [6].

5.2 Phase COTS Activity Summary and Overview

Inception. All projects spent certain amount of effort on assessment (A) in the Inception phase, i.e. the first spiral cycle. For projects 6 and 7 the initial assessment was a “what remains to be done” with respect to a pre-selected COTS products/packages mandated by the clients. All other projects collected and assessed a variety of COTS candidates with respect to the project’s high-level system objectives, constraints and priorities. Some projects needed to do some tailoring of the COTS candidates to help initial assessment.

Table 2.

Project No	Inception	Elaboration	Construction	Transition
1.	Initial Assessment (A)	Further assessment (A) proved non-COTS solution (C) better. However, at the end of Elaboration phase, the Client introduced a new COTS candidate	Another round of assessment (A) on the new COTS package; Tailoring (T); Glue code development (G)	Glue code development found out COTS did not completely satisfy the requirements; redirect to non-COTS solution (C)
2.	Initial Assessment (A)	Detailed assessment (A); prototyping by customizing the likely winning COTS package (T). However, the client was interested in further assessment involving more COTSs rather than starting building an operational system. Project was redirected to assessment intensive.	Learning and studying new COTS candidates; setting up testing environment –COTS installation and configuration (A)	COTS evaluation testing; final COTS package recommendation. (A)
3.	COTS selection (A)	Tailoring (T), glue code development (G) found out COTS not working as expected; reevaluate project feasibility (A)	Integrating selected COTS (G).	Transitioning the operational system.
4.	Initial filtering (A)	Detailed assessment (A); Prototyping by tailoring (T) and glue code development (G)	Project re-scoped: assessing COTSs (A) for two main features needed; installation, tailoring (T), glue code development (G)	Transitioning the operational system.
5.	Evaluate one COTS product 1 (A); Prototyping with selected COTS 1 (T);	De-compiling COTS 1 to explore other features; evaluate COTS 2 for other features (A); configuring COTS 1 (T)	Configuring COTS 1 and 2 (TT)	Transitioning the operational system.
6.	Study mandatory COTS (A)	Prototyping by tailoring COTS (T)	Continuing tailoring (T) and glue code development (G)	Transitioning the operational system.
7.	Study the mandatory COTS product (A); Prototyping by tailoring COTS (T)	Prototyping by tailoring COTS (T); reprioritizing requirements with respect to COTS performance and limitations	Configuring COTS (T)	Transitioning the operational system.
8.	Assessment by tailoring COTS set 1 (AT)	Re-evaluation according to new requirements (A); assessing COTS set 2 to handle other features (A), tailoring (T); glue code development (G).	Continuing COTS tailoring (T), glue code development (G), and custom development (C)	Transitioning the operational system.
9.	Initial assessment (A)	Detail evaluation (A); installation and tailoring (T)	Continuing tailoring (T)	Glue code development (G); Transitioning the operational system.

Elaboration. In the Elaboration phase, COTS assessment (A) tends to be done more thoroughly along with some COTS tailoring (T) and glue code development (G) in order to gain first-hand experience with the candidate COTS, collect data for analyzing COTS, or test the operational performance of COTS for the particular context of the project. Meanwhile, possible frequent requirement changes and different COTS characteristics result in a wide variation of effort sequences.

Construction. Except for two incomplete projects 5 and 7, the Construction phase typically consists of T and/or G, whereby customizing the selected COTS packages and integrating them into the target system is performed. Project 2 is a notable exception as described below.

Transition. While most of the CBA projects delivered an operational system at the end of transition phase, some did not and never were expected to. For instance, project 2 delivered a set of COTS evaluation and recommendation reports on which COTS should be used and why. This outcome is not considered a failed project so long as the project goal from the outset was to deliver a feasibility study. Furthermore, if a CBA project concludes quickly with “this project is unfeasible,” then this too is a success of sorts so long as no unreasonable expenditures (e.g. cost, effort, schedule) were made to come to this conclusion.

6 Defining Sequence Patterns

Analyses of the case study CBA project sequence data show that there are certain patterns that exist in both the successful and the not so successful CBA projects. With respect to the CBA decision framework, we detail notable patterns that were expected (and indeed appeared) and some that were a surprise. Such patterns are interesting in their own right, however they have primarily been used to validate and refine the CBA decision framework when first described in [6].

6.1 Anticipated Patterns

An anticipated CBA sequence pattern is a sequence of COTS activities that:

a. Is expected or perceived as common within a typical project
b. Has a viable rationale to exist within a typical project
c. Is a valid “walk” within the CBA decision framework indicated in figure 1 (more specifically, is a valid set of composable elements from the framework as detailed in [6])
d. Does not violate critical constraints of a typical project

Given the above, we described anticipated patterns by listing the percentage of case-study projects the pattern exits in and the rationale as to why the pattern is consistent within a project.

Pattern 1: Assessment first. After identifying system objectives constraints and priorities and collecting an initial set of COTS candidates, COTS assessment is usually performed. This commonly done to evaluate the extent that the candidate COTS capabilities can satisfy requirements.

Observed in case-study projects: 100%

Note that this pattern is clearly affirmed within CBA decision framework, in which the assessment process element is suggested at the very beginning of the CBA process decision framework.

Pattern 2: Assessment to tailoring (A ⇒ T). It is a seemingly natural relationship between assessment and tailoring. While assessment is on going or once assessment is done, it becomes more clear what needs to be done and what can be customized or parameterized in a COTS product before being utilized

Observed in case-study projects: 100%

Pattern 3: Tailoring to glue code (T ⇒ G). When integrating COTS packages, often tailoring can help prepare unrelated COTS packages to "fit" together with glue-code. For example setting the output of the COTS packages to a standard data interchange format such as CSV or XML.

Observed in case-study projects: 33% (100% when subsets of pattern 4 are counted)

Pattern 4: Assessment to tailoring and glue code (A ⇒ (TG) or A ⇒ T ⇒G). As A ⇒ T and T ⇒ G patterns are common, this pattern is generally just the aggregate of these two patterns (in the two ways they can aggregate) when both patterns are already present. This is particularly true with multiple COTS candidates where a thorough assessment with tailoring and/or glue code development to do experiment testing on COTS candidates to avoid faulty candidate being selected.
Observed in case-study projects: 67%

Pattern 5: After Inception, A, T, TG as a repeatable pair (A ⇒ A or T ⇒ T or (TG) ⇒ (TG)). Due to frequent requirement changes, COTS volatility, re-assessing, or re-tailoring a COTS package is common in addition to possibly a certain amount of rework on glue code to accommodate the changes. In addition, some projects aim to utilize COTS without any tailoring or perhaps with at most, some tailoring but no glue code (for example in a single COTS package solution).

Observed in case-study projects: 67%

Unanticipated Patterns

An unanticipated pattern is a pattern that is not expected typically occur (and as a result, not often planned for) in a project, yet is a valid within the CBA framework, has project rationale, and is observed within the case-study projects.

UPattern 1: Assessment to Custom Development (A ⇒ C). It is a little disappointing to find that no COTS candidates will feasibly satisfy pre-defined system OC&P's. These OC&P's cannot be changed or negotiated to accommodate the constraints imposed by the available COTS packages. Therefore, a custom development becomes the only viable option. This pattern is unexpected as people do not generally look for COTS solutions unless there is some initial indication that it is feasible to pursue (i.e. flexible OC&P's, known COTS candidates, etc.)

Observed in case-study projects: 11%

Tailoring to assessment (T ⇒ A or (TG) ⇒ A). Most likely, such a sequence segment results from requirement changes, COTS changes, or insufficient early assessment. Another reason is that during integration, COTS software is packaged and delivered in many different forms (e.g. function libraries, legacy application, commercial application, tools, system services, and frameworks. The integration mechanism largely depends on the way a COTS is packaged and delivered and must be re-assessed with respect to the target system (e.g. different windows platforms). A third reason might be interoperability difficulty: many COTS packages are implemented using different component technologies and there is a lack of interoperability standards to facilitate the integration. In all above cases, further assessment is typically required.

Observed in case-study projects: 56%

Unlikely Patterns

Unlikely patterns are sequences that violate the CBA framework, are contradictory, or lack any project rationale to exist.

Tailoring or glue code first in Inception. No observation was found in this case even though several projects started with mandatory, pre-determined COTS packages.

Custom development first in Inception. The goal of COTS is to avoid custom development effort. Even if some custom development is required, generally the development would orient around the COTS capabilities first, then focus on the custom to complete the project. This is supported by the data in the case studies where the two projects that had custom development performed the majority of this activity late in the development.

Assessment, Tailoring, and Glue code in parallel ((ATG)). How can glue-code be written without some prior assessment or tailoring? What would determine what needs to be done? There may be exceptional cases such as when integrating a large number of COTS in which parallel sub-projects may be formed to manage the complexity and efficiency of the development.

Glue code to Tailoring ($G \Rightarrow T$) . The authors at this time do not have an explication as to why this does not seem to appear in practice.

7 Risk Arising from COTS Activity Sequences

The CBA decision framework emphasizes risk-reduction by dynamically composing a set of success-critical COTS related activities into a certain sequence as timely response to the dramatically changing COTS world. In this section, we will highlight the common COTS risks that arise within COTS activity sequences.

The table below lists certain sequence patterns, discusses what kinds of risk items might be present, and what kind of risk mitigations might be planned to reduce these risks.

8 Conclusions and Future Work

COTS activity sequences provide an effective means analyzing complex and often subtle COTS process elements. In particular, they have proven invaluable for validating and refining the CBA decision framework that has already proven to be of substantial value to developers inexperienced in COTS based system development. CBA sequence patterns provide an empirical means of identifying and possibly avoiding COTS risks. CBA sequence can aid in strategic planning to meet cost, schedule, and quality goals.

In future work we hope to compile a repository of CBA sequence patterns and the various contexts in which they apply and do not apply. We also plan to expand the expressiveness of the notation for describing CBA sequences, perhaps to include duration data, regular expressions for repeated sequences and constraints, and more detailed activity categories or sub-categories. Furthermore, we would like to correlate the presence of patterns and upatterns with project success metrics. Lastly, investigate patterns that arise from risks. At present, we have examples of how risks arise from certain sequences.

Acknowledgements. The authors would very much like to thank Jesal Bhuta for contributing insight into COTS tailoring processes and COTS risks within the case studies. This work is a derivative of the original concept for a composable COTS process initiated by Barry Boehm and the many stimulating subsequent discussions and publications that ensued. Without these, this work would never have come to light.

Table 3.

Segment of Activity sequence	Indicated Risk Items	Risk Mitigations	# Of occurrences in case-studies

A(T)
Or
A(TG)
	Selecting faulty COTS candidate;
	Insufficient early assessment will cause problem while integrating
	Faulty COTS vendor claims
		Performing very detailed assessment:
		➢ Hands on experiment;
		➢ Prototyping by tailoring
		➢ Prototyping by developing some glue code
		Investigating more information other than obtained from the vendors or marketplace.
			9

TA	Requirement changes make initial COTS unsatisfactory; COTS version upgrade during development demands re-evaluation; Tailored package didn't perform as expected. Insufficient early assessment	Introduce a new assessment activity to evaluate the change impacts. Introduce a new assessment activity with newly obtained information and/or newly learned knowledge.	3
GA	Requirement changes make initial COTS no longer satisfying; COTS version upgrade during development demands re-evaluation; Insufficient early assessment Integrated system did not perform as expected. Inexperienced COTS integrator Lack of interoperability standards to facilitate the integration	Introduce a new assessment activity to evaluate the change impacts. Introduce a new assessment activity with newly obtained information and/or newly learned knowledge. Introduce a new assessment activity to find best integration option	2
(GC)	COTS package incompatibilities may result in feature loss and significant project delays Requirements mismatch	Determine the interconnection topology options and minimize the complexity of interactions; Implement the appropriate interfaces simultaneously with application code development.	1
AC, or TC, or GC	Critical requirements mismatch No suitable COTS satisfied	Carry out a pure custom development from scratch.	1

References

[1] C. Abts, B. Boehm, and E. Bailey Clark, "COCOTS: A Software COTS-Based System (CBS) Cost Model," *Proceedings, ESCOM 2001*, April 2001, pp. 1-8.

[2] C. Albert and L. Brownsword, "Evolutionary Process for Integrating COTS-Based Systems (EPIC): An Overview," CMU-SEI-2002-TR-009, July 2002.

[3] B. Boehm, "A Spiral Model of Software Development and Enhancement," Computer, May 1988, pp. 61-72.

[4] M. Morisio, C. Seaman, A. Parra, V. Basili, S. Kraft, and S. Condon, "Investigating and Improving a COTS-Based Software Development Process," *Proceedings, ICSE 22*, June 2000, pp. 32-41.

[5] Dan Port, Ye Yang, Jesal Buhta, and Barry Boehm, "Not All CBS Are Created Equally: COTS-Intensive Project Types", 2nd International Conf. on COTS-Based Software Systems (ICCBSS'03), Ottawa, Canada, Feb. 2003.

[6] Barry Boehm, Dan Port, Ye Yang, and Jesal Buhta, "Composable Process Elements for Developing COTS-Based Applications", accepted by 2003 ACM-IEEE International Symposium on Empirical Software Engineering (ISESE 2003).

[7] V. Basili, B. Boehm, "COTS-Based Systems Top 10 List", IEEE Computer, Vol. 34, No. 5, May 2001

[8] W.E Royce, Software Project Management: A Unified Framework, Addison-Wesley, 1998.

[9] D. Carney, P. A. Oberndorf, and P.R.H. Place, "A Basis for an Assembly Process fo COTS-Based Systems (APCS)," Technical Report, CMU/SEI-2003-TR-010, May 2003.

Assessing COTS Assessment: How Much Is Enough?

Dan Port[1] and Scott Chen[2]

[1]Department of Information Technology Management
University of Hawaii at Manoa, Honolulu, HI
dport@hawaii.edu
[2]Center for Software Engineering
University of Southern California, Los Angeles, CA
chen@cse.usc.edu

Abstract. COTS products are now ubiquitous and clearly have become a key factor in modern software systems development. If COTS are chosen poorly, a project will likely fail. As a result, the careful assessment of COTS products has become an essential element of the development process. There are numerous approaches to COTS assessment; however none of them address the crucial question of how much assessment effort to perform. If too little assessment is done, inappropriate COTS may be used; if too much assessment is done, the effort expended may place the project at risk. It is important to achieve a satisfactory balance between COTS uncertainty risks and risks resulting from project delay. To address this, we develop a method for the strategic planning of COTS assessment by determining "how much is enough" effort (in time, cost, or quality) with respect to critical project risk factors such as project schedule, market window, and a multitude of COTS assessment attributes such as availability, ease of use, maturity, and vendor support. The method is practical, and provides valuable aid in the planning of COTS based system development.

Keywords: COTS, COTS assessment, COTS integration, software risk, COTS evaluation

1 Introduction

COTS (Commercial-Off-The-Shelf) have become a key issue in software system development. However, introducing COTS into the development process greatly changes project risk factors [1, 13]. In particular, COTS introduces a significant amount of risk arising from uncertainty within concerns such as faulty vendor claims, support of standards, interface with legacy systems, hidden defects, and a host of others (see 13] for a more elaborate list). If the wrong COTS are used, the risk a project will fail increases dramatically [1]. Consequently, the thorough assessment of COTS has become an essential practice in the development of COTS based systems (CBS). What is COTS assessment? Simply said, it is finding the right COTS for your project. More precisely, using a series of methods to evaluate COTS products with respect to the requirements of your project and determine which ones (if any) are suitable for your project. There are a large variety of COTS assessment approaches. A brief list of popular methods include PORE - Procurement-Oriented Requirements

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 183–198, 2004.

Engineering method [2], OTSO - Off-The-Shelf Option [5], Checklist Driven Software Evaluation Methodology (CDSEM) [10], COTS-based Integrated System Development (CISD) Method [9], CAP–COTS Acquisition Process method [11], CRE - COTS-Based Requirements Engineering method [8], CEP - Comparative Evaluation Process Activities [12], CBA Process Decision Framework [13], A Proactive Evaluation Technique [14], STACE - Socio-Technical Approach to COTS Evaluation [15], Storyboard Process [16], Combined selection of COTS components method [17], PECA process [18], and COTS-DSS [19].

As indicated above, there are many approaches to COTS assessment. Each has their own unique set of strengths and shortcomings with respect to the COTS attributes that are to be assessed. Ideally one would select the most appropriate approach for each particular attribute to be assessed. However, this is a non-trivial endeavor; furthermore, none of the assessment approaches address the question of how much assessment should be performed. This introduces a degree of risk into the project which now will be elaborated.

Any development activity involves risk [22]. Risks are situations or possible events that can cause a project to fail to meet its goals. They range in impact from trivial to fatal and in likelihood from certain to improbable. In the case of COTS assessment, risk often results from uncertainty. With too little COTS assessment, there is risk that the appropriate COTS may not be selected; with too much assessment, other development tasks may be robbed of critically needed effort, putting the project completion schedule at risk. Indeed, a particularly common COTS assessment pitfall is “analysis paralysis” whereby assessment never converges to a satisfactory end point. Trade-offs are considered, re-considered, and the quest for the “perfect” COTS drags on and on. A stopping point needs to be established based on fundamental project factors outside of COTS considerations such as buy versus build considerations and project completion goals. This work develops a new approach for making strategic design decisions by determining how much effort (or time) should be spent assessing COTS products with respect to project risk factors such as cost, market window, COTS assessment attributes such as availability, ease of use, maturity, and vendor support. It provides a valuable assessment aid when making strategic software design decisions. We lay the foundation for a practical, empirically based approach for the strategic planning of COTS assessment effort. Here we describe a means for assessing and comparing the cost/benefit of assessing COTS attributes (including composing multiple assessment approaches) and counterbalancing this with critical project risk factors such as buy versus build options, minimum COTS implementation requirements, and project completion goals.

2 Risk Considerations in COTS Assessment

In preparation for an analysis of COTS assessment, we introduce the technique of *risk profiling*. To begin this discussion we define *risk exposure* (RE) [22], which is computed as the product of the probability of loss P(L) and size that loss S(L) summed over all sources of loss for a particular risk item. Since risk considerations dictate the path development takes, it is important that risks be investigated candidly and completely. See the references for a taxonomy of risks [20] and a method for identifying them [21, 22]. A *risk profile* (or *RE profile*) is the evaluation of RE as a function of monotonically increasing quantity such as time, cumulative effort, or cumulative cost. Expressing development considerations in terms of risk profiles enables quantitative assessment of generally qualitative attributes. A useful property of RE is that if it is computed entirely within a particular project (i.e. no external loss sources), we may assume RE is additive. This will be true regardless of complex dependencies analogous to 'expectation' in classic probability. We will take advantage of the additively of RE in order to optimize risk profiles for COTS assessment. Such analyses provide strategic trade off considerations and have proven useful in numerous other applications such as defect removal, development process agility, and architecture flexibility determination.

What kinds of risks are involved within COTS assessment? Generally there is risk of project error (REerror) from doing too little effort and project delay (REdelay) from doing too much. We assume that a project starts out with a given amount of risk due to uncertainty and potential problems, and that it is desirable to expend effort to reduce overall RE. Lower RE implies less loss is expected to be incurred on a project. The relationship between REerror and COTS assessment effort is simply that assessment reduces uncertainty and potential errors that may arise from issues such as mismatch of COTS capabilities and system requirements, COTS defects, faulty vendor claims, incompatible COTS architecture, and so on. Problems in these areas tend to result in significant project rework and other project costs. Hence it is vital to employ a risk reduction strategy. However, it is not obvious that any given strategy will be effective (or even feasible). This issue will now be elaborated.

2.1 REerror: COTS Assessment Error Risk

The risk profile corresponding to the cumulative potential loss from errors in utilizing COTS (e.g. COTS defects, vendor problems, integration issues, etc.) will be called REerror. The more assessment that is done, the lower REerror is resulting from to unforeseen or uncontrolled COTS attributes (such as those listed in Table 2). Assessed COTS attributes reduce both the size of loss due to "COTS surprises" and the probability that surprises still remain. Prior to embarking on COTS assessment, the project will likely contain many potential COTS errors, either known or from COTS uncertainties [1]. This results in an initially high (relative to the project) P(L) value. In this, some may be critical, and so S(L) will be (again relative to the project) high. Thus REerror initially will be high. When COTS assessment has been employed, the likelihood

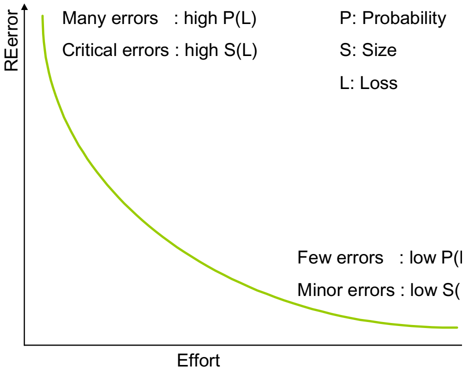


Fig. 1. REerror Profile

of errors will be reduced. If assessment is done throughly, most of errors remaining are likely to minor, resulting in low REerror. So at the beginning, REerror decreases fast.

It is generally not feasible to be totally exhaustive when performing COTS assessment. As a result, the ideal COTS assessment risk reduction profile (as illustrated in Figure 1) is where REerror decreases rapidly at the beginning slows as greater assessment effort is expended. This profile is ideal because it provides the maximum risk reduction for any given amount COTS assessment.

To give a concrete example of REerror, consider the data in Table 1a taken from the USC Collaborative Services System project (please see the case study section for further details on this project). The data in column A indicates the COTS attribute to be assessed as listed in Table 3 and taken from the more comprehensive list of attributes in Table 2. Here S(L) is the size the potential loss in terms of percentage of the project value that would result from an error in the COTS attribute, while P(L) is the corresponding probability (as a percentage) of such a loss occuring. These in turn are used to calculate the corresponding RE reduction if the attribute is fully assessed. The RE has been normalized to the fractional portion of the total RE that may be reduced through assessment. Column E indicates the effort in terms of effort (in hours) to perform the assessment of the attribute, and CB is the cost-benefit ratio calculated as RE/E which will be elaborated on later.

Table 1a. Risk Exposure Data for USCCS Project

A	S(L)	P(L)	RE	E	CB
1	4.64	26	0.0452	13.92	0.0033
2	7.73	35	0.1015	23.19	0.0044
3	25.77	32	0.3093	77.31	0.0040
4	3.09	10	0.0116	9.27	0.0013
5	4.12	30	0.0464	12.36	0.0038
6	2.58	28	0.0271	7.74	0.0035
7	4.12	15	0.0232	12.36	0.0019
8	7.73	33	0.0957	23.19	0.0041
9	7.73	25	0.0725	23.19	0.0031
10	4.12	12	0.0185	12.36	0.0015
11	3.09	18	0.0209	9.27	0.0023
12	3.09	20	0.0232	9.27	0.0025
13	5.15	29	0.0560	15.45	0.0036
14	3.09	8	0.0093	9.27	0.0010
15	5.15	26	0.0502	15.45	0.0033
16	4.12	23	0.0355	12.36	0.0029
17	4.64	31	0.0540	13.92	0.0039

Some care must be taken in choosing a strategy will result in the ideal profile indicated in Figure 1. Figure 2 compares three common strategies for the order in which the assessments from Table 1a might be performed. The tick marks on each RE profile correspond to the assessment of a particular attribute. If the attributes are assessed in an arbitrarily order, the curve will typically look like the approximately linear curve indicated in the middle of Figure 2 which clearly does not achieve the ideal of Figure 2. Another strategy is simply to do the least effort assessments first. This generally results in the even less desirable 'supra-linear' REerror reduction profile indicated in the top-most curve in Figure 2. Performing the assessments in the order of highest cost-benefit will archive the desired REerror reduction profile.

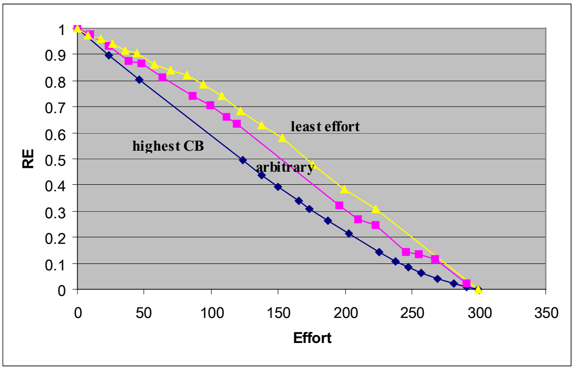


Fig. 2. REerror Profiles for USCCSS Project

For our particular USCCS project example the differences are not very pronounced because the differences in RE (and to a lesser extend the effort) between the attributes are relatively small. However in larger, more complex projects, the differences between the strategies can be profound. This is our first indication that it is prudent to invest a little in assessing your COTS assessment strategy before enacting it. If no assessment of the strategy is considered, then you are likely to end up with a less than ideal REerror reduction profile. The consequence of this are felt if all assessment tasks are not performed. This occurs frequently as assessment stops when it "feels" like enough risk has been reduced or higher priority is given to other tasks. This results in a project leaving a high-degree of risk amounting to the usual consequences [1, 9]. On the other hand, if exhaustive assessment is performed, then the overall effort expended may introduce new project risk. This later risk consideration is of a different nature than REerror and be discussed next.

2.2 REdelay: COTS Assessment Delay Risk

Ideally one would do as much as COTS assessment as possible to lower the risk of hidden COTS problems. What happens if too much assessment is done? Excessive assessment not only increases effort, but also will delay the project. This means that assessment activity itself introduces new risk into the project. Such risks can negatively impact or even paralyze a project to the point that the project will not complete on time or complete at all. Project delay may result in losses due to non-use of the system when required or expected, dissatisfied customers, or the combination of competitors entering the market and decreased profitability on the reduced market share. The risk profile associated with the potential failure to complete the project on time will be called REdelay. Here the relationship between REdelay and COTS assessment effort is elaborated. REdelay will monotonically increase since REdelay represents the cumulative RE due to delay. We assume that the project starts out with no risk of delay and that any effort expended contributes to the overall delay risk. Due to compounding factors, it has been empirically suggested within the COCOTS [26] assessment sub-model that REdelay will increase supra-linearly. Owing to this, a reasonably good approximation to the REdelay profile can be generated through the identification of a few well chosen data points. For COTS assessment we have found the following three 'critical points' to be useful:

Point 1: Buy versus Build Option
This point indicates the boundary that, if passed, the project will no longer have the resources required to custom develop the entire system. This may be due to over commitment of development resources, staff skill deficiencies, training overhead, or simply a lack of remaining time due to over-investment in COTS assessment activities. REdelay will increase as COTS risks that were previously mitigated by the option to custom build the required capabilities is no longer viable. For example, not having the option to build escalates the criticality for identifying or acquiring suitable COTS.

Point 2: Implementation of COTS
This point marks the boundary where there will not be sufficient resources to implement COTS within the project. If COTS are to be used, then some significant amount of resources must be dedicated to acquiring, installing, training, tailoring, and integrate the packages into the project [1, 4, 26]. Passing this point dramatically increases REdelay

Point 3: Project Completion
Every project, no matter how flexible, has constrained resources, be they budget, schedule, or other economic considerations. There always exist a point where project resources will be exhausted and the project must complete prior to this point. Passing this point will put the project at extreme risk. Less clear is that over-investment in some activities, in particular COTS assessment activities, may be incompatible with project completion constraints. The extreme case of this is 'analysis paralysis' where COTS assessment never converges and eventually the effort reaches a level to which the project cannot be finished prior to the desired project completion point.

As the schedule for points 1, 2, 3 are necessarily successive, and the effort to arrive at these points is cumulative, REdelay at these points successively increase respectively. Moreover, the increase in REdelay between points 1 and 2 is strictly greater than that between the starting point and point 1. Similarly for points 2 and 3, except the increase is more pronounced. The critical points for the USCCS Project are contained in Table1b and the resulting REdelay profile is illustrated in Figure 3.

Table 1b. USCCS Critical Points

Point	S(L)	P(L)	RE	C
Buy/build	30	15	0.045	100
COTS Implement	65	40	0.26	275
Proj Comp	80	70	0.56	400

As is evident by USCCS Project example, calculating risk exposures requires an organization to accumulate a fair amount of calibrated experience on the probabilities and size of losses as functions of assessment effort (e.g. cost, duration) and project delay. There are many practical approaches that address this topic specifically (see for example [9, 11, 21, 22]) and while important, will not be considered henceforth.

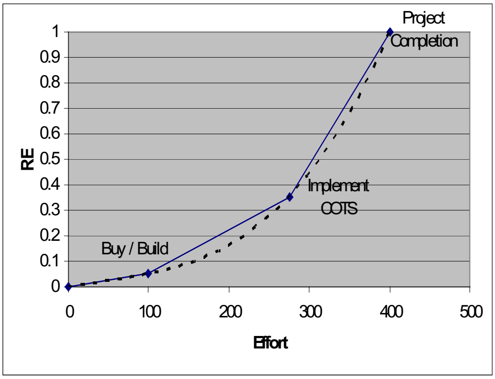


Fig. 3. USCCS Critical Points

3 Balancing Risks: How Much Is Enough COTS Assessment?

At this point we have seen that it is important to have an efficient strategy for reducing REerror and avoid coming in conflict with REdelay critical points. We have indicated that COTS assessment reduces REerror while simultaneously increasing REdelay. A desirable overall COTS assessment strategy decreases REerror but not expend so much effort that this reduction is dominated by REdelay. This is the more formal solution to the informal question "how much is enough COTS assessment?" This approach follows the general principle for avoiding over-evaluation and under-evaluation of COTS assessment attributes based on risk-considerations:

- If it's risky to not evaluate extensively, DO evaluate extensively (e.g., scalability, safety);
- If it's risky to evaluate extensively, DO NOT evaluate extensively (e.g., well-established, well-known, highly tested products).

Our goal is to apply the above principle to balance REerror and REdelay to determine a strategic amount of COTS assessment to perform before committing to a particular system design. The optimal effort to expend will be that which minimizes the sum REerror + REdelay. Doubtless there are many dependencies among the risk factors, however recall that despite this, RE's will be additive. As shown in Figure 4a, the decreasing REerror and increasing REdelay must achieve a minimum – the "sweet spot" at some intermediate effort point. Assuming the ideal REerror reduction strategy discussed earlier, a strategic stopping point for COTS assessment is when this intermediate effort point has been reached.

Note that the location of the sweet spot will vary by type of organization. For example, in a "dot.com" company where REdelay increases rapidly due to market pressures, the result sweet spot is pushed to the left indicating that less COTS assessment should be done. By contrast, a safety-critical product such as for a nuclear power plant will have greater REerror due to larger potential losses. The sweet spot is pushed to the right, indicating that more COTS assessment should be done. The sweet spot determination for the USCCS Project is shown in Figure 4b. A 3rd order polynomial (shown in the figure) was used to interpolate between the critical points so that the REerror + REdelay could be estimated.

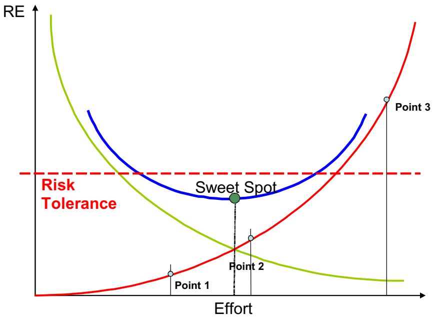


Fig. 4a. Balancing REerror and REdelay

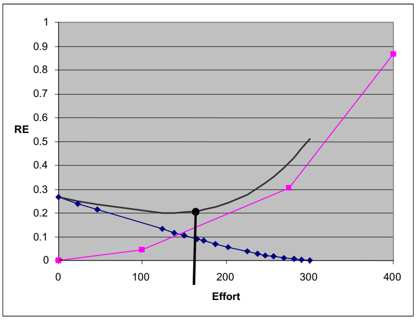


Fig. 4b. Sweet Spot for USCCS Project

There are situations where the sweet spot is not an acceptable determination of how much assessment to perform. Acceptability is achieved only when (as indicated in Figure 4a) the RE at sweet spot is below a given risk tolerance and the effort at the sweet spot is less than the effort at critical point 2. While ideally the assessment effort should be less than the effort at point 1, for most projects the additional risk incurred by passing this point is tolerable [1]. This risk is softened by the fact that the effort at the sweet spot cannot be far from the effort at point 1. The effort beyond point 1 to complete the COTS assessment is small enough that the buy vs. build decision will mostly be settled prior to passing point 1. The exception is if REdelay increases very gradually, but then in turn the increase in risk would be much less pronounced.

What can be done in the event that the sweet spot is above an acceptable risk tolerance? There are two solutions: find another assessment approach that can lower REerror, or mitigate potential losses (e.g. insurance policy) due to project delay in order to lower REdelay. It is best if both solutions are applied. If the sweet spot effort exceeds the effort for point 2, the only reasonable approach is to find another assessment approach that can lower REerror faster. Paradoxically, if REdelay were increased, say by imposing greater project cost, schedule, or effort constraints, this too could potentially move the sweet spot effort in front of point 2. However the price to pay for this is a heavy increase in the overall project risk which likely will exceed a tolerable level.

Table 2. Sample COTS Assessment Attributes

Correctness	Ease of Use	Product Performance
Accuracy	Usability / Human Factors	Execution Performance
Correctness	Complexity Ratio	Information / Data Capacity
	Effort for operating	Precision
Availability / Robustness	Administrability	Memory Performance
Availability	Time to use	Response Time
Fail Safe	Time to configure	Throughput
Fail Soft	Time to admin	Memory utilization
Fault Tolerance	Time to expertise	Disk utilization
Input Error Tolerance		
Redundancy	Version Compatibility	Understandability
Reliability	Downward Compatibility	User Documentation
Robustness	Upward Compatibility	Computer Documentation
Safety	Downward Compatibility	Documentation Quality
Failure removal		Simplicity
Error Handling	Intercomponent Compatibility	Testability
Persistent	With Other Components	Help System
Security	Interoperability	
Access Related	Data Compatibility	Functionality
Sabotage Related		Functionality Coverage
Data Encryption	Flexibility	Service Implementation Coverage
Controllability	Extendibility	Standardization
Auditability	Flexibility	Certification
		Tailorability

4 Assessing COTS Attributes and Assessment Techniques

When assessing COTS products there are a large number of attributes that affect RError such as Correctness, Version Compatibility, and Vendor Support. What is an attribute? Here we use this definition [23]: An attribute is a quality property to which a metric can be assigned, where a metric is a procedure for examining a component to produce a single datum, either a symbol (e.g. Excellent, Yes, No) or a number. What we want to know is the degree to which each of these attributes should be evaluated so as to provide an effective assessment strategy given a projects particular requirements and constraints. We would like to answer questions such as "it is better to buy or build?" and "should we choose a more reliable vendor or a product with more features?" Only by assessment of COTS attributes can we answer such questions.

Table 2 is a sample of COTS assessment attributes from [23, 24, 25]. They can be utilized to formulate a COTS assessment strategy by selecting attributes that are relevant to your project, assessing their relative risks, and considering cost-benefit ratios. However, the degree to which each assessment attribute is significant is relative to each COTS product, the project itself, and to the particular assessment technique applied. For example, Ease of use tends to vary greatly from product to product. Security may not be relevant for some uses of a particular component within the system regardless of which COTS products are used. Prototyping tends to provide more comprehensive results than product references (e.g. product testimonials). While Prototyping (as am assessment technique)may reduce the risk of error more than testimonials, it also takes more time and thus increases risk of delay.

Also relative to the particular product and project is how attributes vary with respect to the amount of assessment effort. For some COTS products, a single assessment is enough to resolve the uncertainty risk for the attribute (as is typical with Price). For others, it may be a function of time outside project control such as with Upgrade Ease and Vendor Support. There may also be attributes whose risk attributes are resolved "stepwise" through repeated evaluation, as typical with Availability / Robustness, Performance, and Ease of use. A COTS risk exposure assessment strategy should address the above issues. We now present a stepwise approach to formulating an assessment strategy that accomplishes this. In particular, it provides a practical to implement means of selecting an ideal REerror reduction profile when there are multiple assessment techniques for each attribute.

The multi-attribute, multi-assessment technique REerror reduction strategy algorithm:

1. Identify the most significant COTS assessment attributes from Table 2. Label them *1,…, n.*
2. Identify the most significant COTS assessment techniques (e.g. product testimonials, prototyping, etc.) applicable to the project, available resources (e.g. staff skills). Label them *1,…, m.*
3. Estimate the relative $S_i(L)$ quantities for attributes $i=1,\ldots,n$
4. Estimate the effort E_{ij} and the change in probability $\Delta P_{ij}(L)$ resulting from assessing assessment attribute *i* using assessment technique *j.* Henceforth we associate *ij* with (attribute *i*, technique *j*)
5. Order (decreasing) the assessment activities cost-benefit ratios $CB_{ij} = [S_i(L) * \Delta P_{ij}(L)]/ E_{ij}$ Label them *T(k)* where $k = 1,\ldots, n*m$. Set $E_{T(k)}$ to be the corresponding E_{ij} and $\Delta P_{T(k)}$ to be the corresponding $\Delta P_{ij}(L)$.
6. Graph the cumulative RE drop, $\mathrm{RE}_{\mathrm{total}} - \Sigma\Delta P_{T(k)}$ as a function of the cumulative effort $\Sigma E_{T(k)}$ where $k = 1,\ldots, n*m$.

This above process produces an ideal REerror reduction strategy that can be applied for sweet spot determination as presented earlier. When the sweet spot is determined, the strategy dictates to perform *T(k)* for $k = 1,2,3,\ldots$ until $\Sigma ET(k)$ exceeds the sweet spot effort. The algorithm assumes that the entire effort allocated for each *T(k)* will be expended. As a result, there may be more optimal with respect to absolute RE reduction, but multi-attribute optimization techniques such as simulated annealing could potentially be applied to find these. Since the algorithm is somewhat involved, an example to illustrate it is presented in Tables 3abcde, resulting in the REerror reduction strategy displayed in Figures 5ab.

Table 3a. Attribute Loss Size (Steps 1,2,3)

pbty : portability
rplc : replaceability
vnmt : vendor maturity
sblt : scalability
expn : expandbility

pdt = product testimonials
rvc = review checklist
ppy = product prototyping

pbty	rplc	vnmt	sblt	expn
50	40	40	70	50

Table 3b. (attribute *i*, technique *j)* Effort (Step 4)

E_{ij}	pbty	rplc	vnmt	sblt	expn
pdt	10	10	10	10	10
rvc	30	20	20	30	30
ppy	70	70	1	80	73

Table 3c. (attribute *i*, technique *j)* RE Reduction (Step 4)

$\Delta P_{ij}(L)$	pbty	rplc	vnmt	sblt	expn
pdt	40	10	60	11	20
rvc	70	30	55	30	30
ppy	90	90	0	90	90

Table 3d. (attribute *i*, technique *j)* Cost-Benefit (Step 5)

CB_{ij}	pbty	rplc	vnmt	sblt	expn
pdt	200	40	240	77	100
rvc	117	60	110	70	50
ppy	64	51	0	79	62

Table 3e. Highest Cost-Benefit Sorted (attribute *i*, technique *j)* (Step 5)

T(n)	CB_{ij} sorted	$\Sigma E_{T(k)}$	$\Delta P_{T(k)}$	$RE_{total} - \Sigma\Delta P_{T(k)}$
1	240.00	10	240	896
2	200.00	20	40	656
3	116.67	50	70	616
4	110.00	70	55	546
5	100.00	80	20	491
6	78.75	160	90	471
7	77.00	170	11	381
8	70.00	200	30	370
9	64.29	270	90	340
10	61.64	343	90	250
11	60.00	363	30	160
12	51.43	433	90	130
13	50.00	463	30	40
14	40.00	473	10	10
15	0.00	474	0	0

While somewhat involved, the algorithm is fairly straightforward to implement and use. For example, the authors performed all analysis for this paper completely within a spreadsheet.

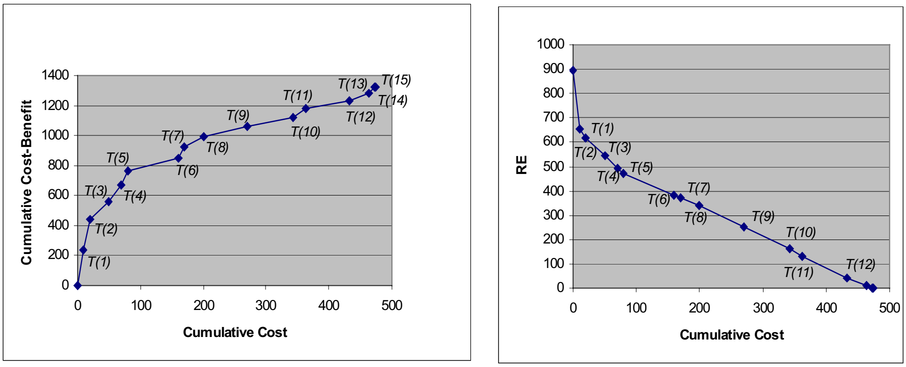


Fig. 5. (a) Cost-Benefit T(k) (b) REerror reduction T(k) (Step 6)

5 The USCCS Project

The approach to assessing COTS assessment described in this work was utilized within the USC Collaborative Services System project (all documentation is available in MBASE archives in Center for Software Engineering at USC). The customer for this project was the Information Service Division (ISD) at USC. The USCCS project is a COTS assessment-intensive project. The system provides online collaborative services for students, faculty and staff at USC who found that nowadays their tremendous need for group-oriented collaboration cannot be effectively met via traditional means such as email, phone call or in-person meetings. The project assessed and evaluated the diversity of current online collaboration solutions and COTS products and, eventually, identified a cost effective solution based on the COTS evaluation results. There were many COTS candidates. The candidates were quickly filtered down to three COTS packages:

- EProject: www.eproject.com,
- Iplanet: www.iplanet.com
- Blackboard: www.blackboard.com

A thorough assessment on cost-effectiveness needed to be done before choosing one of them. For this project, Boehm's COTS assessment approach [24] [13] was used to determine the S(L) and P(L). A sample of some of the assessment attributes and their corresponding weights that were used is shown in Table 4.

Table 4. Sample COTS Evaluation – Criteria and weight

No.	Attributes	Description	Weight
1	Inter-component Compatibility	The ability of two or more systems or components to perform their required functions, exchange information while sharing the same hardware or software environment.	90
2	Product performance	The degree to which a COTS component perform its functions within given timing constraints.	150
3	Functionality:	The degree to which a COTS component has the functional capability needed by the system. For determinators, referring to the feature checklist in table 8.	500

The functionality of a COTS component was determined by a feature checklist a sample of which is shown in Table 5 an concluded in Table 6. The data in Table 1ab was generated from these assessment activities.

Table 5. Sample COTS Evaluation – Feature Checklist

weight	features		
		weight	Sub_features
0.05	User authentication	0.025	User Login
		0.025	New User Registration
0.1	Create new group	0.1	
0.05	Group Page	0.05	
0.15	chat room	0.03	Easy to use
		0.03	Speed
		0.03	Can the chat content be saved?
		0.03	Support voice chat?
		0.03	Can the chat content be printed out?

Table 6. COTS Detail Assessment Results

Weight	Evaluation Criteria	eProject		iplanet		Blackboard	
		Ave.	Score	Ave.	Score	Ave.	Score
90	Inter-component compatibility	6.4	576	7	630	8	720
150	Product Performance	8.8	1320	9	720	7	1050
500	Functionality	8.239	4119.5	5	500	7.28	3640
60	Documentation understandability	8.2	492	9	540	9	540
80	Flexibility	6.6	528	8	640	8	640
50	Maturity of the product	9.8	490	10	500	9	450
80	Vendor support	9.2	736	9	720	9	720
150	Security	9.2	1380	9	810	9	1350
150	Ease of use	8.4	1260	8	720	7	1050
80	Training	7	560	7	560	8	640
60	Ease of Installation/Upgrade	7	420	7	420	8	480
60	Ease of maintenance	6.2	372	8	480	8	480
100	Scalability	8.4	840	9	900	8	800
60	vendor viability/stability	8.6	516	8	480	9	540
100	Compatibility with USC IT infrastructure	6.6	660	10	1000	9	900
80	Evolution ability	7.2	576	8	640	9	720
90	Ease of Integration with third-party software	9	810	8	720	8	720
1850	**Total**		**15655.5**		**14630**		**15940**
	Ave. (Average)		**8.06985**		**7.5412**		**8.261**

6 Conclusions and Future Work

COTS assessment appears qualitatively intuitive for any given aspect; however there are often many factors that must be considered simultaneously. The impact of all considerations may be quite complex and counter intuitive. This complexity shows up in particular when attempting to translate qualitative evaluations into quantitative results such as determining an optimal amount of assessment effort to expend a degree of detail for a multitude of COTS assessment attributes and assessment techniques. REerror and REdelay profiles are a tangible and practical means of assessing individual aspects that affect COTS assessment efforts. The total risk exposure is computed as the sum of the individual risk exposures (a linearity condition) and optimal total values will indicate optimal values for the individual risk contributions. In this way individual effort can be allocated in such a way to achieve minimum total risk exposure. Typically it is fairly straightforward to estimate individual risk exposures. COTS assessment attributes such as those listed in Table 1a provide tangible guidance as to which areas to assess. Critical delay risks are straight forward to estimate especially with the use of principled effort estimation els such as COCOMO and COCOTS [24]. Risk-based design can help answer difficult development questions before they become serious problems such as what COTS assessment approaches are sufficient and how much assessment is enough. Our approach also can be used to improve COTS assessment approaches. When the sweet spot is not acceptable with a given assessment strategy, look for better assessment techniques.

In the future, we would like to refine our approach to include variable individual assessment durations (perhaps via simulated annealing) and inclusion of additional critical project points. We would also like to apply options theory and portfolio analysis to provide rational valuation of assessment options and track changes to these valuations over time or effort.

Acknowledgements. We gratefully acknowledge Dr. Barry Boehm for originally suggesting this topic and the numerous subsequently stimulating discussions. We also thank Dr. David Klappholtz for serving as a fine sounding board.

References

1. V.Basili, B.Boehm, "COTS-Based Systems Top 10 List", IEEE Computer, Vol. 34, No. 5, May 2001
2. Ncube, C., Maiden, N. A. M. (1999). “PORE: Procurement-Oriented Requirements Engineering Method for the Component-Based Systems Engineering Development Paradigm”, 1999 International Workshop on Component-Based Software Engineering.
3. Cornelius Ncube and Neil Maiden. “COTS software selection: The need to make tradeoffs between system requirements, architectures and COTS components”, COTS workshop. Continuing Collaborations for Successfull COTS Development, 2000
4. Alves, C., Finkelstein, A., "Challenges in COTS-Decision Making: A Goal-Driven Requirements Engineering Perspective", Workshop on Software Engineering Decision Support, in conjunction with SEKE'02. Ischia, Italy, July 2002
5. J. Kontio, “OTSO: A Systematic Process for Reusable Software Component Selection”, CS-TR-3478, 1995. University of Maryland Technical Reports. University of Maryland. College Park, MD.
6. Dukic, L., “Non -Functional Requirements for COTS Software Components”, Workshop Ensuring Successful COTS Development. May 2000
7. John C. Dean, “Timing the Testing of COTS Software Products”, Proceedings of the 1st Workshop on Testing Component Based Systems, Los Angeles, CA. May 17, 1999. pp.5-8. NRC 41625
8. Alves, C. Castro, J. “CRE: A Systematic Method for COTS Components Selection”, XV Brazilian Symposium on Software Engineering(SBES) Rio de Janeiro, Brazil, October 2001.
9. Vu Tran and Dar-Biau Liu. “A Risk-Mitigating Model for the Development of Reliable and Maintainable Large-Scale Commercial-Off-The-Shelf Integrated Software Systems”, in Proceedings of the 1997 Annual Reliability and Maintainability Symposium, pp361-67, Jan 1997.
10. J. Jeanrenaud and P. Romanazzi. "Software Product Evaluation: A Methodological Approach". In Software Quality Management II:Building Software into Quality, pp59-69, 1994.

11. Ochs, M.; Pfahl, D.; Chrobok-Diening, G.; Nothelfer-Kolb, B., "A Method for Efficient Measurement-based COTS Assessment and Selection - Method Description and Evaluation Results". Pro. of the Seventh International Software Metrics Symposium METRICS 2001, April 2001, London, pp. 285-297
12. B.C. Phillips and S. M. Polen, "Add Decision Analysis to Your COTS Selection Process", Software Technology Support Center Crosstalk, April 2002.
13. Barry Boehm, Dan Port, Ye Yang, Jesal Bhuta, Chris Abts, "Composable Process Elements for Developing COTS-Based Applications", 2002
14. Dean, J. Vidger, M. "COTS Software Evaluation Techniques". Proceedings of The NATO Information Systems Technology. Symposium on Commercial Off-the-shelf Products in Defence Applications, Brussels, Belgium. April 2000
15. D. Kunda and L. Brooks, “Applying Social-Technique approach for COTS selection”, Proceedings of 4th UKAIS Conference, University of York, McGraw Hill, April 1999.
16. S. Gregor, J. Hutson, and C. Oresky, "Storyboard Process to Assist in Requirements Verification and Adaptation to Capabilities Inherent in COTS", Proceedings of ICCBSS, February 2002, Orlando, Florida USA, 2002, pp 132-141.
17. X. Burgu´es, C. Estay, X. Franch, J.A. Pastor, and C. Quer, "Combined selection of COTS components", Proceedings of ICCBSS, February, Orlando, Florida USA, 2002, pp 54-64.
18. S. Comella-Dorda, J. C. Dean, E. Morris, and P. Oberndorf, “A process for COTS Software Product Evaluation”, Proceedings of ICCBSS, February 2002, Orlando, Florida USA, pp 86-92.
19. G. Ruhe, "Intelligent Support for Selection of COTS Products", in Proceedings of the Net.ObjectDays 2002, Erfurt, Springer 2003
20. Carr, M. J.; Konda, S. L.; Monarch, I.; Ulrich, F. C. & Walker, C.F., "Taxonomy-Based Risk Identification", Software Engineering Institute, Carnegie Mellon University, Technical Report CMU/SEI-93-TR-6, ESC-TR-93-183, June, 1993
21. Hall, E. M., Managing Risk, Addison Wesley Longman, 1998
22. Boehm, B. “Software Risk Management: Principles and Practices”, IEEE Software, Jan. 1991, p.32-41
23. M. F. Bertoa, A. Vallecillo. "Quality Attributes for COTS Components". In Proc. of the 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2002). Málaga, Spain, June 2002.
24. Boehm. B., Abts, C., Brown, A.W., Chulani, S., Clark, B., Horowitz, E., Madachy, R., Reifer, D. and Steece, B., "Software Cost Estimation with COCOMO II", Prentice Hall, 2000.
25. M.Torchiano, L.Jaccheri. "Assessment of Reusable COTS Attributes", 2nd Int. Conference on COTS Based Software Systems (ICCBSS 2003), Ottawa, Canada, February 10-12, 2003
26. C. Abts, B. Boehm, and E. Bailey Clark, “COCOTS: A Software COTS-Based System (CBS) Cost Model,” Proceedings, ESCOM 2001, April 2001, pp. 1-8.

Legal and Contractual Implications in the European Union

Ignatio Delgado Gonzales

Martin & Lawson c/Alameda Urquijo, 28 – 2ºC
48010 Bilbao, Spain
idg@martinlawson.com

1 Product Liability and Product Safety

The Directive 85/374/EEC of 25 July 1985 on the approximation of the laws, regulations and administrative provisions of the Member States concerning liability for defective products, establishes the principle of objective liability or liability without fault of the producer in cases of damage caused by a defective product. If more than one person is liable for the same damage, it is joint liability. The producer (or the importer) is liable to compensate the death or personal injury and damages to property caused by the defective product, whether or not he is negligent.

The European company that introduces the COTS product in the European market, will be liable for the damages caused by the product (COTS) to the European consumers. If an European company incorporates and assembles COTS from outside the EU in a product, and that product is marketed in the EU, it will be liable for the damages caused to the European consumers. That is why it is very important to ask for secure and reliable COTS specially when buying outside the E.U..

It can be certified by a neutral third party like certification bodies and technological institutes or regulate in a contract clause.

The Council Directive 92/59/EEC of 29 June 1992 on general product safety Product means any product intended for consumers or likely to be used by consumers, supplied whether for consideration or not in the course of a commercial activity and whether new, used or reconditioned.

The companies selling their COTS products in the E.U. have to comply with the above mentioned regulation, and must facilitate all the information needed for a correct and safe use of the product. The COTS users can always ask for the information to the COTS producer.

The new revised Directive on General Product Safety (2001/95/EC) has to be transposed into national legislation by 15 January 2004. The new Directive maintains the existing requirements, but in addition introduces a number of new or reinforced provisions.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 199-202, 2004.

2 EU Competition Law

The EU legislation on competition has to be taken into account in the following areas: restrictive agreements and concerted practices, abuse of a dominant position, and mergers.

3 Consumer Protection

The COTS are considered products not services, so COTS users in the European Union are consumers of a product. The directive on liability for defective products is applicable.

The European concept of Consumers does not include juridical persons. However, Member States are able to extend the level of protection and consider firms, companies, etc…, as consumers in their respective national laws.

4 Data Protection

In order to ensure a high level of protection within the EU, data protection legislation has been harmonised. The Commission also engages in dialogues with non-EU countries in order to insure a high level of protection when exporting personal data to those countries.

5 Contractual Matters

Generally speaking, national contract law regimes lay down the principle of contractual freedom. Accordingly, contracting parties are free to agree their own contract terms. However, the laws and court decisions of a particular state govern each contract. Some of these national rules are not mandatory and contracting parties may decide either to apply these rules or to agree different terms instead. Other national rules, however, are mandatory, in particular where there is an important disparity between the positions of the contracting parties. Normally these different national regimes do not create any problems for cross-border transactions, as parties can decide which law will govern their contract. By choosing one national law, they accept all the mandatory rules of that law, as well as those non-mandatory rules, which they do not replace by different terms. In the event that an European consumer (COTS user) buys directly from US company, US law will apply to that contract, it is a more restrictive and less protective system for the consumer (COTS user). In the same case if the European consumer (COTS user) bought it from an European distributor or agent of the US vendor, European legislation will apply, is it the only one related to the parties and the contract.

6 Taxation

When selling COTS in the EU the companies have to check the TARIC code in order to pay the custom duties, also it is important to the different VAT applied in each EU Member State.

7 Intellectual Property Rights; Copyright/Patent

This topic related to COTS products acquisition is the most important, because it has to deal with software licenses. Licenses are used not only for proprietary software but also freeware, shareware and open-source software. The License is a cost that can vary depending on the negotiations with the COTS vendor, so it is important to know your rights when dealing this issue. It is still pending in European legislation whether to protect software invention via patent or copyright as it has been up to now. Patents may be applied for, processed and granted either at the European Patent Office (EPO) under the centralised system of the European Patent Convention (EPC), or via national patent offices in the Member States according to national law. However whichever route is chosen, national law applies in all cases after grant. Thus, granted European Patents become national patents which have to be validated, maintained and litigated separately in each Member State.

Patent and copyright protection are complementary. The same program may be protected by both patent and by copyright law. That protection may be cumulative only in the sense that an act involving exploitation of a particular program may infringe both the copyright in the code and a patent whose claims cover the underlying ideas and principles of the invention using the program.

8 Conclusions

- COTS are goods not services.
- Goods offered to a general public in a standard way, as shrink wrapped licenses or adhesion contracts.
- Contracting terms are into discussion whether COTS can be purchased or licensed.
- COTS users are considered consumers in the EU and they can benefit form the protection given by the EU legislation.
- Contractual terms and conditions in the EU market that weaken consumer protection are void.
- Importer of COTS from outside the EU is responsible for damages of the product.
- When acquiring COTS by E-commerce beware of conditions and legislation to apply, specially click wrapped contracts.
- Buying COTS does not imply buying copyrights.
- Licensing and renting COTS implies that you will never be the owner of the product.
- EU and US law are not only contradictory but also incompatible.

- European COTS' users must be aware of the fact that when contracting with vendors from the US, the contract may include provisions imposing the application of US law and US jurisdiction.
- US Law is more restrictive for COTS users than EU Law.
- The way to avoid application of US Law could be to make an intracommunitary agreement of the relationship between the vendor and the user: contracting with a European branch of the US vendor will avoid US Law application. Those conclusions are general and should be considered as such, when you are dealing with COTS it is advisable to start from the very beginning with the help of your legal adviser.

Best Practices for the Acquisition of COTS-Based Systems: Lessons Learned from the Space System Domain

Richard J. Adams[1] and Suellen Eslinger[2]

[1]The Aerospace Corporation
P.O. Box 92957 – M1/112
Los Angeles, CA 90009-2957
Richard.J.Adams@aero.org
[2]The Aerospace Corporation
P.O. Box 92957 – M1/112
Los Angeles, CA 90009-2957
Suellen.Eslinger@aero.org

The incorporation of Commercial Off-the-Shelf (COTS) software into software-intensive systems brings promises of significantly reduced cost and schedule, and improved reliability and maintainability, by using "proven" software. However, Air Force Space and Missile Systems Center (SMC) and National Reconnaissance Office (NRO) programs often find that the reality is very different!

In support of Air Force Space and Missile Systems Center's Directorate of Systems Acquisition, the authors performed an in-depth study of actual COTS-Based System (CBS) development and sustainment experiences on SMC and NRO programs. The use of COTS software poses major risks in the acquisition and sustainment of SMC and NRO software-intensive systems. The purpose of this study was to assist in mitigating these inherent software risks and to share COTS software lessons learned across SMC and NRO programs. The results of this study were presented at the Software Technology Conference (STC) 2001, and a paper "COTS-Based Systems: Lessons Learned from Experiences with COTS Software Use on Space Systems" was published in the STC 2001 conference proceedings.

This study determined six major lessons learned concerning CBS development and sustainment. These lessons are summarized as follows:

- Critical aspects of CBS development and sustainment are out of the control of the customer, developer and user.
- Full application of system and software engineering is required throughout the CBS life cycle.
- CBS development and sustainment require a close, continuous and active partnership among the customer, developer and user.
- Every CBS requires continuous evolution throughout development and sustainment.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 203–205, 2004.

- Current processes must be adapted for CBS acquisition, development and sustainment.
- Actual cost and schedule savings with CBS development and sustainment are overstated.

This study demonstrated that only careful acquisition, development and sustainment preparation and execution achieve the potential CBS benefits. CBS success depends upon preparing for a complex development and sustainment effort; preparing for inherent cost, schedule and performance risks beyond Government or developer control; and preparing to make adjustments to current acquisition, development and sustainment processes.

Clearly, one of the principal components of a successful software development project is the software engineering processes used. This statement is based on the well-established fact that the quality of a software product is highly dependent upon the quality of the processes used to develop and maintain that product. A great deal of work (especially by the Software Engineering Institute's COTS-Based Systems Initiative) has been done in the past few years toward improving the processes used for CBS development.

However, software acquisition processes (i.e., the processes used by the Government to acquire software-intensive systems) are also very influential in achieving a successful software development project. The software acquisition processes used can positively encourage, or adversely constrain, the developers in their application of high-quality software engineering processes to a software development effort. This is especially true when acquiring COTS-based systems.

Software acquisition best practices, by definition, are practices that have been identified through experience as being significant contributors to the successful acquisition of software-intensive systems. A comprehensive set of best practices must provide a consistent and integrated approach to software acquisition throughout the acquisition life cycle, both pre- and post-contract award. In addition, because software always exists within the context of the system, the software acquisition best practices must be consistent and integrated with a comprehensive set of system acquisition best practices. Finally, the set of best practices must be suitable for acquiring today's complex software systems that will be developed using the latest software development process and product technologies. A set of software acquisition best practices satisfying these criteria has been developed by the authors based on experiences from the space systems domain. A paper describing these best practices was presented at the 2nd OSD Conference on the Acquisition of Software-Intensive Systems and is available from the website: http://www.sei.cmu.edu/products/events/acquisition/.

This experience presentation addresses software acquisition best practices that are specific to the acquisition of COTS-based software-intensive systems. These best practices have been identified by the authors based on the experiences (both positive and negative) of recent space programs with CBS acquisitions. The best practices are directed toward enabling a program to effectively implement the above six lessons learned. Some of the frequently asked questions concerning the acquisition of COTS-based software-intensive systems are:

- How can a program specify contractual requirements to ensure only appropriate trades are made between requirements and COTS capabilities?
- How can a program ensure that the delivered system will contain no COTS software that is already unsupported by the vendor?
- How can a program select a bidding team with mature, high quality CBS development and maintenance processes?
- How can a program best incentivize a contractor to use mature, high quality processes for CBS development and maintenance?
- How can a program ensure that their contractor has performed an adequate evaluation of the proposed COTS software packages?
- What actions can a program do to reduce risk in their CBS acquisition?
- How can a program prepare for the inherent uncertainty in CBS development and acquisition?
- How can a program plan for adequate funds for CBS development and sustainment throughout the system life cycle?

This presentation addressed these and other questions important to the successful acquisition of COTS-based software-intensive systems.

Managing Vulnerabilities in Your Commercial-Off-The-Shelf (COTS) Systems Using an Industry Standards Effort (CVE)

Robert A. Martin

MITRE Corporation

Organizations around the world, in every type of industry and market, are moving towards networks that are based on the Internet protocols. In addition, third-party commercial and open source software has become a critical element to these organizations and the infrastructure of networks, utilities, and services they rely upon to function. That means the software problems in these COTS software products can quickly cause significant difficulties for any organization. When such software problems have security implications, they are referred to as "vulnerabilities."

CVE, the Common Vulnerabilities and Exposures Initiative [cve.mitre.org], is a new international, community-based effort from industry, government, and academia that is working to create an organizing mechanism to make finding and fixing these COTS and open source software product vulnerabilities more rapid and efficient.

As recently as 1999, system administrators and security analysts were faced with a cacophony of naming methods for defining individual security problems in software. This made it difficult to assess, manage, and fix these vulnerabilities and exposures when using the various vulnerability services, tools, and databases. The problem was compounded with the software suppliers' various update announcements and security alerts. While trying to properly research options for selecting some new security tools for its own corporate networks, MITRE began designing a method to sort through this vulnerability naming confusion. The approach involved the creation of a unified reference list of software vulnerability and exposure names that were mapped to the equivalent items in each tool and database.

THE CVE LIST. The common names in the CVE List are the result of open and collaborative discussions of the CVE Editorial Board, along with various supporting and facilitating activities by MITRE and others. The Board includes prominent information security specialists from numerous information-security-related organizations around the world, including commercial security-tool vendors, academic and research institutions, and government agencies. From the original 12 in 1999, the Board now includes 49 members from 35 organizations. With MITRE's support, the Board identifies which vulnerabilities or exposures to include on the CVE List and agrees on the common name, description, and references for each entry. MITRE maintains the CVE List and Web site, analyzes submitted items, moderates Editorial Board discussions, and provides guidance throughout the process to ensure that CVE remains objective

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 206–208, 2004.

and continues to serve the public interest. From the original 641 names in 1999, the list of CVE names has grown to 6,430 items as of January 2004, as shown in Figure 1.

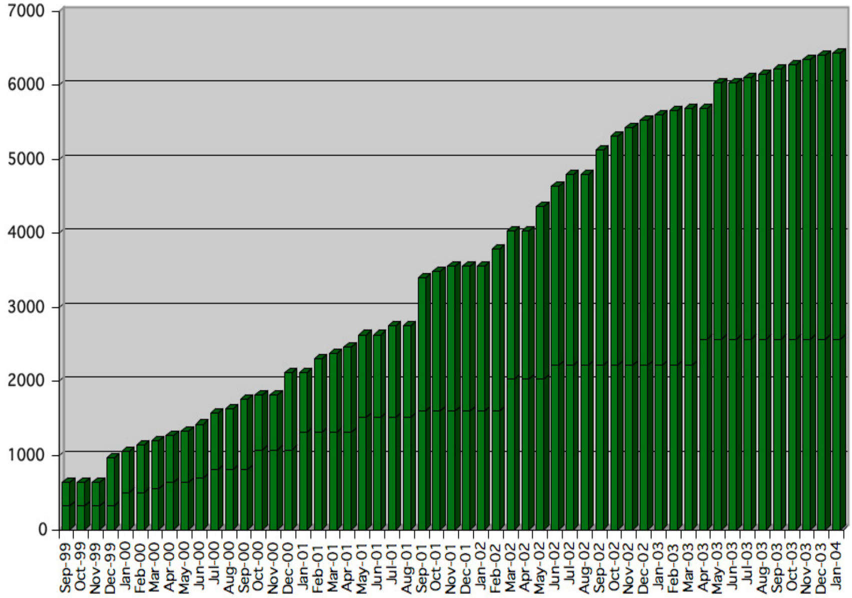


Fig. 1. CVE growth over time.

CVE Compatibility. The basic premise of the CVE List is that there be one name for a vulnerability or exposure. A CVE-compatible product or service must understand the CVE names for vulnerabilities and allow its users to interact with the product/service in terms of those CVE names. This does not mean that the product/ service only uses CVE names for vulnerabilities, but rather that in addition to its own native label for a vulnerability, it is aware of the CVE name for that vulnerability. This support for CVE names is central to the concept of CVE compatibility and ensures the CVE-compatible product uses CVE names in a way that allows users to correlate its information with other products that also use CVE names.

Uses of CVE Compatibility. Integrating vulnerability services, databases, Web sites, and tools that incorporate CVE names will provide an organization with more complete and efficient coverage of security issues. For example, a report from a vulnerability scanning tool that uses CVE names will enable the organization to quickly and accurately locate fix information in one or more of the CVE-compatible databases and Web sites. It is also possible to determine exactly which vulnerabilities and exposures are covered by each CVE-compatible tool or service, because the CVE List provides a baseline for comparison. After determining which of the CVE entries apply to its platforms, operating systems, and commercial software packages, an organization can compare this subset of the CVE List to any particular tool's or service's coverage.

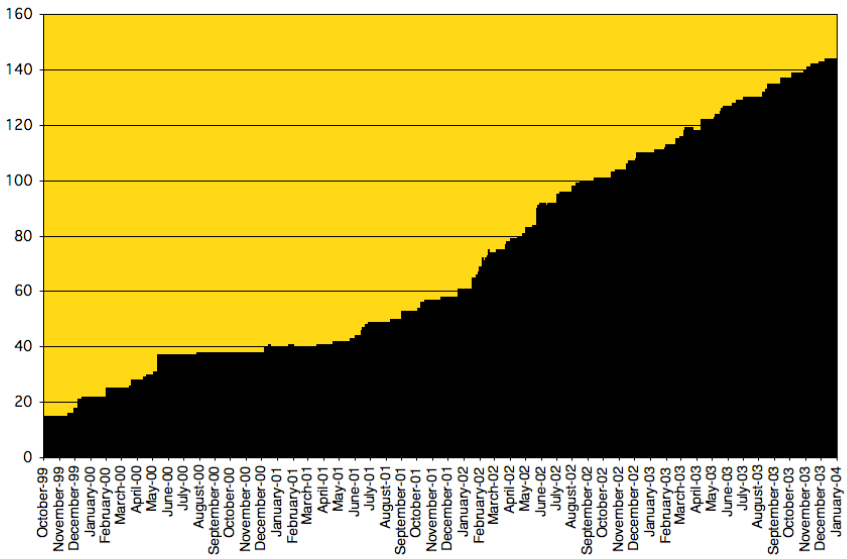


Fig. 2. CVE –Compatible product growth over time.

The U.S. National Institute for Science and Technology (NIST) has published a recommendation to all federal government agencies and services for the use of CVE-compatible products and services whenever possible [NIST SP800-51]. Similarly, the Department of Defense, in their new 8500.2 Information Assurance Implementation guidance, gives preference for products that support the CVE naming standard.

Growth of CVE-Compatible Products and Services. The list of organizations declaring CVE-compatible products and services is continuously expanding and is international in scope. In October 1999, 15 products intended to be CVE-compatible. As shown in Figure 2, as of January 2004, 97 organizations are working toward compatibility for 144 products or services.

Challenges and Opportunities. As CVE moves forward, it faces a variety of challenges and opportunities. Challenges include addressing the impact of vulnerability disclosure practices on CVE accuracy (including vendor acknowledgement and replication of the vulnerability); identifying the proper scope CVE; and renumbering the CVE List. At the same time, opportunities include analyzing vulnerability causes, improving vulnerability testing methods and veracity, filling in some gaps in research (such as analysis of configuration problems and vulnerabilities), and delivering real improvements in the way organizations manage the risks from vulnerabilities and exposures.

Costing COTS Integration

Linda Brooks

This session will provide a roadmap for doing an estimate for COTS-intensive systems development. It will discuss the key lessons learned about COTS development pitfalls and translate them into estimation steps and checklists. The following topics will be discussed: frequently overlooked and underestimated tasks, the components of a complete estimate, estimating size and cost drivers, using parametric modeling for estimation, and risk analysis. An example of the calibration of a parametric model using historical data from a Northrop Grumman Mission Systems COTS-intensive development will be presented, along with sample worksheets for compiling an estimate.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, p. 209, 2004.

U.S. Coast Guard, Differential GPS, Nationwide Control Station

Frank Klucznik, Kristi McRacken, John Killers, and Jason Judy

This presentation discusses the lessons learned using COTS products on the United States Coast Guard's Nationwide Differential Global Positioning System (NDGPS). The Coast Guard manages a NDGPS for the United States Department of Homeland Security, where the system is required to provide accuracy within 3-meters and 99.7% system signal availability. Due to the environment, most of the COTS products evaluated and selected for the system were pushed beyond their design limits. This pushed the development team to develop creative and innovative solutions to overcome the challenges. The NDGPS was a very challenging engineering project that combined an Online Transaction Processing (OLTP) environment with a graphical user interface, which required real-time updates. It also incorporated the storage of operational data with archive data, which allowed users to run data analysis on the same hardware as the operational software. While this is not a desired state, it allowed the fielding of a system in a very short period of time that met initial customer requirements. A spiral software development approach will be used to evolve the system to the desired state over the next few years.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, p. 210, 2004.

Requirements Analysis and Management (RAM) of COTS-Based Systems – A "Success Story"

Gail M. Talbott, Program Director
Lockheed Martin (LM) Asset Solution Integration (ASI)

1102 John Glenn Blvd., Suite A
Titusville, FL 32780
gail.m.talbott@lmco.com

Abstract. As Commercial-Off-The-Shelf (COTS) software solutions are developed and integrated to address and enhance government and commercial industries business processes needs, specific and systemic issues are coming to light. Two essential key process areas – Requirements Management and Software Product Engineering testing – must be innovatively accomplished by documented and institutionalized processes and procedures, in order to preclude the erroneous utilization of resources to address poorly defined requirements and subsequent associated rework during the implementation of a COTS-based system. This presentation is based on the experience of Lockheed Martin Asset Solution Integration (ASI) as a COTS product solution integrator in the Enterprise Asset Management (EAM) marketplace, and will describe the proven approach to requirements analysis and management and implications of that methodology developed during our successful COTS-based systems implementations and deliveries.

1 Introduction

Lockheed Martin ASI's experience indicates a well-defined process for performance of Requirements Management and test management aspects of Software Product Engineering is critical to ensuring successful COTS-based system implementation. When good sound software engineering methodologies for these key process areas are institutionalized, quality products are delivered and customer satisfaction – the main goal of all COTS-based System Integrators – is achieved.

2 The Business Case – A Required Method

Initially, ASI was not required to meet any formal software engineering or related standards in order to support a large-scale COTS software implementation effort. This allowed ASI to use an existing homegrown Change Request log to track customer requests for changes to/tailoring of the customer-specified integrated COTS-based system. All customer requests were considered valid and were worked on and bundled based on the impact to specific COTS software modules. With this approach, there was not a

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 211–215, 2004.

requirement for Engineering Review Board (ERB) or Change/Configuration Control Board (CCB) review and approval. In addition, there was no formal allocation of requirements to the design or visible traceability of customer requests to design steps, test case steps, or test discrepancies. ASI had no clearly defined process for eliciting, documenting, managing, assessing, baselining, and tracing requirements from identification through customer verification and acceptance.

On one project, requirements were gathered and implemented "on-the-fly" by members of the software engineering group – the developers – without formal documentation and review, and without adequate understanding of data relationships and business processes. The project was "halted," and re-started "from scratch," with much of the previous work simply discarded. As work progressed on the COTS software implementation project, ASI recognized the need to improve its software engineering business practices. ASI was selected to participate in a new company sponsored initiative to achieve a Software Engineering Institute (SEI) Capability Maturity Model for Software (SW-CMM) Level 3 rating. Requirements Management and the Software Product Engineering, which includes test, were two of the key process areas that were addressed and improved. The initiative enabled ASI to become a successful System Integrator of COTS-based systems. ASI's new well-defined and institutionalized approach to performing requirements management and testing of integrated COTS products had repeated success on the large-scale implementation effort. The end result was a very satisfied customer and an institutionalized and successful Requirements Analysis and Management (RAM) methodology for COTS-based systems.

3 The Requirements Analysis and Management (RAM) Methodology – A Proven Approach

ASI established a formal requirements definition and change control process, managed by Engineering and Configuration Control Boards, and implemented a state-of-the-art requirements management tool – Rational Requisite Pro, commonly referred to as ReqPro. ASI's Requirements Analysts use ReqPro to track requirements, software design/change steps, and test steps, providing complete traceability "from womb to tomb." Traceability ensures all approved requirements are addressed, and that all resulting software changes are tested. ReqPro's flexibility allows ASI to document multiple types of requirements with varying levels of detail for each project, as well as defining requirement attributes and values. This re-usable capability is copied to subsequent projects, with additional tailoring as required. Typical requirement attributes are: Status, Business Process, Requestor, Priority, Testability, Difficulty, Stability, Planned Build, Affected Item(s), and at what point they were added to the baseline. All requirements are tracked, whether customer driven or elicited by the Requirements Analyst based on Contract Line Items (CLINs) in a Statement Of Work (SOW).

The Customer, along with the Requirements Manager and peers in the Integrated Product Team (IPT), review the documented requirements to ensure they meet the established quality standards, that they are accurate, complete, clearly stated and unambiguous, consistent with each other, and testable, prior to baselining. ASI defines up front as many of the requirements as possible, to support COTS product selection (if not pre-selected by the customer) and Rough Order of Magnitude (ROM) estimates. As part of the ASI modified-COTS Waterfall Life Cycle, final, more detailed requirements are documented as the project progresses in detailed specifications, including "child requirements" which may require individual validation and verification. Data, Process, and Function models are used where appropriate to assist in communicating the Customer's requirements and business processes to the ASI software engineering (development) staff. Requirements are baselined at defined milestones in the project life cycle: the Functional Baseline (FBL) upon customer approval at Functional Requirements Review (FRR); the Allocated Baseline (ABL) upon customer approval at the Design Review; and the Integrated Product Baseline (PBL) upon customer approval at the Operational Readiness Review (ORR) (following successful Integrated and User Acceptance Testing (I/UAT)).

A Requirements Traceability and Verification Matrix (RTVM) is produced from the PBL. The RTVM provides the visible traceability of customer requests to design steps, test case steps, or test discrepancies. ASI tracks changes to baselined requirements (i.e., new features, enhancement requests, or clarification of existing requirements) in a database separate from the requirements. When changes to requirements are needed, they must be approved by the ERB and CCB and then added to the appropriate requirements baseline prior to implementation. Requirements, and changes to requirements, are communicated to affected groups for analysis of feasibility, difficulty/complexity, severity/risk, identification of affected COTS components or other software, relationship to other changes (e.g., constraints, logical packaging/bundling), and impacts to schedule and cost. ASI also tracks the root cause of each test discrepancy and product defect. The results of these metrics are used for "lessons learned" to improve the requirements management process for subsequent projects. The requirement to create software products using a defined and systematic approach has been proven over time.

One successful approach to meeting this requirement has been the interpretation and implementation of the Software Engineering Institute (SEI) Capability Maturity Model for Software (SW-CMM) to perform software engineering. Efficient software system deliveries depend on a variety of factors that are associated with software development. These factors include not only developing software code to meet system requirements, but also ensuring customer requirements associated with cost, schedule, and accountability are satisfied.

4 The RAM Methodology – Successful Results

Defining a formal, more rigorous, approach to the Requirements Management and Software Product Engineering key process areas – the application of SEI SW-CMM Level 3 discipline – increases the time needed for the RAM and testing phases of the project life cycle. ASI's experience demonstrates this approach has improved the quality of our software products and significantly reduced the amount of rework required to resolve discrepancies and/or defects. Case studies by the Air Force and Raytheon have determined that requirement errors typically comprise over 40% of all errors in a software project, and finding and fixing requirement errors can consume 70-85% of the total project rework costs. ASI's Requirements Management and test processes, coupled with peer reviews, comprehensive documentation of design, thorough testing, and verification of approved requirements, have resulted in a major reduction of post-production problems. ASI has successfully delivered all of the last five software builds for the large-scale COTS-based system implementation with zero defects after institutionalizing these key process areas.

5 The RAM Methodology and Implications of COTS-Based System Implementation

When automating business practices with COTS software, successful system integrators must help the customer find a balance between two extreme approaches to implementation:

1. Deploying the COTS as-is ("plain vanilla"), with minimal configuration changes, but with major changes to the customer's business practices, or
2. Tailoring/Customizing the COTS product to the customer's current business practices.

Neither of these approaches is desirable; in the first, the customer and end users often consider it too risky or difficult to completely change their business practices; in the second, the customer may lose all the benefits of having purchased COTS software due to the major customization effort required to adapt it to current business practices. ASI's experience validates the following assertion made by Serge Charbonneau, a Rational Software Engineering Specialist; in his Rational Edge article "Using RUP to Implement a Packaged Application," Charbonneau explains that any organization implementing a COTS application should find a balance between these two extremes. This has several implications to the Requirements Analysis and Test Management processes. They are:

1. The Requirements analysis team must be familiar with both the proposed COTS product functionality and limitations and with the customer's business practices. This familiarity is required to assist the customer in defining requirements that leverage the inherent functional capabilities of the COTS with minimal impact to the customer's business practices, while allowing for customization that ensures customer satisfaction.
2. The familiarity with both the product functionality and the customer business practices allows the Requirements analysis team to perform a "gap analysis" of proposed requirements against the COTS software, and document the deltas that are not already native functions of the COTS product.
3. Knowledge of native COTS software functionality and the customer's business practices allows the test analysts and engineers to develop Test Plans and Test Case Specifications (for IST and/or UAT) for the tailored/customized capability

of the COTS software per the customer's established business practices. The only native functions of the COTS product that are addressed by this testing are those required to successfully execute the business practices using the tailored/customized COTS product.

COTS Selection and Adoption in a Small Business Environment: How Do You Downsize the Process?

William B. Anderson

This presentation describes the experiences of the Software Engineering Institute (SEI) under the auspices of the Technology Insertion Demonstration and Evaluation (TIDE) program in collaborating with two small businesses as they selected, inserted, and evaluated an Integrated Manufacturing Execution System (IMES). IMES is the manufacturing sector's enterprise wide business management software suite, integrating business quotation to execution, billing, and financial reconciliation. The comprehensive nature of the application makes it critical to the ongoing operation of the business and touches virtually every aspect of the enterprise. This presentation summarizes the approach and lessons learned as the SEI 'right-sized' large-enterprise COTS processes to better suit the dynamics of these small enterprises.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, p. 216, 2004.

Managing the COTS Chaos: Experiences from the Trenches Using the Evolutionary Process for Integrating COTS-Based Systems

Lisa Brownsword[1]and Minton Brooks[2]

[1]Software Engineering Institute, Arlington, VA USA
llb@sei.cmu.edu
[2]Indpendent Consultant
minton.brooks@comcast.net

With increasing pressure to remain competitive in their respective markets, many commercial companies are turning to a greater use of COTS products to provide more capability faster to their end-users. Early attempts to leverage COTS products using existing waterfall-oriented development approaches have met with many failures. As a result, organizations are searching for more viable alternatives. This presentation shares the early experiences of one commercial project to identify and transition from a waterfall development process to a process explicitly designed to leverage the commercial marketplace and other sources of existing components to form delivered solutions.

The chosen process was the Evolutionary Process for Integrating COTS-based systems (EPIC), a risk-driven, collaborative, spiral development process that builds on the IBM Rational Unified Process (RUP) to define, build, field, and evolve an integrated solution with COTS products and custom and legacy components. This presentation will summarize why this process was selected, the obstacles encountered, the key challenges overcome, and the value added.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, p. 217, 2004.

Characterization of a Taxonomy for Business Applications and the Relationships Among Them[¥]

Juan P. Carvallo[1], Xavier Franch[1], Carme Quer[1], and Marco Torchiano[2]

[1] Universitat Politècnica de Catalunya UPC-Campus Nord (C6)
08034 Barcelona, Catalunya, Spain
{carvallo, franch, cquer}@lsi.upc.es
http://www.lsi.upc.es/~gessi

[2] Politecnico di Torino
C.so Duca degli Abruzzi, 24
10129 Torino, Italy
torchiano@polito.it
http://softeng.polito.it/torchiano

Abstract. In this paper we propose a taxonomy for classifying COTS business applications, i.e. products that are used in the daily functioning of all types of organizations worldwide, such as ERP systems and document management tools. We propose the identification of characterization attributes to arrange the domains which these products belong to, and also we group these domains into categories. We define questions and answers as a means for browsing the taxonomy during COTS selection. We show the need of identifying and recording the relationships among the domains and propose the use of actor-oriented models for expressing these relationships as dependencies. Last, we explore the definition of quality models for the domains, to be used in COTS selection, focusing on their reusability and stepwise definition downwards the hierarchy.

1 Introduction

The amount of Commercial Off-The-Shelf (COTS) products [1, 2] available on the market is growing more and more. This tendency is due both to the increasing adoption of component-based software technologies by the community, and to the continuous creation of new communication and marketing channels that bridge the gap between providers and consumers of those products. Therefore, there is an increasing need for organizing the types of available COTS products to achieve more efficient and reliable selection processes.

Such a need is especially felt in the *business applications* (BA) [3] context. All the aspects of the daily operations of small, medium and large organizations, either private companies or public administrations, heavily depend on the existence of adequate software products to undertake crucial tasks, such as accounting, human re

[¥] This work is partly supported by CICYT TIC2001-2165 and WISE IST-2000-30028.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 221–231, 2004.

sources management, document administration, team work, people communication, business processes monitoring, etc. Therefore, having specific means for discovering which BAs satisfy the needs of an organization is utterly convenient in order to select the most suitable.

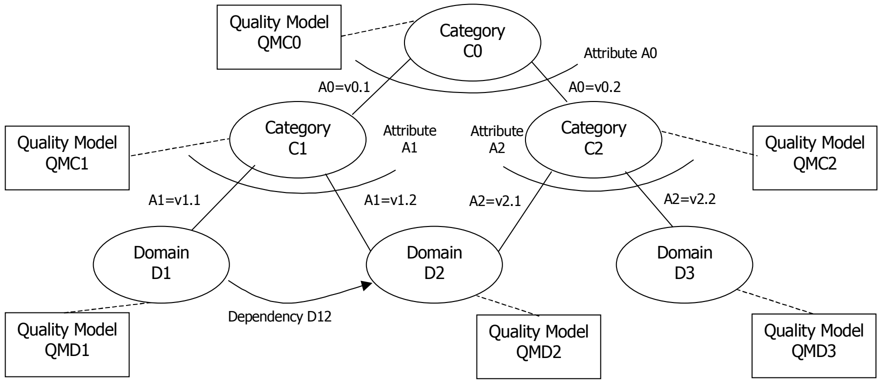


Fig. 1. The fundamental elements of a taxonomy

The purpose of this paper is to provide support for improving the BAs selection processes. Our proposal is built upon a taxonomy of BAs (see fig. 1). We consider that BAs belong to one or more BA domains which appear as leafs in the taxonomy. A domain encloses a significant group of functionality; they are grouped into categories, which in their turn can be grouped to form a multi-level taxonomy. We also represent dependencies among the domains. The taxonomy can be adopted in our COTS selection practices, which use of quality models to assess the adequacy of components with respect to requirements [4, 5]: we attach quality models to nodes in the taxonomy, supporting model reuse by inheriting them downwards the hierarchy.

Several taxonomies can be found not only for BAs but for other domains too [6, 7, 8]. But more important than the concrete form that a taxonomy takes, is the rationale behind its construction, i.e. which are properties that may help to organize the BAs and how the taxonomy can be searched. This is especially true when considering not just the construction of the taxonomy, but its evolution. Categories and domains may be arranged with respect to various characterization attributes. We propose to use as rationale the notion of characterization attribute as introduced in [9, 10].

The rest of the paper is structured as follows. Section 2 introduces a conceptual model for the proposal. Section 3 presents some highlights of the taxonomy for BAs. Section 4 addresses to the relationships among BAs, while section 5 focuses on the use of the taxonomy for the definition of quality models. Finally, section 6 presents the conclusions and some future work.

2 A Conceptual Model for COTS Taxonomies

A formal representation of the concepts that appear in a valid taxonomy can be provided using a conceptual model that uses the UML notation [11] (see fig. 2). A taxonomy is composed of *categories* and *domains,* generalized as *scopes*. Domains are *grouped into* categories and categories on their turn are *grouped into* other categories. A category C shall be partitioned into sub-categories, each corresponding to a *value* of a *characterization attribute* (*attribute*, for short) that *applies to* C. Also *dependencies* of four different types may be declared between domains (see section 4); the ternary association allows more than one dependency among two particular domains.

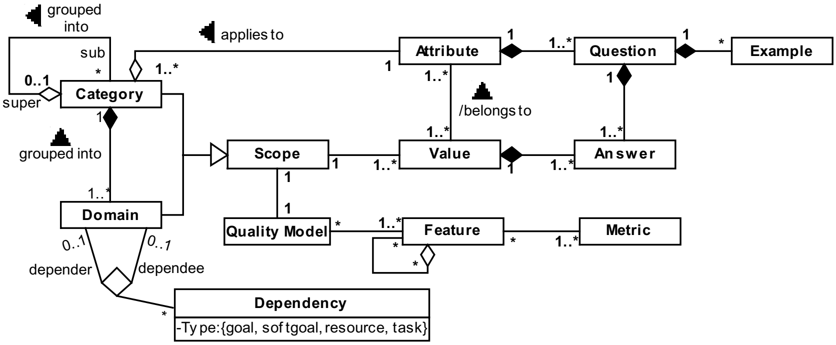


Fig. 2. Conceptual model for COTS taxonomies (integrity constraints are not included).

To simplify the classification of BAs and the identification of domains of interest during selection processes, we associate one or more *questions* to attributes of each category. When two or more questions are associated to an attribute they apply different criteria. People involved in selection processes will answer these questions and, since each *answer* is associated with an attribute value, they will browse the taxonomy at a more abstract level than using directly characterization attributes and their values. *Examples* can be associated to questions to clarify their understanding.

Finally, we define quality models for each scope in the taxonomy. A *quality model* [12, 13] is defined as a multilevel hierarchy of quality *features*. Quality features are measurable and their values are computed using some *metric*. The quality models associated with each sub-category must differ from each other at least by one quality feature. If a partition of a subcategory does not enrich the quality models then it only adds "noise". That is, it makes the taxonomy more complex and adds a question that has no purpose; therefore we apply the Occam's razor ("one should not increase, beyond what is necessary, the number of entities required to explain anything").

Attributes in general are not orthogonal, i.e. an attribute used to split a subcategory depends on the attribute used to split the super-category. In fig. 1, attribute

A1 that is used to split category C1, makes sense because attribute A0 has value v0.1 while it could be meaningless if applied to category C2 for which A0 equals v0.2.

3 A Set of Characterization Attributes for Business Applications

Since reuse is the main motivation for this work, we apply it also in our research method. Instead of starting from the scratch, we preferred to take an existing one as starting point and then refactor it to obtain a taxonomy conforming to the criteria defined above. Given the particular type of systems we are addressing, we investigated the way that professional software consultant companies organize the BAs' services that they offer to their customers, and we selected the classification of one of them, which is application-oriented and well-suited for our purposes, compared to others whose classification is based on business areas or that includes not only software but other assets.

The refactoring process we performed was based on the identification of a set of characterization attributes and their associated values, questions and answers. Once the attributes were defined, they were used as the rationale to rearrange the BA categories and domains defined in the original taxonomy, with the goal of obtaining a final taxonomy as close as possible to the original one.

An excerpt of the taxonomy is presented in table 1. As first step during refactoring, we noticed that although the root of the original taxonomy was partitioned into eight sub-categories, we couldn't find an attribute that could be used to support it. Therefore we introduced intermediate categories. At this point an attribute able to discriminate among the original subcategories was needed. We found that a good option is looking at the number of users; we considered therefore two main categories, corresponding to two values of the attribute *number of users*: *single user* and *multi-user*. Single user systems are used typically by one person to work on his/her own data, while multi user systems operate on information shared by several people. We identified two different questions to elucidate the attribute's value:

- *How many users has the system*? Answers *One* or *More than one*.
- *Does the system reconcile the interests of many stakeholders*? Answers *yes* or *no*.

Starting from this first classification, we identified subsequently other characterization attributes, their corresponding values, questions, and answers (see table 2). The questions are applied at different levels in the taxonomy, and some of them are applied in more than one branch (e.g., see rows 3 and 7 in table 2). Please note the correspondence among the questions' level and the taxonomy structure.

After we finished the process, we identified the differences among the original and the resulting taxonomies. We summarize the results below.

- *Identification of new scopes*. This is the most usual action we have taken, due to the nature of our activity. Many examples exist: categories such as *Single User Systems* and its heirs *Management* and *Operational Tools* or domains as *Workflow*.

Table 1. Partial view of the BA taxonomy.

Categories					Domains
Level 1	Level 2	Level 3	Level 4	Level 5	
a. Sing-le-User Systems	c. Manage-ment	…	…	…	…
	d. Operational	…	…	…	…
b. Multi-User Systems	e. Internal Software	f. Information Systems	h. ERP Software		Integrated ERP Software.
					Financial Management Software
			i. Knowledge Management Software	l. Capture and Forms Processing Software	Optical Character Recognition
					Forms Capture Software
				m. Content Management Software	Document Management
					Content Courseware
					Records Management
					Web Content Management.
				n. Knowledge Pre-sentation Software.	Virtual Classrooms
					Information Access
		g. Collabora-tion Software	j. Team Support Software.		Workflow
					Versioning &Concurrence Control
			k. Workgroup Comm. Suites		Real Time
					Off-Line

- *Division of existing scopes.* Occasionally, some original scopes were too coarse-grained mixing different concepts belonging to different characterization attributes. This is the case for instance of *Collaboration and Knowledge Management Software* which has been split into *Collaboration Software* and *Knowledge Management Software* (the latter as subcategory of *Information Systems*).
- *Merge of existing scopes.* The other way round, some scopes are so similar that their differentiation does not really makes sense (e.g., the *Business Metrics* and the *Corporate Performance Management Software* domains).
- *Promotion of domains to categories*. Some proposed domains could be actually categories, although we have not found examples of this situation.
- *Degradation of categories to domains*. Complex domains such as *Enterprise Content Management Software* were originally considered as categories although there exist products covering their intended functionalities (which makes them domains).
- *Removal of existing scopes*. Not very often, but some scopes appearing in the original taxonomy have been removed. Three main reasons behind:
  - The scope is not a software domain, e.g. *Voice and Call Processing Equipment.*
  - The scope does not really add value to the taxonomy.
  - The scope does not really belong to the category of BAs. For instance, the *Instant Messaging* or *Audio and Video Conferencing* domains are technical means for some team work domains.

Table 2. An excerpt of the attributes, questions and answers to browse the taxonomy.

Level	Cat.	Attribute	Values	Question	Answers
1	Root	Number of users	Single user, Multi-user	How many users has the system?	One, More than one
				Does the system rec-oncile the interests of many stakeholders?	No, Yes
2	a	Objective	Management, Operational	Is for management or operation?	For Management, For Operation
3	c	Orientation	Data, Process	Is it data or process oriented?	Data-oriented, Process-oriented
4	…	Data processing	Acquisition, Storage, Preparation, Analysis	What type of data processing does it perform?	Acquisition, Storage, Preparation, Analysis
3	d	Utility	Technical, Office	Is it technical or office oriented?	Technical-oriented, Office-oriented
2	b	User's location	Internal, Internal &External	Where are the users located?	Just inside the company, Inside & outside the company
3	e	Orientation	Data, Process	Is it data or process oriented?	Data-oriented, Process-oriented
4	f	Data type	Operation, Support	What type of data does it process?	Operation, Support
4	g	Type of group work	Coordination, Communication	What is the use of the system?	Coordinate teamwork, Communicate people
3	…	User's role	B2B, Customer	Is it for suppliers or customers?	For suppliers, For customers

It is important to remark that the taxonomy we obtained is just one of the possible taxonomies for BAs. The goal here is not to find the "best" (if any exists) taxonomy but show how it is possible to build one.

4 Identifying Dependencies Among Business Applications

The analysis of any segment of the COTS market shows that COTS products are not designed to operate isolated; instead, they work together and therefore some relationships may be established between them. This is especially true in COTS products of a large granularity, such as the BAs considered in this paper.

Commonly, COTS products of certain domains depend on others with different aims [14, 15]. Among others we mention:

- *Enabling their functionality*. A product from a domain requires a product from other domain to provide a given functionality. E.g., in order to follow document life-cycles, document management tools need workflow technology to define them.

- *Complementing their functionality.* A product from a domain requires a product from another domain to offer an additional feature, not originally intended to be part of its suitability. For instance, a web page edition tool can complement a web browser to facilitate the edition and modification of web pages.
- *Enhancing their quality attributes.* A product from a domain requires a product from another domain to improve its quality of service. For instance, resource utilization can be improved significantly using compression tools.

As far as we know, in the context of COTS selection, these dependencies among COTS domains have been dealt with in a case-by-case basis, i.e. they have been discovered in each new selection process, with the only exception of product lines architectures. Remarkably, the proposals of domain taxonomies mentioned in the introduction of the paper do not include these relationships.

It is our strong belief that dependencies among domains shall be identified and recorded explicitly for their repeated use during different selection processes. Specifically, these dependencies help organizations involved in a selection process to find out that some goals that they want to achieve with a COTS of a domain will not be satisfied if they do not have or procure COTS from other domains. For instance, an organization selecting a document management tool would discover quickly that they need a document imaging tool for scanning and storing paper documents.

Furthermore, we think that domain taxonomies provide a great opportunity for including this information as an additional element for structuring COTS domains. In fact, we may say that taxonomies are arranged along two different dimensions, one using characterization attributes and the other using dependencies. The selection process can use the first dimension to determine the domain of interest through a repeated question-answer pattern, and then the second dimension to determine the additional products required.

Actor-oriented approaches allow the optimal identification and formal representation of dependency relationships between COTS products from different domains in a taxonomy. Specifically we propose the use of *i** SD models [16], where actors represent COTS products and dependencies are established among them. In a dependency relationship, a *depender* COTS product from a domain depends on *dependee* COTS products from other domains. The four types of dependencies of i*, namely goals, soft goals, tasks and resources, allow us to represent different relationships between domains:

- *Goals*: The *depender* depends on the *dependee* to achieve a new functionality. For the fact of being a goal dependency, the additional functionality can be provided in different ways. For example, the COTS products in the *Document Management* domain depend on COTS products of the *Web Content Management* domain to offer the functionality of *Visualization of Managed Documents through Web Pages*.
- *Soft goals*: The *depender* depends on the *dependee* to enhance some of its non-functional attributes. For example, the *Efficiency* of COTS products in the *Virtual Classroom* domain depends on others that in fact are not BAs but technologic components (i.e., belonging to a different domain).
- *Resource*: The *depender* depends on the *dependee* for its access to some resource. For example, the *Collaborative Engineering* COTS products need the *Definition of Resources* like users, user types and document types, to define the life cycle and

business processes of documents in an organization. This definition will be provided by a *Document Management* COTS product.

- *Tasks*: The *depender* depends on the *dependee* to carry out a task. For example the *Document Management* COTS products depend on COTS products from the *Content Management* domain to *Store Documents in a Certain Format*.

Figure 3 shows an *i** SD diagram that includes a subset of the dependencies identified in the taxonomy of BA during the analysis of the document management domain carried out during a COTS selection process in which we have been involved.

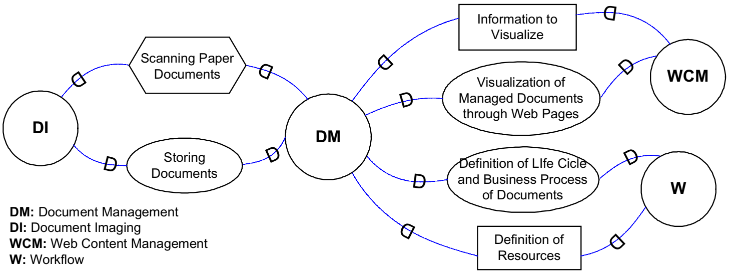


Fig. 3. *i** SD diagram with dependencies involving the *Document Management* domain

5 Organizing Quality Features Around the Taxonomy

Quality models provide a general framework to get uniform descriptions of the quality features that apply to a COTS domain. We have used them to support negotiation between requirements and product capabilities as shown in [4]. We have built quality models for some domains both in real-world and academic settings (some of them reported [5, 17, 18]). As a result, we claim that quality models enhance COTS selection reliability but, on the other hand, building quality models is a time-consuming activity. Also, we face the problem that new COTS domains appear continuously and building quality models for them from the scratch is really impractical. For these reasons, we search for methods and techniques that allow quality model reuse.

Reusability of quality models downwards categories and domains of the hierarchies is a way to support this objective. We have observed throughout our experiences that some quality features appear over and over, and this repetition is directly connected to the characteristics embedded in the characterization attributes. The recognition of COTS domains and categories improves reusability: once a new COTS domain has been identified, its quality model can be constructed by inheriting the features of the quality models for those COTS categories in the hierarchy which it belongs to. During the process, new categories may be identified, abstracting commonalities of this new domain with others. As a result, a quality model bound to a category of the taxonomy collects all the quality features common to all its subcategories and domains. Since then, any quality model for a particular selection process may reuse the quality model of the corresponding COTS domain.

Fig. 4 shows the stepwise refinement of quality models downwards the hierarchy (see [15] for details). The quality model bound to the root is proposed to be a refinement of the ISO/IEC 9126-1 quality model [13]. For instance, we split the *Suitability* subcharacteristic of this standard into two, *Basic Suitability* and *Added Suitability*, keeping apart those functionalities that are inherent of a COTS domain from those that have been added as the products in the domain are enriched in new versions. Then, each category and domain adds or redefines some quality features which are related to the corresponding characterization attribute's value. For example, the quality model for the *Multi-User Systems* category includes security features such as the support for *Login and Passwords*. This feature has been reused in both the *Internal Software* and the *External Software* category quality models and some additional features such as *Security Communication Protocols* has been added to the quality models of the latest one.

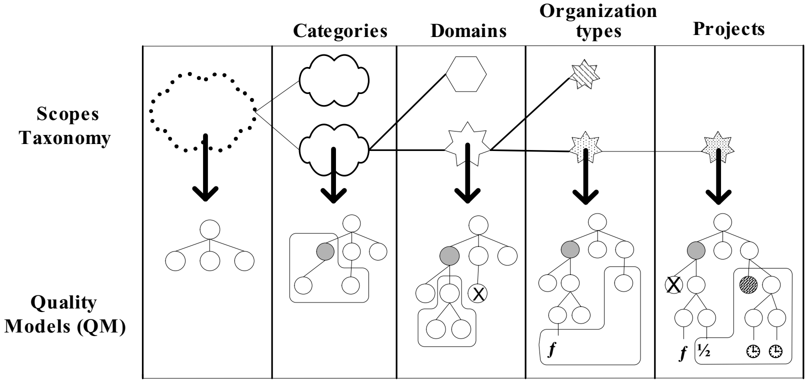


Fig. 4. Refinement of quality models downwards the taxonomy.

In our BA context, we also have observed that even quality models bound to domains such as *Virtual Classrooms* or *Integrated Enterprise Resource Planning Software* can be specialized to particular types of organizations. Consider two furniture manufacturing companies. One of them produces all the parts for its products from the basic materials while the other builds its products by assembling parts that have been previously ordered to different suppliers. As a result, the quality model for first company should be tailored to be more concerned on the control and reduction of manufacturing costs (e.g. *Definition of production budgets and schedules*, *Cost components tracking* and *Quality control*), while the other may be more concerned on the control of the logistics related to the process of buying the parts and their timely reception (e.g. *Purchase orders emission*, *Shipping scheduling and rescheduling* and the *Coordination of shipments from multiple providers*). We reflect this fact by introducing organization-type quality models.

Last, concrete COTS selection processes may take place with even a further refinement of a quality model, specifically tailored to the requirements of this process. For instance, the quality attributes for human-computer interface may be refined, or their metrics defined, to adapt to the particular interface requirements of the context of use of the system-to-be.

6 Conclusions

We proposed characterization attributes as the rationale for building taxonomies in any category of COTS components; questions, answers and examples simplify their use. Although introduced in the particular context of business applications, our approach can be applied to any other broad category of COTS products, such as scientific packages or life-cycle support tools. As far as we know, this rationale distinguishes our approach of other taxonomy proposals as [6, 7, 8].

We have demonstrated the feasibility of our approach in one of the most populated and critical fields of the COTS market. In particular, we have solved some problems observed in the departing hierarchy: categories were not defined precisely (as a consequence, their granularity was not always adequate); categories often overlapped; the criteria for decomposing categories was never declared and often was not evident, making hard the use of the hierarchy.

We represented explicitly the relationships between business application domains using the concept of dependencies present in actor-based models. As a result, the implications of the use of a particular business application become clearer. As far as we know, this information is not included in other taxonomy proposals.

Finally we showed how to define quality models for business application domains reusing part of them along the taxonomy; a category's quality model is inherited from the sub-categories.

Different actors may benefit from our approach:

- Software consultant companies offering assessment for business automation may structure their services better.
- Medium- and large-size companies with their own IT department may be more confident on their own selection.
- Software engineers which usually carry out COTS selection may structure better their knowledge and may aim at a better return of investment.

As future work, we mention:

- Application of the approach to other domains such as application development, software infrastructure and wireless service engineering.
- Instead of having a static taxonomy, the browsing process of the hierarchy could be built upon a dynamic taxonomy. At any moment, any sound attribute (i.e., set of answers) could be applied to discard some categories and select others. Selection of a category after giving a value to an attribute restricts the set of sound attributes during the browsing process.
- Definition of a COTS selection process around the taxonomy. The selection process will be based on three main activities, in addition to other usual ones:
  - Browsing the taxonomy for locating the COTS domain of interest.
  - Analyzing the dependencies for understanding the implications of the selection.
  - Refining the quality model to be used in the evaluation of components.

References

1. D. Carney, F. Long. "What Do You Mean by COTS? Finally a Useful Answer". *IEEE Software*, 17 (2): 83-86, March/April 2000.
2. B. Craig Meyers, P. Oberndorf. *Managing Software Acquisition*. SEI Series in Software Engineering, 2002.
3. IT Papers, “Business Applications”. http://www.itpapers.com/category/busapp.html
4. X. Franch, J.P. Carvallo. "Using Quality Models in Software Package Selection". IEEE Software, 20(1), January/February 2003, pp. 34-41.
5. J.P. Carvallo, X. Franch, C. Quer, "Defining a Quality Model for Mail Servers". 2nd International Conference on COTS-Based Software Systems (ICCBSS), LNCS 2580, Ottawa (Canada), 2003.
6. R. Glass and I. Vessey, "Contemporary Application-Domain Taxonomies" IEEE Software, 12 (4): 63-76, July 1995.
7. INCOSE, "Software Engineering Tools Taxonomy". http://www.incose.org/tools/tooltax/se_tools_taxonomy.html
8. CBSE Net, "Application Domain Taxonomy". Available on-line (previous registration) at: http://www.cbsenet.org/pls/CBSEnet/eco_ricerca_documenti.concept_search_frame
9. M. Morisio, M. Torchiano. "Definition and classification of COTS: a proposal". 1st International Conference on COTS-Based Software Systems (ICCBSS), LNCS 2255, Orlando (USA), 2002.
10. L. Jaccheri, M. Torchiano, "Classifying COTS products". 7th European Conference on Software Quality (ECSQ), Helsinki (Finland), 2002.
11. Unified Modelling Language (UML) 1.4 Specification. OMG document formal/ (formal/2001-09-67). September 2001.
12. ISO Standard 8402: *Quality management and quality assurance-Vocabulary*, 1986.
13. ISO/IEC Standard 9126-1 Software Eng. – Product Quality – 1: Quality Model, June 2001.
14. X. Franch, N. Maiden. “Modeling Component Dependencies to Inform their Selection”. 2nd Intl. Conference on COTS-Based Software Systems (ICCBSS), LNCS 2580, Ottawa (Canada), 2003.
15. P. Botella, X. Burgués, J.P. Carvallo, X. Franch, C. Quer. “Using Quality Models for Assessing COTS Selection”. 5th Workshop on Requirements Engineering (WER), Valencia (Spain), 2003.
16. E. Yu. “Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering”. 3rd IEEE International Symposium on Requirements Engineering (ISRE), 1997.
17. P. Botella, X. Burgués, J.P. Carvallo, X. Franch, J.A. Pastor, C. Quer. “Towards a Quality Model for ERP System Selection”. Chapter in *Component-Based Software Quality: Methods and Techniques*, A. Cechich, M. Piattini, A. Vallecillo (eds.), LNCS 2693, 2003.
18. X. Franch, J. Marco. "A Quality Model for the Ada Standard Container Library". In *Proceedings of the 8th ICRST-Ada Europe'03, LNCS 2655*. Toulouse (France). June 2003.

Author Index