Analysing the Tradeoffs Among Requirements, Architectures and COTS Components

Carina Alves, João Bosco Pinto Filho, Jaelson Castro
Centro de Informática, Universidade Federal de Pernambuco
E-mail: {cfa, jbapf, jbc}@cin.ufpe.br

Abstract. The development of software systems from already built COTS components has been motivated by the prospect of reduced cost and development time. However, developing COTS-based systems introduces new challenges and risks different from building systems from scratch. In particular, this new paradigm requires simultaneous tradeoffs among user requirements, COTS products and system architecture. In this paper, we describe a set of guidelines for handling risks and uncertainties associated with such tradeoffs.

Keywords: COTS-Based Development, Requirements Engineering and Software Architecture.

1. Introduction

As the size and complexity of software systems grows, increases the interest in developing systems based on reusable components, known in literature as COTS (Commercial-Off-The-Shelf). The potential benefits of this new technology are reduced costs and shorter development time 0. The nature of COTS suggests that the model of component-based software development should be different from the conventional one. As a result, a significant shift has been observed from the development-centric toward a procurement-centric approach 0. This approach focuses on building large software systems by integrating previously existing software components available in the market. In general, the COTS-based development (CBD) lifecycle consists of the following phases: identification, evaluation, selection, integration and update of components 0.

According to Wallnau 0, COTS-based systems comprise a spectrum, ranging from COTS-solution systems at one extreme, to COTS-intensive systems at the other extreme. A COTS-solution system refers to an off-the-shelf solution that one substantial product is tailored to provide a solution; examples include data management, financial management, or manufacturing execution. On the other hand, COTS-intensive systems are far more complex; this kind of system integrates many products from different vendors to provide the system functionality. In such systems an important issue is the way components are interconnected and how they communicate through the software architecture. Therefore, it is necessary to define an architecture that allows the integration of the various components.

It looks very promising to use COTS components in order to improve productivity and quality of software systems development. Although, the use of COTS software introduces new problems and risks different from building a system from scratch 0.

Many of these problems are introduced because of the black-box nature of COTS components. In particular, a careful COTS evaluation process is crucial for any effective COTS-based system. During this stage, components capabilities must be assessed against the evaluation criteria, which mainly include stakeholders' requirements, architecture constraints and non-technical aspects (such as vendor guaranties, legal issues) 0. The adaptation process addresses potential sources of conflict among components that cannot effectively integrate into system architecture. We argue that building systems from COTS components might be a compromise among all these concerns.

This paper is organized as follows. Section 2 introduces the basic concepts of COTS-based development, requirements engineering and software architecture. Section 3 discusses some challenges developers have to deal when building COTS-based systems. Section 4 presents some guidelines to deal with common risks associated with COTS-based development. Finally, Section 5 presents the conclusions of this work.

2. Background

Prior to present the proposed approach to systematically deal with critical issues related to building COTS-based systems, it is necessary to introduce some basic related concepts. These aspects are presented in the sequel.

1.1 Component-Based Development

Currently there is no general agreement over what constitutes a component or a COTS. These two concepts are closely related and are used in a range of domains. Although, we aim to make a distinction between them. Brown 0 defines a component as:

A non-trivial, independent and replaceable part of a system that fulfils a clear function in the context of a well-defined architecture;

A run-time, dynamically bindable software of one or more programs managed as a unit and accessed through documented interfaces that can be discovered at run-time;

Current research in component-based software engineering mainly focuses on component infrastructure capabilities and middleware solutions for connecting components. Some technologies have been developed to provide a standardization for component infrastructure, such as: CORBA (Common Object Request Broker Architecture)[1], Sun's Java Beans and Enterprise Java Beans[2], COM (Common Object Model)[3]. Each of these approaches relies on underlying services to provide the communication and coordination necessary to construct component-based applications.

[1] http://www.corba.org

[2] http://java.sun.com/products/ejb/

[3] http://www.microsoft.com/com/

Broadly speaking, the term COTS refers to things that one can buy, ready-made from some vendors. Vidger 0 considers that a COTS software component is software that is acquired from a commercial source and is integrated into a working system. According to Oberndorf 0, "COTS are products that are sold, leased or licensed to the general public; that is usually available without source code; that is supported and evolved by the vendor who returns the intellectual property rights".

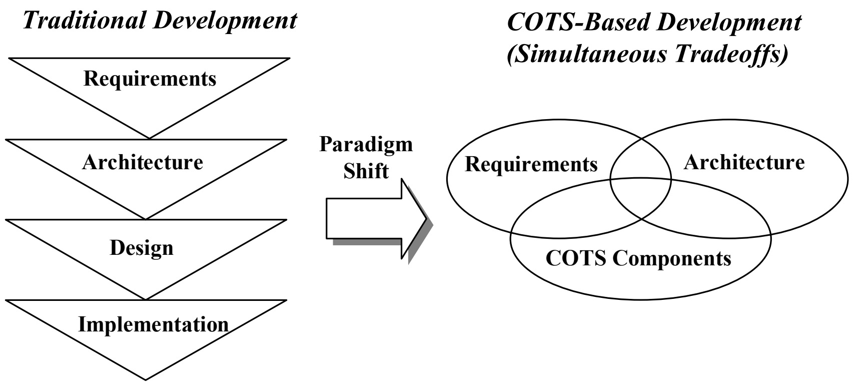

Figure 1. Paradigm Shift

On the opposite of traditional software development, that in general follows a pre-established sequence of activities, the COTS-based paradigm is based on a constant, simultaneous and iterative tradeoff among user requirements, software architecture and COTS components, see figure 1. It means that such systems might be a commitment of these aspects. The notion of lifecycle activities is also affected by the paradigm shift, since some activities such as product evaluation, wrapping, bridging, differs from traditional development activities. Yet, they are extremely important for COTS-based systems.

1.2 Requirements Engineering

According to Zave 0, "requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software, and to their evolution over time and across software." It has been argued that requirements engineering is increasingly recognized as a critically important activity in any systems engineering process 0 0.

In general, requirements engineering encompasses the processes of acquiring, modeling and managing stakeholders requirements. At the beginning of this process, it is necessary to gather information about the organization structure, understand the problem to be solved and interpret stakeholders needs and constraints. Then, requirements are modeled in order to establish an agreed set of requirements that are complete and consistent. In most system development, stakeholders have conflicting requirements since they have different needs and priorities. Requirements negotiation is the process of discussing the conflicts in requirements and find some compromises which all of the stakeholders can accept 0. In COTS-based development systems, all these requirements engineering activities are also performed, especially during the phases of evaluation and selection of COTS products. Although, there are some differences between the requirements process for traditional systems and for COTS-based systems. Basically the differences are because of the following factors:

Requirements instability are more severe because of COTS market volatility;

Requirements specification may be affected by COTS capabilities, i.e. new requirements can be gathered from products evaluation;

COTS products may change due to the releasing of a new version, so requirements may also change;

COTS products usually have more functionalities than a particular customer needs;

COTS impose additional constraints on the system and these requirements are usually unforeseen;

Some stakeholders requirements cannot be satisfied by any available product in the market;

We argue that all these issues must be taken into account in order to obtain an effective COTS-based development. In particular, it can only be achieved by an iterative process of requirements engineering, COTS selection and architecture design 0. These compromises also requires a careful consideration of non-functional requirements because they address important issues of quality for software systems. Non-functional requirements are usually known as "ilities" attributes, such as: portability, security, maintainability, usability, stability, etc. In general, non-functional requirements have a global nature, which means that the satisfaction of a single NFR may affect several design components.

1.3 Software Architecture

Software architecture is the highest abstract description of a software design, which is defined at the initial stages of the software development. It is commonly described in terms of three basic abstractions: components, connectors and configurations. Components represent a wide range of different elements, from a single client to a database, and have an interface used to communicate with the external environment. Connectors represent communication elements between components. The configuration describes how components and connectors are wired. A traditional view of software architecture is shown in figure 2.

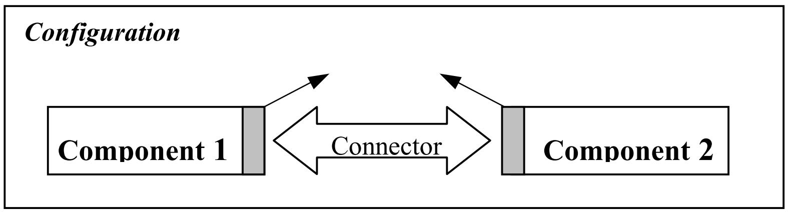

Figure 2. Software Architecture

Garlan 0 describes software architecture as the level of design which deals with structural issues such as gross organization and global control structure; protocols for communication, synchronization, and data access; assignment of functionality to design elements; physical distribution; composition of design elements; and selection among design alternatives. The main purposes of software architecture are to define the major components of a system, how the components interface with each other, and the interactions between components to provide the system services 0. In general, COTS-based development do not follow the system's architectural requirements in many different ways. For instance, conflicting COTS must be adapted in order to effectively interconnect with the rest of the system. These issues can be dealt with the help of software integration elements like the ones described below:

Wrappers are pieces of code custom built in order to isolate the unwanted functionalities of the COTS component from other components of the system. They provide the only allowed access method to the wrapped component.

Glue is the code used to provide the functionality to integrate different components. It deals with control flow, component bridge and exception handling.

Component tailoring refers to the ability for system architects to improve the component's functionality. The tailoring is done by adding some elements to the component in order to include functionalities that are not provided by the vendor.

Developers should be aware that many of the problems raised by the use of COTS software components can be addressed within a more formal and defined process, by carefully defining the architecture and design of the system. This systematic process results in a more reliable software that can evolve over time.

3. The challenges of Building Systems from COTS Components

Although building systems from COTS components offers the opportunity to reduce the development time and cost of software systems, there are still many problems that need to overcome. The development of systems based on the integration of COTS possesses a particular set of business, technical and non-technical challenges. Following we list some challenges that are often experienced:

The black box nature of COTS components. Customers often do not have access to source code and cannot modify the code to change the properties of the component. It also means that just black box tests can be performed during COTS assessment. This type of testing is also seen during acceptance test and is considered to be the foundation of validation testing which confirms that the software actually performs the required functionalities.

The selection of COTS components is a non-trivial task. There is a lack of well-defined selection processes, most organizations are under pressure to develop systems faster and cheaper and perform COTS selection in an ad-hoc manner. Moreover, the evaluation criteria are sometimes subjective and ambiguous and do not provide an effective description of customer needs.

COTS specifications are usually incomplete and superficial. Most products documentation that is available consists of user manuals and advertising material and does not provide an effective description about COTS capabilities and constraints. For example, it does not describe the behavior of the system in response to abnormal input, which is related to quality aspects of reliability and stability.

Frequent updates of commercial products. The components marketplace changes rapidly and new COTS versions are released with a lot of different functionalities. This aspect has a significant influence on the selection process, since a new release of the product may have a feature that is not available in the product that is currently being evaluated.

Requirements evolve during development process. Some requirements for COTS-based systems will only be known after initial evaluation and at the moment the system is being integrated. In addition, some products impose additional architectural requirements, such as interoperability and reliability constrains that are usually unforeseen during early evaluation.

Including a new COTS means incorporating additional constraints. These constraints can affect the system's overall architecture, functionality, and non-functional aspects. For instance, a new component can have a negative contribution to the system's overall quality attributes. These conflicts result in tradeoffs that must be analyzed and solved.

Adaptation of COTS is usually required in order to hide unwanted functionality. COTS vendors often overload their systems with a large amount of functionality that take no part on user's requirements. Since this inconvenience cannot be solved modifying black box COTS components, system's architects must provide a way of masking the unwanted functionality so that it is inaccessible to the end-users and system programmers.

Set of COTS components may be mismatched. Integration of COTS components usually brings multiple inconsistent architectures frameworks within a single system. This problem has been identified as an architectural mismatch; Garlan 0 identified four categories of mismatches between conflicting components, they are assumptions about: the nature of the components, the nature of the connectors, the global architectural structure, and the construction process. These architectural mismatches have been identified as a fundamental obstacle to component-based development.

Customers have little or no control over the evolution of the product. The COTS vendors have several customers, so it is almost impossible to attend specific needs of a single customer. They have minimal influence over how the component evolves and the capabilities added to futures updates are mainly introduced by market strategies and technology trends.

Vendor dependency over the maintenance of the system. Usage of COTS means taking risks that future system extensions might be limited due to dependence on the support of COTS vendors for the new required capabilities. In addition, there is no guarantee of long-term support from vendors or compatibility between versions of newer selected COTS with those originally integrated in the system. If the vendor stops supporting the product or goes out of business, customers are forced to change to a different COTS. This results in modifications on the system architecture.

Future COTS replacements influence architecture design and quality aspects. Replacement of a particular unsuitable COTS product can result in several inconsistencies and expensive redesign of the system. The following situations are examples of such inconsistencies: a new component may not be ported to the platforms the user requires, it may interfere with the operation of some required functionality, or it may interact in some unexpected way with other components.

In order to effectively address the potential problems identified above, a systematic and detailed approach is needed to assist the development process of COTS-based systems. Next section provides a discussion about the management of risks that may occur during the development of such systems.

4. Guidelines for Managing Risks of COTS-Based Systems Development

In this section, we propose a set of guidelines that properly deals with the risks associated with COTS-based development. In particular, some shortfalls in requirements and architecture are identified. The groundwork for this paper was our previous experience when developing and using the CRE (COTS-Based Requirements Engineering) method, which is detailed presented in 00. The CRE Method was developed to facilitate a systematic, repeatable and requirements-driven COTS software selection process.

The method has four iterative phases: *Identification, Description, Evaluation*, and *Acceptance*. In particular, this sequence is not rigid, since the CRE is based on the iterative process of requirements elicitation/specification and COTS selection. The main contribution of the CRE method over other selection methods 000 is a novel approach that effectively treats non-functional requirements during the selection of COTS products. In addition, we performed some real case studies in order to validate the method. Some studied domains were: selection of IDE (Integrated Development Environment) Java and selection of medical packages. These case studies have provided some key insights into the situations that can bring risks when building COTS-based systems. We argue that these risks must be analysed and managed to fully exploit the advantages of COTS-based development. According to Charette 0, risk analysis process encompasses the following activities:

The identification of risks;

The estimation of risks (often in terms of severity);

The evaluation of risks.

In addition, risk management is concerned with making decisions in which risks are continuously identified and analyzed for relative importance. Moreover, during this process risks must be mitigated, tracked, and controlled. Following we present some guidelines to deal with risks identified during the COTS-based development lifecycle.

Situation	Risks	Guideline
Lack of well defined selection process	▪ (1) A poorly scoped domain, in which the domain analysis process can be susceptible to a loss of focus.	▪ (1) Acquire information about system domain from specialists and key stakeholders, using interviews and questionnaires techniques.
	▪ (2) Lessons from previous experiences are not learnt.	▪ (2) Define a systematic selection process, using strategic planning and appropriate selection tools and methods, like the CRE method 0 .
	▪ (3) Organizations are under pressure to perform evaluation process faster and without allocating qualified individuals. This situation can result in a ineffective COTS selection.	▪ (3) Designate an evaluation team with specialists in selection processes, domain experts, requirements engineers.
	▪ (4) Individuals within the organization are opposed to COTS-based development (CBD).	▪ (4) Conduct educational sessions, explaining the benefits of CBD and describing successful industrial cases using this development approach.
Ineffective evaluation criteria	▪ (5) The lack of consideration of non-functional requirements increases the risks of COTS failure and the costs of the final system.	▪ (5) Use approaches that specify non-functional requirements, such as the NFR framework 0. The analysis of non-functional requirements helps the discrimination between competing products and improves the decision making process to choose a suitable set of COTS that will be integrated to build the final system.
	▪ (6) If requirements are too specific and inflexible it might be impossible to find an appropriate solution that meet the simultaneous tradeoffs among requirements, architecture and available COTS components.	▪ (6) Initially, describe requirements broadly in order to find potential COTS candidates in the market; then refine requirements statements, specially the non-functional ones. Use techniques, like WSM and AHP 0 to assist the decision making process that must take a careful balancing of requirements and architecture constraints.

Situation	Risks	Guideline
	▪ (7) Developers might have the misconception that the cost of COTS-based systems are just related to acquisition cost.	▪ (7) There are a range of associated costs when developing COTS-based systems such as training, adaptation and maintenance. Use COCOTS model 0 to perform cost estimation.
COTS components are developed as generic products	▪ (8) COTS are developed to satisfy an entire market instead of meet requirements of particular customers. In this way, COTS products usually provides more functionalities than a customer needs.	▪ (8) The evaluation process must be context-driven, i.e. COTS assessments are conducted within the scope of the system to be built and the organization domain. ▪ (9) Perform techniques to mask out currently unneeded capabilities, such as wrapping and scripting.
	▪ (9) There is no guaranty that a COTS product can meet all stated requirements.	▪ (10) Perform prioritization of stakeholders requirements and try to ensure that at least critical requirement are met. ▪ (11) Additional components need to be developed to meet the shortfalls, but may be specific to a particular domain.
Lack of detailed COTS components descriptions	▪ (10) Selecting unsuited COTS components that present integration problems due to low conformance with quality requirements.	▪ (12) Conduct demonstration sessions to explore products capabilities in a more realistic environment. During these sessions, it is important to evaluate product's compatibility, integrability and interoperability with system architecture.
	▪ (11) COTS are described using different vocabulary leading to difficulties to make comparisons between them.	▪ (13) Use documenting standards to describe COTS capabilities. In 0, a proposal is made to describing COTS using XML schemas templates.

Situation	Risks	Guideline
Architecture can significantly affect the whole COTS-based system	▪ (12) The specified architecture is inflexible and difficult to adapt for particular circumstances.	▪ (14) Avoid early commitment to an architecture. Architectural decisions should be made concurrently with COTS evaluation, in such a way that it can be changed as far as requirements statements are refined. It is worth noting that requirements may also be changed in order to meet architectural constraints.
	▪ (13) Inappropriate properties are presented in the integrated system and are being preserved during maintenance.	▪ (15) Perform verification tests to determine the level of conformance of the COTS integrated with the architectural description.
	▪ (14) COTS components interact directly in a way that updates on one of them may affect the others as well as the global system.	▪ (16) COTS should not interact directly to each other. Wrappers and glues should evolve with component's updates.
COTS components evolve rapidly	▪ (15) Customers have no control over COTS product's evolution.	▪ (17) Use flexible architectures facilitating modification and update of COTS. ▪ (18) Carefully evaluate COTS vendors' track records with respect to predictability of product evolution and establish a pro-active system release strategy, synchronizing COTS upgrades with system releases.
	▪ (16) Uncritically accepting COTS vendors' statements about product capabilities and support.	▪ (20) Establish strategic partnerships or other incentives for COTS vendors to provide support and negotiate critical vendor support agreements.

Table 1. Guidelines to handle risks of COTS-based development

We believe the presented categorization covers a wide range of critical situations when developing COTS-based systems. Moreover, it provides a practical strategy to manage and minimize some potential risks associated with these identified situations.

To illustrate how the guidelines presented above can be used in a real case, consider the selection of medical package for a clinic management. In this case, the guidelines supported the processes of COTS assessment and integration into the organization domain. In order to avoid risk 1, we identified stakeholders' main goals using interviews sessions. During this initial process we had found core requirements to be considered during the localization of COTS products available in the market.

According to guideline 3, the evaluation of packages candidates was conducted by a team with domain experts and requirements engineers. In particular, guideline 4 was not considered because the clinic staff agreed with the use of COTS. As can be observed, the guidelines are helpful to conduct the process of COTS-based development. It is worth noting that developers must decide which guidelines are suitable to use in each situation.

5. Conclusions

Developing systems using COTS products has a number of potential benefits to organizations. However, underestimating the technical risks associated with selecting and integrating these components can result in poor systems that don't meet stakeholders requirements.

This paper has discussed some critical issues related to the continuous tradeoffs among user requirements, architectural descriptions and COTS components. Main contributions of this paper include the proposal of a set guidelines that deals with some potential risks that might occur during COTS-based development. Finally, we believe that the presented strategy improves the chances of success of COTS-based systems.

References

[1] Oberndorf, P. and Brownsworld, L. "Are You Ready for COTS?" *Software Institute Engineering*. August 1997.

[2] Tran, V and Liu, D. "A Procurement-centric Model for Engineering Component-based Software Systems". *Proceedings of the Fifth International Symposium on Assessment of Software Tool*. June 1997.0

[3] Oberndorf, P. "Facilitating Component-Based Software Engineering: COTS and Open Systems". *Proceedings of the Fifth International Symposium on Assessment of Software Tools - SAST'97*. June 1997.

[4] Wallnau, K.C. Carney, D. e Pollak, B. "How COTS Software Affects the Design of COTS-Intensive Systems". *Software Engineering Institute*, Carnegie Mellon University, USA. Jun. 1998.

[5] Vidger, M. R and Gentleman, W. M. "COTS Software Integration State of the Art". *National Research Council of Canada*, Institute for Information Technology technical report, January 1996.

[6] Kontio, J. Caldiera, G. and Basili, V. "Defining Factors, Goals and Criteria for Reusable Component Evaluation". *CASCON'96*. November 1996.

[7] Brown, A. W. and Wallnau, K. C. "Engineering of Component-Based Systems, Component-Based Software Engineering". *Software Engineering Institute*, IEEE Computer Society Press, 1996.

[8] Zave, P. "Classification of Research Efforts in Requirements Engineering". ACM Surveys, 29(4), 1997.

[9] Kotonya, G. and Sommerville, I. *Requirements Engineering – Processes and Techniques*. John Willy & Sons, 1997.

[10] Nuseibeh, B. e Easterbrook, S. "Requirements Engineering: A Roadmap". *Proceedings of the 22nd International Conference on Software Engineering*. Limerick, Ireland. Jun. 2000

[11] Ncube, C. Maiden, N. "COTS Software Selection: The Need to make Tradeoffs between System Requirements, Architectures and COTS/Components". *Workshop Ensuring Successful COTS Development*. Los Angels, May 2000.

[12] Shaw, M. and Garlan, D. *An Introduction to Software Architecture*. Carnegie Mellon University, 1994

[13] Vigder, M and Dean, J. An Architectural Approach to Building Systems from COTS Software Components.NRC40221FALTA

[14] Garlan, D. Allen, R. and Ockerbloom, J. "Architectural Mismatch (Why it is hard to build systems out of existing parts)", *Proceedings of the 17th International Conference on Software Engineering*, April, 1995.

[15] Alves, C. Castro, J. "Um Método Baseado em Requisitos para Seleção de COTS". *Fourth Workshop Iberoamerican on Software Engineering and Software Environment (IDEAS'01)* San Jose, Costa Rica, April 2001.

[16] Alves, C. Seleção de produtos de Software "Utilizando uma Abordagem Baseada em Engenharia de Requisitos". *MSc. Thesis*, Universidade Federal de Pernambuco, Centro de Informática. March 2001.

[17] Kontio, J. "A COTS Selection Method and Experiences of Its Use". *Proceedings of the 20th Annual Software Engineering Workshop*, Maryland, November 1995.

[18] Kunda, D. and Brooks, L. "Applying Social-Technical Approach for COTS Selection". *Proceedings of the 4th UKAIS Conference*, University of York, April 1999.

[19] Ncube, C and Maiden, N. "PORE: Procurement-Oriented Requirements Engineering Method for the Component-Based Systems Engineering Development Paradigm". *International Workshop on Component-Based Software Engineering*, May 1999.

[20] Charette, R. N. Software Engineering Risk Analysis and Management, McGraw-Hill, New York, 1989.

[21] Chung, L. Nixon, B. Yu, E. and Mylopoulos, J. "Non-Functional Requirements in Software Engineering". *Kluwer Academic Publisher*, 2000.

Saaty, T. *The Analytic Hierarchy Process*. New York: McGraw-Hill, 1990.

[22] Boehm, B. Abts, C. Bailey, E. "COCOTS Software Integration Cost Model: an Overview". *Proceedings of the California Software Symposium*. October 1998.

[23] Iribarne, L. Troya, M. and Vallecillo, A. "Trading for COTS Components in Open Environments". *Proceedings of 27th Euromicro. Workshop on Component Based Software Engineering*. Warsaw, Poland. September, 2001.