LNCS 3009

Frank Bomarius
Hamid Iida (Eds.)

Product Focused Software Process Improvement

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

Springer

Lecture Notes in Computer Science 3009

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

Editorial Board:

Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Frank Bomarius Hajimu Iida (Eds.)

Product Focused Software Process Improvement

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

Springer

Volume Editors

Frank Bomarius
Fraunhofer Institute for Experimental Software Engineering
Sauerwiesen 6, 67661 Kaiserslautern, Germany
E-mail: frank.bomarius@iese.fraunhofer.de

Hajimu Iida
Information Technology Center, Nara Institute of Science and Technology
Takayama-cho 8916-5, Ikoma City, Nara 630-01, Japan
E-mail: iida@itc.aist-nara.ac.jp

Library of Congress Control Number: 2004102972

CR Subject Classification (1998): D.2, K.6, K.4.2, J.1

ISSN 0302-9743
ISBN 3-540-21421-6 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Protago-TeX-Production GmbH
Printed on acid-free paper SPIN: 10994818 06/3142 5 4 3 2 1 0

Preface

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

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

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

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

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

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

January 2004

Frank Bomarius
Hajimu Iida

Conference Organization

General Chair

Seija Komi-Sirvio, VTT Electronics (Finland)

Organizing Chair

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

Program Co-chairs

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

Tutorial, Workshop, Panel Chair

Shinji Kusumoto, Osaka University (Japan)

Publicity Chairs

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

PROFES Advisor

Markku Oivo, University of Oulu (Finland)

Program Committee

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

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

Table of Contents

Software Process Improvement

Software Quality

Measurement

Methods and Tools

Experimental Software Engineering

Industrial Experiences

Agile Methods

Software Process Assessment

Requirements Engineering

Software Reuse / COTS

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

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

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

Abstract. Advances have been made in the development of software process improvement (SPI) standards and models, i.e. Capability Maturity Model (CMM), more recently CMMI, and ISO's SPICE. However, these advances have not been matched by equal advances in the adoption of these standards and models in software development which has resulted in limited success for many SPI efforts. The current problem with SPI is not a lack of a standard or model, but rather a lack of an effective strategy to successfully implement these standards or models. In the literature, much attention has been paid to "what activities to implement" rather than "how to implement" these activities. We believe that identification of only "what activities to implement" is not sufficient and that knowledge of "how to implement" is also required for successful implementation of SPI programmes.
The aim of this research paper is to empirically explore the viewpoints and experiences of practitioners regarding SPI implementation and based on the findings to develop a model in order to guide practitioners in effectively implementing SPI programmes. This SPI implementation model has six phases and provides a very practical structure within which to implement SPI programmes in organizations.

1 Introduction

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

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

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

26. Vu N. Tran, Dar-Biau Liu: Application of CBSE to Projects with Evolving Requirements- A Lesson-learned. Proceeding of the 6th Asia-Pacific Software Engineering Conference (APSEC' 99) Takamatsu, Japan, December (1999) 28 - 37
27. Padmal Vitharana: Risks and Challenges of Component-based Software Development. Communications of the ACM, August (2003), 46(8): 67 - 72
28. Katharine Whitehead: Component-Based Development: Principles and Planning for Business Systems. Addison-Wesley, (2002)
29. http://www.idi.ntnu.no/grupper/su/spike.html
30. http://www.ifi.uio.no/~isu/INCO/

Managing COTS Components Using a Six Sigma-Based Process*

Alejandra Cechich[1] and Mario Piattini[2]

[1] Departamento de Ciencias de la Computación, Universidad Nacional del Comahue, Buenos Aires 1400, 8300 Neuquén, Argentina
acechich@uncoma.edu.ar

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

Abstract. While the objectives of Six Sigma are to reduce variation and prevent defects, it is also a management philosophy that includes the need for fact-based decisions, customer focus, and teamwork. In attempting to build a COTS integrated system, selection of candidates typically pays attention to specify search criteria and goals to be met. Yet they often overlook some elements in the process such as fact-based decisions and teamwork, which might drive the process helping increase the probability of success. In this paper, we describe and illustrate a Six Sigma-based proposal for the process of selecting and integrating COTS components. Our approach also suggests some tools and measures to be applied during its specific phases.

1 Introduction

As Component-based Software Development (CBSD) starts to be effectively used, some software vendors have commenced to successfully sell and license commercial off-the-shelf (COTS) components. CBSD advocates the use of pre-fabricated pieces, perhaps developed at different times, by different people, and possibly with different uses in mind. The goal, once again, is the reduction of development times, costs, and efforts, while improving the flexibility, reliability, and maintainability of the final application due to the (re)use of software components already developed, tested and validated.

The particular nature of COTS components (black-box binary entities developed by third parties, and independently deployable in unforeseen contexts) imposes specific quality mechanisms to be put in place for their effective integration in the devel-

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

F. Bomarius and H. Iida (Eds.): PROFES 2004, LNCS 3009, pp. 553–567, 2004

opment life cycle of software applications. Firstly, components need to be properly documented and offer measurement mechanisms in order to be assessed and evaluated for selection. Secondly, both the quality and some of the extra-functional properties of the final (composed) system heavily depend on the individual properties of its constituent components, and therefore some traceability between the quality attributes at these two levels is required. Finally, the use of third-party COTS components may also introduce risks, such as potential harmful side effects to the system, or quality degradation of the final product. COTS component testing and certification may offer partial solutions to these problems [5].

It is important but often difficult to determine the appropriate level of Component-Based Systems (CBS) quality. Fitness for use implies that the appropriate level of CBS quality is dependent on the context, and determining the quality is difficult when different users have different needs. Even the field of component quality has experienced significant advances during its relatively brief history, CBS quality has not yet advanced to the point where there are standard measurement methods, and few enterprises routinely measure COTS component quality. However, some efforts have started to define software metrics to guide quality and risk management in a CBS by identifying and quantifying various factors contributing to the overall quality [1,18].

On the other hand, Six Sigma is an approach to product and process improvement that has gained wide acceptance and has delivered large business benefit across many industries [12]. While the objectives of Six Sigma are to reduce variation and prevent defects, it is also a management philosophy that includes the need for fact-based decisions, customer focus, and teamwork. Six Sigma precepts can guide stakeholders through the selection and integration of COTS components and help increase the probability of success. In [19], the Six Sigma approach has been suggested selecting packaged software, however the evaluation mainly relies on the information provided by demos and additional documentation of the COTS software. Then, the lack of measures makes this process perfectible.

In this paper, we present a preliminary approach that can be used in the process of selecting and integrating COTS components in conjunction with Six Sigma precepts. The major goal of our work is to reduce the effort required to select COTS components by defining a framework based on customer focus, fact-based decisions, and teamwork. Particularly, when reasoning about provided and required services of components, we analyse them in terms of committed functionality. In Section 2 of the paper we introduce the main characteristics of our approach. Section 3 then presents the detailed phases by using a motivating example. Section 4 adds some remaining issues and finally, we conclude with an overview and future research directions.

2 A Six Sigma-Based Process: Motivation and Overview

Six Sigma is typically divided into five phases, creating what is referred to as DMAIC, which is an acronym for the following phases [19]:

1. *Define* the problem and identify what is important: Identify the problem and the customers; define and prioritise the customer's requirements; and define the current process.
2. *Measure* the current process: Confirm and quantify the problem; measure the various steps in the current process; revise and clarify the problem statement, if necessary; and define desired outcome.
3. *Analyse* what is wrong and potential solutions: Determine the root cause of the problem; and propose solutions.
4. *Improve* the process by implementing solutions: Prioritise solutions; and develop and implement highest benefit solutions.
5. *Control* the improved process by ensuring that the changes are sustained: Measure the improvements; communicate and celebrate successes; and ensure that process improvements are sustained.

Selection and integration of COTS components need to ensure, as in any Six Sigma project, that decisions are based on facts and that customer's requirements have been considered. However, in the continuing attempt to introduce CBSD, organisations have problems identifying the content, location, and use of diverse components. Component composition requires access to tangible and intangible assets. Tangible assets, which correspond to documented, explicit information about the component, can vary from different vendors although usually include services, target platform, information about vendors, and knowledge for adaptation. Intangible assets, which correspond to tacit and undocumented explicit information, consist of skills, experience, and knowledge of an organisation's people.

Six Sigma might help to put all these pieces together and define a measure-based procedure for selecting COTS components. As the selection process implies, the software is packaged with the goal that it will be used by many different systems. To introduce our approach, two kind of systems are referred to as follows:

- *The source system.* We refer to a source system as a set of pieces of software that running all together satisfy part of the user's requirements expressed as functional and non-functional properties, whose quality is able to be measured.
- *The Component-Based System (CBS).* We refer to a component-based system as a system that uses at least one component in conjunction with other components, legacy systems, and other pieces of software – including COTS components – to satisfy user's requirements. The concept of CBS is introduced to emphasize the fact that the output from the system satisfies the user's requirements by using the functionality supplied by at least one COTS component.

System satisfaction can refer both CBS and the source system – not necessarily both CBSs. In Figure 1 (a), the source system is stable in the sense that its outputs O_1 and O_2 satisfy the requirement R_1 (a previously stated and measurable requirement), although there is still a requirement (R_2) not provided by the system.

Let us introduce our case. Suppose that requirement R_2 in Figure 1 (a) can be satisfied by using the functionality provided by a COTS component. It could be the case that this COTS component alone can provide the entire functionality of R_2 – as Figure 1 (b) shows – by means of the output O_3 and using some inputs from the system. The point here is to determine whether the system – now a CBS system – remains stable. In other words, whether the system still preserves the previously quality attributes and the new attributes defined for R_2. Furthermore, how does the system preserve those attributes? Have the interfaces changed? Is quality affected? Have we introduced side effects that make the quality poor? It could also be the case that we need to introduce major changes to the system as Figure 1 shows.

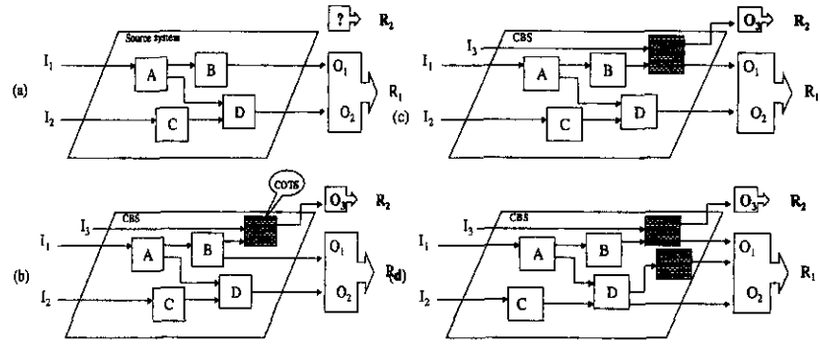

Fig. 1. Cases of an altered source system

In Figure 1 (c), the COTS component is more complex than in the case of Figure 1 (b), providing more functionality by replacing some outputs from the source system. As we can see, component B is not directly contributing to the requirement R_1 but supplying an input interface to the COTS component, that in turn contributes to reach the requirement R_1. In Figure 1 (d) a second COTS component is added providing that outputs from B have to be complemented with new outputs from D due to effects on the composition's functionality.

The process of selecting COTS components and detecting the degree of system satisfaction can be guided by a Six-Sigma approach. The five-phases of the COTS component selection process are shown in Figure 2. Deliverables are summarised in the figure to indicate the existence of a documentation suite composed of criteria for selection, and measures that identify suitability with respect to a source system. Tools in the figure refer to well-known tools for quality assessment as well as specific tools and measures for selecting COTS components.

Let's briefly introduce the phases of Figure 2, when applying the Six Sigma approach to our COTS selection case:

1. *Define* system's stability [6] and stakeholder's preferences. The Define phase sets the goals and context of the project. This phase is concerned with identifying key stakeholders, and determining their needs and criteria for selection. Some of the activities are as follows:
   - Given a source system, we should specify the current degree of satisfaction/stability of the system under evaluation by defining goals, constraints, component context, domain as well as product's quality attributes.
   - Then, we should identify sub-domains and relationships, i.e. relations among pieces of software from an architectural point of view – legacy systems, COTS and standard components – and relations among subdomains.

2. *Measure*. This phase is concerned with assisting the selection process by creating a complete description of the COTS candidates. Some of the activities are as follows:
   - Measure COTS components to determine their suitability with respect to the required services.
   - Assess the impact of the integration of COTS components on a stable system.

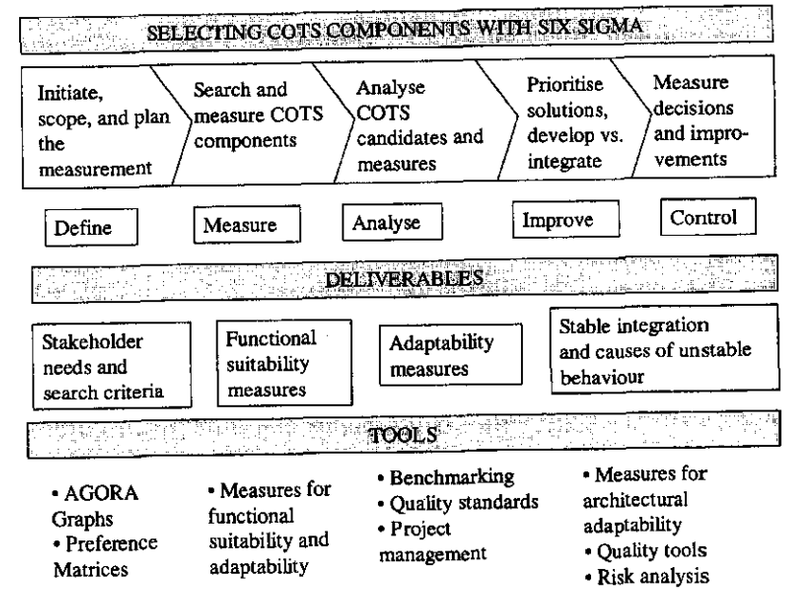

Fig. 2. Five-step measuring process

3. *Analyse.* In this phase, a COTS candidate or a set of COTS candidates is selected from several alternatives. Decisions are made based on previous definitions and measurements. Some of the activities are as follows:
   - Detect sub-domains affected by the recently incorporated COTS.
   - Identify dissatisfactions – functional dissatisfactions and accuracy dissatisfactions – according to the stability state established in phase 1.
   - Determine dissatisfaction's relative magnitudes referring the initial system's stability.
   - Decide whether perturbations affect the initial stability.

4. *Improve.* This phase is concerned with ensuring that the selected candidates can be integrated and supported within the required quality. For example, causes of low architectural adaptability should be determined, along with causes of functional dissatisfaction. Some of the activities are as follows:
   - Identify dissatisfaction causes (i.e., using Ishikawa diagrams).
   - Prioritise solutions deciding on integrating the COTS component/s.

5. *Control.* In this phase a successful COTS selection procedure should communicate its practices and suggests possible corrective/adaptive actions to sustain its success.

3 Moving into Selecting COTS Components

The following sections present our process by using an example of COTS component selection. Due to brevity reasons, only some of the activities involved in each phase are further detailed here.

3.1 Moving into the Define Phase

To ensure that decisions are fact based, it is important that the "define" phase will be reinforced. Although it is always important to understand the current process and the problem to be solved, the importance is greater when selecting COTS components because requirements and services must be balanced as part of a negotiation procedure.

Firstly, COTS component's required functionality should be expressed to define search goals and criteria for evaluation. To do so, we have adapted the model in [1], which explores the evaluation of components using a specification-based testing strategy, and proposes a semantics distance measure that might be used as the basis for selecting a component from a set of candidates. This model supposes that there is an architectural definition of a system, whose behaviour has been depicted by scenarios or using an architecture description language (ADL).

Our proposal [7], has adapted the model as a basement for quality measurement. We express the semantics distance in terms of functional suitability measures, which

provide a better identification of the different COTS component functionalities. Then, a system can be extended or instantiated through the use of some component type. Due several instantiations might occur, an assumption is made about what characteristics the actual components must possess from the architecture's perspective. Thus, the specification of the architecture *A (SA)* defines a specification S_C for the abstract component type *C* (i.e. $SA \Rightarrow S_C$). Given a specification S_C for an abstract component type *C*, a candidate component *K* to be a concrete instance of C must conform to the interface and behaviour specified by S_C. Although the process of selecting a component *K* consists of evaluating interface and semantic mappings, in our work only semantic mappings are addressed. Mappings in S_C, which represent the different required functionalities, are established between input and output domains. We focus on incompatibilities derived from functional differences between the specification in terms of mappings of a component K_i (S_{Ki}) and the specification in terms of mappings of S_C.

We will illustrate the procedure by using an E-payment system as an example. We suppose the existence of some scenarios describing the two main stages of the system – *authorisation* and *capture*. Authorisation is the process of checking the customer's credit card. If the request is accepted, the customer's card limit is reduced temporarily by the amount of the transaction. Capture is when the card is actually debited.

The scenarios will provide an abstract specification of the mappings of S_C that might be composed of:

- Input domain: (AID) Auth_IData{#Card, Cardholder_Name, Exp-Date}; (CID) Capture_Idata{Bank_Acc, Amount}.
- Output domain: (AOD) Auth_Odata{ok-Auth}; (COD) Capture_Odata{ok_capture, DB_update}.
- Mapping: {AID → AOD};{CID → COD}

Suppose we pre-select two components to be evaluated, namely K_1 and K_2 respectively. A typical situation for inconsistency in the functional mappings between S_{K1}, S_{K2} and S_C is illustrated in Figure 3, where dashed lines indicate (required) mappings with respect to S_C, and the solid lines are (offered) mappings with respect to S_{K1} (grey) and S_{K2} (black). Note that the input domain of the component K_1 does not include all the values that the specification S_C requires, i.e. the capture functionality is not provided. Besides, the input domain of the component K_2 includes more values than the required by S_C, although the mapping satisfies the required functionality. We should also note that there is another functionality provided by K_2, which might inject harmful effects to the final composition.

On the other hand, committing requirements for COTS component selection can be problematic. Traditionally, the requirements elicitation techniques have widely used a family of goal-oriented requirements analysis (GORA) methods, such as I* [13], and KAOS [16], as an approach to refine and decomposing the needs of customers into more concrete goals that should be achieved. Our proposal extends a version of a Goal-Oriented Requirements Analysis Method called AGORA in [15] considering additional features of COTS components.

Initial goals, as considered in AGORA graphs, are the needs of the customers that will be refined and decomposed into sub-goals one after another. Therefore, with the functional goals of the component specified by mappings in S_C, the next step is to refine the goals considering the perspectives of different stakeholders – reuse architect, certifier, business process coordinator, etc. Then, the computation of stakeholder's preference values for the refined goals will allow us to add preferences to mappings of S_C.

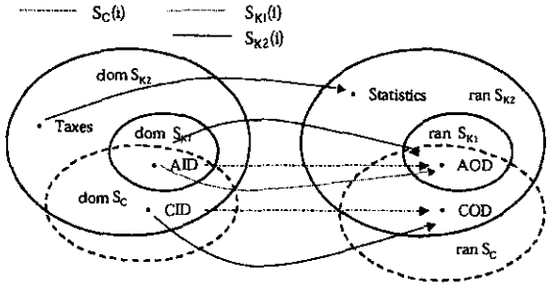

Fig. 3. Functional mappings of S_C and S_{K1}/S_{K2}

In our context of component-based systems, an AGORA graph describes the abstract specification of a required component (S_C) according to the scenario S. Figure 4 shows a snapshot of a possible AGORA graph for our E-payment case. Here, there are two types of conflicts on goals; one is the conflict between goals and the other one is the conflict on a goal between stakeholders. The first type of conflicts appears in Figure 4 between the goals "Prevent unauthorised debits" and "No authorisation", whose edge has a negative contribution value. The second type appears in the figure on the goal "Input AID data". The diagonal elements of the preference matrix show that the reuse architect gave the preference value 8 by himself, while the business process coordinator's preference is –5 given by himself (diagonal values on the matrix represent preferences given by a stakeholder on his own judgment).

When incorporating COTS components, goals should be balanced against COTS services. For example, using the main concepts of goal-oriented requirements engineering, the goal acquisition and specification process [2] includes the necessity of identify goals that help to distinguish between products (called core goals) from those that are provided by most available products (called peripheral goals). This characterisation would lead to dealing with a third type of conflicts on goals: conflicts between the abstract specification of a component and its possible instantiations. These conflicts should be resolved when the COTS component selection actually take place. Then, it is important to add some extra information on the graph, so the negotiation will be possible, i.e. the AGORA graph in Figure 4 has added the labels <core>/<peripheral> to facilitate the future negotiation. Besides the classification of goals as core and peripheral, the attributes desirability (level of importance for a goal

to be met), and modifiability (level in which a goal can be modified) are proposed as attributes for goal description when selecting COTS components [2].

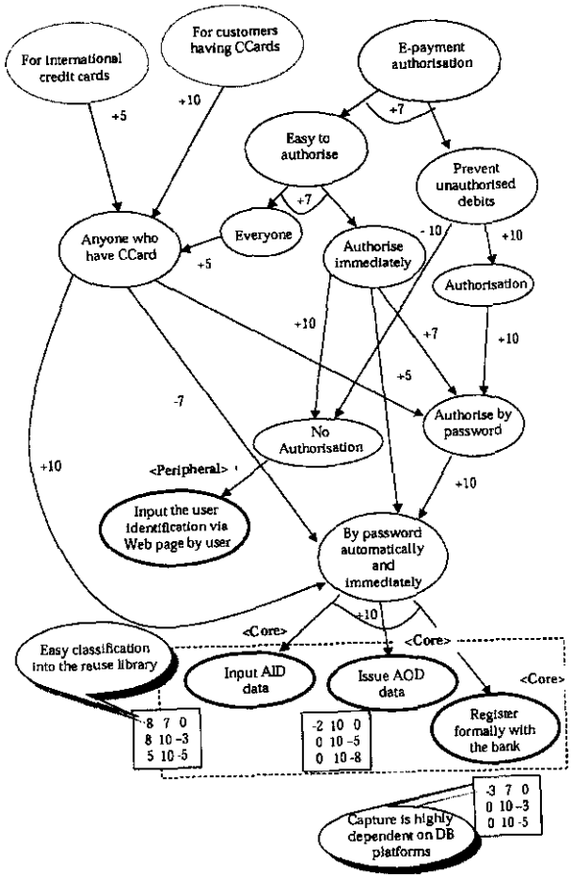

Fig. 4. A snapshot of the AGORA graph to balance stakeholder's preferences on required goals

By using an AGORA graph, we can estimate the quality of several properties of the adopted goals. Particularly, correctness is assumed as a quality factor that represents how many goals in a specification meet stakeholder's needs. Correctness in AGORA is strongly related to contribution values on the path of the adopted goal as well as on its stakeholder's preference value. Particularly, measures of modifiability and desirability of goals for selection may be derived from AGORA graphs.

Based on these measures, a weighted suite of required services can be more accurately specified. For example, we may suggest that desirability should be calculated and decisions should be made based on the following cases, where "agreement-threshold" is a indicated value between 0.6 and 0.7 and "core/peripheral" is the type of refined goal:

- Case 1: desirability(refined goal/s) < agreement-threshold ∧ core ⇒ Try other scenarios to get agreement
- Case 2: desirability(refined goal/s) < agreement-threshold ∧ peripheral ⇒ decision to discharge
- Case 3: desirability(refined goal/s) ≥ agreement-threshold ∧ peripheral ⇒ decision to retain
- Case 4: desirability(refined goal/s) ≥ agreement-threshold ∧ core ⇒ keep for the selection process

We should note that our proposal might be easily combined with other methods for weighting preferences such as the AHP method [17].

3.2 Moving into the Measure Phase

Suppose that before starting the selection procedure, a committed specification of the component (S_C) is defined as a reference to be used when comparing candidates. .

Once committed goals have been achieved, we may proceed with the selection process by applying different measures on COTS candidates. Our particular suite of measures of functional suitability of COTS components has been classified into two different groups: component-level measures and solution-level measures. The first group of measures aims at detecting incompatibilities on a particular component K, which is a candidate to be analysed. However, it could be the case that we need to incorporate more than one component to satisfy the functionality required by the abstract specification S_C. In this case, the second group of measures evaluates the functional suitability of all components that constitute the candidate solution.

At the component-level, we have defined the following measures [7]:

- "Compatible Functionality" (CF_C) as the amount of functional mappings provided by S_K and required by S_C in the scenario S.
- "Missed Functionality" (MF_C) as the amount of functional mappings required by S_C in the scenario S and not provided by S_K.
- "Added Functionality" (AF_C) as the amount of functional mappings not required by S_C in the scenario S and provided by S_K.
- "Component Contribution"(CC_F) as the percentage in which a component contributes to get the functionality required by S_C in the scenario S.

For the complete and formal definition of the functional suitability measure suite, we refer the reader to [7].

Now, let's calculate the functional suitability measures on K_2 for the E-payment example. Considering the functional mappings provided by K_2 ({AID → AOD; CID → COD; Taxes → Statistics}), the component-level measure results are as follows:

$CF_C(K_2) = 2;\ MF_C(K_2) = 0,\ AF_C(K_2) = 1;\ CC_F(K_2) = 1.$

These values indicate that the component K_2 is a candidate to be accepted for more evaluation; i.e. the component is completely functionally suitable. But there is one added function that could inject harmful side effects into the final composition. Besides, there are another types of analysis the component should be exposed before being eligible as a solution – such as analysis of non-functional properties [8], analysis of vendor viability [3], and so forth.

During the mapping, it can be the case that the semantics of K are not sufficient for S_C. In this situation, K is somehow lacking with respect to the behavioural semantics of C. The possibility is that K has partial behavioural compatibility with C. In this case, K either has incompatible or missing behaviour with respect to some portion of S_C. To overcome this, a semantic adapter must be specified (and built) such that, when composed with S_K, the adapter yields a component that is compatible with C [1].

Adaptation required by the components should also be quantified. For example, measurement might be defined at three levels:

(1) *Adaptability size measures* will be basically in terms of the amount of adaptability needed by a component-based solution. For example, one of the adaptability's measures might measure the amount of functions that are added when implementing the adapter of the component K (extended functionality); or contribution measures might measure the completeness of the adapters' functionality. For example, we could measure the percentage in which a component adapter contributes to get the extended functionality required by S_c in the scenario S.

(2) *Complexity of adaptation* would be measured in terms of interactions with target components that are identified to determine all potential mismatches. Then, for each mismatch, potential resolution techniques might be considered from the proposal by [10]. Hence, a weighting factor would be assigned to each connection to describe the difficulty with which that solution could be implemented.

(3) *Architectural adaptability* might define calculations for measures of changes that affect system's stability [6] in terms of architectural adaptability. Adaptability of an architecture can be traced back to the requirements of the software system for which the architecture was developed. For example, the POMSAA (Process-Oriented Metrics for Software Architecture Adaptability) framework [9], achieves the need of tracing by adopting the NFR framework that is a process-oriented qualitative framework for representing and reasoning about NFRs (non-functional requirements) [8]. In the NFR Framework, the three tasks for adaptation become softgoals to be achieved by a design for the

software system. An adaptable component of a software system should satisfice these softgoals for adaptation. For example, the underlying architecture for the E-Payment [EPAY] part of a E-commerce [ECOM] system, and the metrics that the POMSAA calculates for the adaptability of the architecture are given in Figure 5. The NFR Framework requires decomposition for the NFRs of interest. Figure 5 also gives the maximum and minimum values (the numbers inside the clouds) for the metrics of different softgoals. We refer the reader to [9] for a more detailed description on the computation of the metrics.

3.3 Moving into the Analyse Phase

In this phase a COTS candidate or a set of COTS candidates is selected from several alternatives, and decisions are made based on previous definitions and measurements. Basement for decisions might include to detect sub-domains affected by the recently incorporated COTS and identify dissatisfactions – functional dissatisfactions and accuracy dissatisfactions – according to the stability state established in Phase 1.

Let's introduce a suggested analysis as an example. The measures calculated in the previous phase, such as Architectural Adaptability, are used as a basement for supporting the different judgments. From Figure 5, we can see that Semantic Adaptation [ECOM, EPAY] has been split into three main hierarchies for analysis – Interface, Protocol, and Data. Each branch of the hierarchy is analysed in terms of complexity and size, using the measures calculated in the previous phase. The semantic adaptation has a metric of -2, which is the minimum score a Semantic Adaptation can get (from previously defined thresholds).

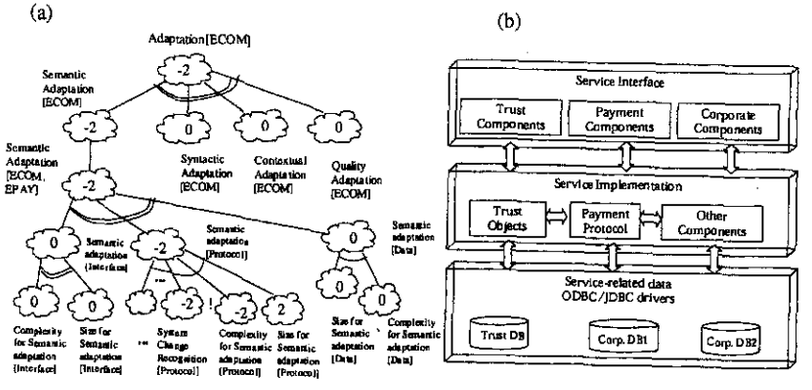

Fig. 5. An underlying architecture (b) and its architectural adaptability measures (a)

We suggest here integrating qualitative and quantitative measures for analysis considering that: (1) qualitative judgments are based on more precise values to be evaluated, which reduce ambiguity of the evaluation process; (2) quantitative values provide a more objective scale for comparison among alternatives; and (3) design decisions might be more formally specified.

3.4 Moving into the Improvement and Control Phases

As we have indicated, this phase is concerned with ensuring that the selected candidates can be integrated and supported within the required quality. For example, causes of low architectural adaptability should be determined, along with causes of functional dissatisfaction.

Evaluation of results can reveal problems that might arise during integration leading to prioritise solutions deciding on integrating the COTS component/s or building the required functionality from scratch. Different architecture's uses may necessitate changes to the same components or connections. In such a case, we should detect uses that affect the architecture keeping designs with the fewest use conflicts. Detecting conflict's causes is a subjective process, involving all stakeholders in the system. The results should reflect the relative importance of the quality factors that the uses manifest.

As an additional concern, improvement must be examined in the context of the system's requirements and design. It includes some aspects relating to verify that the system's requirements h ave been met by integrating the COTS solution.

A successful COTS selection procedure should communicate its practices and suggests possible corrective/adaptive actions to sustain its success. After evaluating results and detecting causes of architectural dissatisfaction, stakeholders must develop metrics and a control plan to ensure that the system continues to deliver the planned improvements. Although these activities do not differ from a traditional system development project or, indeed, from a classic Six Sigma project, they are important and should not be neglected.

4 Additional Remarks

Phases depicted in the previous section are a broad guideline to the Six Sigma-based selection process. We have chosen some activities to exemplify some of the main tasks of each phase; however several equally important activities were not included in our discussion. The following are only two examples of them:

- Once requirements are categorized and weighted, a process to obtain product and vendor information should be carried out. Sources of vendor names include research services, Internet searches, networking, industry group publications and contacts, advertisements in IT journals, and so forth. Also, product information provided by vendors should be assessed. For example, a recent work [4] presents a survey trying to evaluate how much of the information required to assess COTS components is actually available from vendors' information. The main goal of the survey is to ana-

lyse the current gap between the required and provided information, so refinement of quality models can be guided to reduce the gap, yielding in more realistic attributes and metrics.

- Scenarios have been widely used during design as a method to compare design alternatives and to express the particular instances of each quality attribute important to the customer of a system. Scenarios differ widely in breadth and scope, and its appropriate selection is not straightforward. Our use of scenarios is a brief description of some anticipated or desired use of a system. We emphasise the use of scenarios appropriate to all roles involving a system. The architect role is one widely considered but we also have roles for the system composer, the reuse architect, and others, depending on the domain. It is important when analysing a system that all roles relevant to that system be considered since design decisions may be made to accommodate any of the roles. The process of choosing scenarios for analysis forces designers to consider the future uses of, and changes to, the system. It also forces to consider non-functional properties that should be properly measured during the COTS selection process.

5 Conclusion

Six Sigma translates customer's needs into separate tasks and defines the optimal specification for each, depending on how the tasks interact. Based on what the process reveals, the steps that follow can have a powerful effect on the quality of products and customer services as well as the professional development of employees.

In this paper, we have introduced a procedure for selecting and integrating COTS components based on a Six Sigma approach. Its phases were depicted by introducing some techniques and measures, which would help in establishing a formal basis for applying the approach. Besides, the presence of specific measures allows stakeholders to make fact-based decisions improving the analysis of COTS candidates. We have shown, by using a simple example, how this can be done.

However, our proposal needs further validation. To do so, some empirically cases are currently carried out on the domain of E-payment systems. Among others, we are analysing the possibility of dealing with the large volumes of data generated and with low quality information provided by COTS component's suppliers.

References

1. Alexander R. and Blackburn M. Component Assessment Using Specification-Based Analysis and Testing. *Technical Report SPC-98095-CMC, Software Productivity Consortium* (1999).
2. Alves C. and Filkenstein A. Challenges in COTS-Decision Making: A Goal-Driven Requirements Engineering Perspective. In *Proceedings of the Fourteenth International Conference on Software Engineering and Knowledge Engineering*, SEKE'02 (2002).

3. Ballurio K., B. Scalzo, and L. Rose. Risk Reduction in COTS Software Selection with BASIS. In *Proceedings of the First International Conference on COTS-Based Software Systems*, Springer Verlag LNCS 2255, pages 31–43, (2002).
4. Bertoa M., Troya J., and Vallecillo A. A Survey on the Quality Information Provided by Software Component Vendors. In *Proceedings of the 7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE)*, (2003).
5. Cechich et al.: *Component-Based Software Quality: Methods and Techniques*, Springer-Verlag LNCS 2693 (2003)
6. Cechich A. and Piattini M.: Defining Stability for Component Integration Assessment. In *Proceedings of the 5th International Conference on Enterprise Information Systems*, ICEIS, pages 251-256, (2003)
7. Cechich A. and Piattini M. On the Measurement of COTS Functional Suitability. In *Proceedings of the 3rd International Conference on COTS-based Software Systems*, ICCBSS, (to appear February 2004)
8. Chung L., Nixon B., Yu E., and Mylopoulos J. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publisher, (2000).
9. Chung L. and N. Subramanian. Process-Oriented Metrics for Software Architecture Adaptability. In *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering (RE '01)*, pages 310–312, (2001).
10. Deline R. A Catalog of Techniques for Resolving Packaging Mismatch. In *Proceedings of the Symposium on Software Reusability*, Los Angeles, CA, (1999).
11. De Feo J. and Bar-El Z. Creating Strategic change more efficiently with a new Design for Six Sigma process, *Journal of Change Management*, vol. 3(1), pages 60-80 (2002)
12. Gack A. and Robinson K. Integrating Improvement Initiatives: Connecting Six Sigma for Software, CMMI, Personal Software Process and Team Software Process. *Software Quality Journal* 5 (4), 5-13
13. I* homepage, http://www.cs.toronto.edu/km/istar
14. ISO International Standard ISO/IEC 9126. ISO/IEC 9126 - Information technology - Software product evaluation - Quality characteristics and guidelines for their use, (2001).
15. Kaiya H., Horai H., and Saeki M. AGORA: Attributed Goal-Oriented Requirements Analysis Method. In *Proceedings of the IEEE International Conference on Requirements Engineering*, RE'02 (2002).
16. KAOS homepage, http://www.info.ucl.ac.be/research/projects/AVL/ReqEng.html
17. Saaty T.L. *The Analytic Hierarchy Process*. McGraw-Hill (1990).
18. Sedigh-Ali S., A. Ghafoor, and R. Paul. Metrics-Based Framework for Decision Making in COTS-Based Software Systems. In *Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering* (HASE'02), pages 243–244, (2002).
19. Tayntor C. *Six Sigma Software Development*. Auerbach Publications (2002)