Agile Estimation Techniques and Innovative Approaches to Software Process Improvement

Ricardo Colomo-Palacios
Universidad Carlos III de Madrid, Spain

Jose Antonio Calvo-Manzano Villalón
Universidad Politécnica de Madrid, Spain

Antonio de Amescua Seco
Universidad Carlos III de Madrid, Spain

Tomás San Feliu Gilabert
Universidad Politécnica de Madrid, Spain

A volume in the Advances in Systems Analysis, Software Engineering, and High Performance Computing (ASASEHPC) Book Series

Information Science REFERENCE
An Imprint of IGI Global

Managing Director:	Lindsay Johnston
Production Editor	Jennifer Yoder
Development Editor:	Allyson Gard
Acquisitions Editor:	Kayla Wolfe
Typesetter:	John Crodian
Cover Design:	Jason Mull

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Agile estimation techniques and innovative approaches to software process improvement / Ricardo Colomo-Palacios, Jose Antonio Calvo-Manzano Villalon, Antonio de Amescua Seco and Tomas San Feliu Gilabert, editors.
pages cm
Includes bibliographical references and index. ISBN 978-1-4666-5182-1 (hardcover) -- ISBN 978-1-4666-5183-8 (ebook) -- ISBN 978-1-4666-5185-2 (print & perpetual access) 1. Agile software development. 2. Computer software--Development. I. Colomo-Palacios, Ricardo, 1973- editor of compilation.
QA76.76.D47A3823 2014

005.1--dc23
2013044979

This book is published in the IGI Global book series Advances in Systems Analysis, Software Engineering, and High Performance Computing (ASASEHPC) (ISSN: 2327-3453; eISSN: 2327-3461)

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 12

On Software Architecture Processes and their Use in Practice

Perla Velasco-Elizondo
Autonomous University of Zacatecas, Mexico

Humberto Cervantes
Autonomous Metropolitan University, Mexico

ABSTRACT

Software architecture is a very important software artifact, as it describes a system's high-level structure and provides the basis for its development. Software architecture development is not a trivial task; to this end, a number of methods have been proposed to try to systematize their related processes to ensure predictability, repeatability, and high quality. In this chapter, the authors review some of these methods, discuss some specific problems that they believe complicate their adoption, and present one practical experience where the problems are addressed successfully.

1. INTRODUCTION

In recent years, software architecture has begun to permeate mainstream software development and, according to Shaw and Clements (Shaw & Clements, 2006), since the year 2000, architecture has entered a "popularization" period characterized by aspects such as increased attention to the role of the software architect and the introduction of software architecture processes into organizations. As part of this trend, a number of methods have appeared to try to systematize these processes to ensure predictability, repeatability, and high quality outcomes.

The software architecture of a software system is the structure (or structures) of this system, which comprises software elements, the externally visible properties of those elements, and the relationships among them (Clements et al., 2010). In this chapter, by software architecture development we refer to the activities that are typically performed early in a software development project, which contribute to creating the different structures that shape the architecture. Despite the

DOI: 10.4018/978-1-4666-5182-1.ch012

availability of methods to support the processes related to software architecture development, we consider that there is a set of specific problems that complicate the adoption of such methods in practice. A summary of these problems can be stated as follows:

1. **Selection of methods for the software architecture lifecycle:** Ideally, software architecture development should be carried out within the context of a software architecture lifecycle, which imposes a structure on the activities for developing it. Existing software architecture development methods typically focus only on a particular phase of the lifecycle and do not cover it completely. Thus, an appropriate combination of methods to cover the complete lifecycle must be chosen.
2. **Heterogeneity of the existing methods**: Many existing software architecture development methods have been defined by different authors "in isolation," i.e. independently of methods used in other lifecycle phases. This results in having them defined in terms of different activities, work products and terminology. This heterogeneity requires that, once a particular combination of methods is chosen, they must often be analyzed and modified to avoid mismatches, omissions or repetitions.
3. **No consideration of the software development process:** Software architecture development methods are typically defined independent of a particular software development process. Therefore, the introduction of architectural development methods into an organization often demands adapting both the organization development process and the architectural development methods to fit properly (Kazman, Nord, & Klein, 2003).
4. **Architectural design methods are decoupled from everyday practice:** To support the design of an architecture many methods use abstract concepts such as tactics and patterns. These concepts are frequently not the ones that software architects use the most in their day-to-day activities, as many architects tend to favor the selection of technologies such as software frameworks during design. Thus, it is necessary to find ways to include commonly used concepts into architectural design methods (Cervantes, Velasco-Elizondo, & Kazman, 2013).
5. **Difficulty of organizational deployment:** The introduction of architectural methods into an organization often involves costs related to process change, human resources training and technology investment. To promote the successful adoption of software architecture development methods in an organization it is necessary to follow a systematic deployment process.

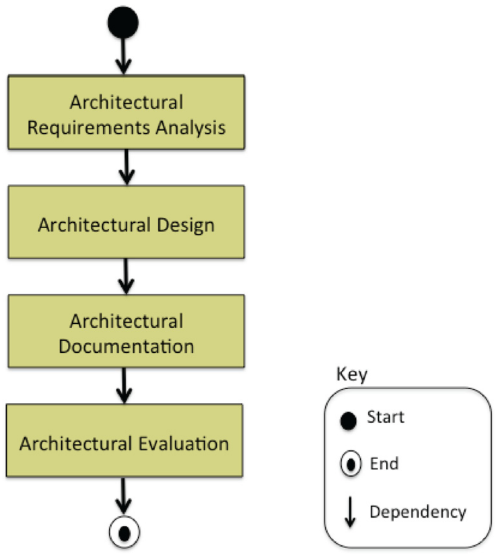
Based on the problems listed above, in this chapter we propose some actions to address them and describe the observed benefits when implementing them in an industrial setting, specifically, in a large software development company in Mexico City, currently rated at CMMI-DEV level 5, which develops custom software for government and private customers.

This chapter is organized as follows. In Section 2, we introduce the notion of software architecture lifecycle and, within this context, review some well-known processes and methods to support it. Next, we discuss in more detail in section 3 the set of problems that we consider have complicated the adoption of these methods in practice. In Section 4, we describe a specific instance where these problems were addressed in practice. Section 5 presents a discussion. Finally, in the last section, we draw some conclusions and describe paths of future work.

2. REVIEW OF SOFTWARE ARCHITECTURE PROCESSES AND METHODS

Before starting the review of software architecture processes and methods, it is important to introduce the notion of an architecture development lifecycle. Ideally, software architecture development should be carried out within the context of a software architecture lifecycle, which imposes a structure on the activities for developing it. The architecture development lifecycle can be seen as a general model that comprises all the activities and work products required to develop a software architecture. The software architecture development lifecycle is composed of a set of phases depicted in Figure 1: architectural requirements analysis, architectural design, architectural documentation, and architectural evaluation. It should also be noted that, although these phases are not necessarily performed sequentially, there is a sequential information dependency between them, i.e. the design phase depends on the availability of the information generated during the requirements analysis phase (Hofmeister et al., 2007).

Figure 1. Phases of the software architecture development lifecycle



Each one of the phases of the software architecture development lifecycle is supported by a general process; to this end a number of methods have appeared to try to systematize these processes to ensure predictability, repeatability, and high quality outcomes. In the following sections, we describe the focus of each one of these processes and review some well-known methods to support them.

2.1 Architectural Requirement Analysis Process and Methods

The architectural requirements analysis process involves the activities of eliciting, analyzing, specifying and prioritizing architectural requirements so that they can later be used to drive the design of the architecture. A representative output of this process is the architectural drivers, which represent the main functional and non-functional requirements, where the latter include quality attributes requirements and constraints.

The Quality Attribute Workshop (QAW) (Barbacci et al., 2003) is a method to elicit, analyze, specify and prioritize quality attributes requirements, e.g. performance, availability, security or testability. In the QAW quality attributes requirements are specified as scenarios, which are textual descriptions of how the system responds, in a measurable way, to some particular stimulus. For example, "…when a door sensor detects an object in the door's path, the door motion is stopped in less than one millisecond" is an excerpt of a performance scenario. Scenarios are described according to a suggested 6-part template with the active participation of the main system stakeholders, who propose and prioritize them. The results of the QAW include a list of quality attributes requirements as well as a prioritized and refined set of scenarios.

Another relevant method in this context is the Architecture Centric Design Method (ACDM) (Lattanze, 2009). ACDM considers a set of eight sequential stages; most of them focus on architectural design and evaluation. Stages 1 and 2 of ACDM discover architectural drivers and establish project scope, focus on eliciting, analyzing and specifying architectural requirements. As in the QAW, these stages require the active participation of the main system stakeholders and scenarios are utilized to specify the quality attribute requirements of the system. Other types of architectural drivers are also addressed in stages 1 and 2 of ACDM, i.e. functional requirements and constraints.

Within the context of the Rational Unified Process (RUP) (Kroll, Kruchten, & Booch, 2003) (Jacobson, Booch, & Rumbaugh, 1999), FURPS+ (Eeles, 2012) is a model defined to support the elaboration of a supplementary (requirements) specification. The supplementary specification contains the requirements that are not captured in the use case model and is generated as part of the Requirements discipline in the Elaboration Phase of the RUP. FURPS+ stands for Functionality, Usability, Reliability, Performance and Supportability. The "+" in the acronym denotes other important development concerns, such as constraints, that must be taken into account. In contrast to the methods described earlier, FURPS+ does not prescribe a particular way of analyzing, specifying and prioritizing quality attributes requirements.

2.2 Architectural Design Process and Methods

Within the context of the architectural development lifecycle, the process supporting the architectural design phase focuses on identifying and selecting the different structures that compose the architecture and that will allow the drivers identified in the architectural requirements analysis to be satisfied. Next, we describe some methods to support the activities of this process.

The Attribute Driven Design (ADD) (Bachmann et al., 2000) is a method to design a software architecture based on the selection of patterns and tactics. In software engineering, patterns are understood as conceptual solutions to recurring problems in specific design contexts. Patterns have names associated with them that facilitate their identification e.g. the layers pattern. Although it is difficult to classify patterns, it is generally accepted that architectural patterns (Buschmann, Henney, & Schmidt, 2007) and design patterns (Gamma et al., 1995) exist. On the other hand, architectural tactics are understood as design decisions that influence the control of a quality attribute response (Bass, Clements, & Kazman, 2012), e.g. the use of a redundancy tactic promotes the degree of availability and the use of an authentication tactic promotes the degree of security. ADD assumes the existence of a set of quality attribute scenarios and follows a top-down recursive decomposition-based approach where, at each iteration, tactics and patterns are selected and applied to satisfy a subset of the system's quality attribute scenarios. In the first iteration the element to decompose is generally the entire system. Subsequent interactions focus on the application of tactics and patterns to the resulting design structures from previous iterations. The architectural design is considered complete when all the scenarios have been satisfied.

As introduced before, ACDM is an eight-stage method that mostly concerns architectural design and evaluation. Once the architectural drivers have been identified in stages 1 and 2 of ACDM, stage 3 focuses on the creation of a design for the system architecture as well its documentation. Thus, architectural design and architectural documentation are not separate stages in ACDM. Although this method does not promote a particular design approach, compared to ADD, it suggests a set of techniques to create the architectural design.

RUP also supports the architectural design activity via specific workflows in the Analysis and Design discipline of the Elaboration Phase. In these workflows, the focus is on creating an initial architecture for the system and completing it by analyzing the system behavior. A similar approach is adopted in OpenUP (OpenUP, 2012), which is a lightweight open-source instance of RUP.

2.3 Architectural Documentation Process and Methods

The architectural documentation process involves creating the documents that describe the different structures that compose the architecture for the purpose of communicating it efficiently to the different system stakeholders. An important output of this process is a set of architectural views, which represent the system's structures, their composing elements and the relationships among them. Because all the details of a software architecture are hard to represent in a single view, documenting the architecture involves creating a set of relevant views which can be classified into different types: module views, which show structures where the elements are implementation units; component-and-connector views, which show how the elements in the structures behave at run time; and allocation views, which show how the elements in the structures are allocated to physical resources like the hardware, file systems, and people (Clements et al., 2010) .

The 4+1 view model (Kruchten, 1995) is an architectural documentation method adopted by RUP. This method considers the generation of five interrelated views: the Logical View, the Process View, the Physical and the Development View. The fifth view corresponds to the Use Case view around which the other views revolve. The views are meant to be documented iteratively based on existing information in previously developed artifacts such as use cases and the supplementary specifications. In the 4+1 view model, the syntax suggested for documenting the architecture is UML.

Views and Beyond (V&B) (Clements et al., 2010) is another method to document architectural views. The V&B approach defines two main stages for architectural documentation: (1) selecting the views that are worth documenting and (2) documenting them using a specific template. The template includes elements such as a primary representation, an architectural elements catalog, a context diagram, a variability guide and an architecture background. Multiple related views can be grouped in a view package that includes the views and information to relate these views to each other.

Another method to support the architectural documentation process is Viewpoints and Perspectives (Rozanski & Woods, 2005). A viewpoint defines a view in which content and conventions for constructing it are standardized. A perspective is a collection of guidelines to achieve a particular quality property relevant to a number of architectural views. The method provides a framework for choosing the relevant views based on the structures that are inherent in the software architecture. Six viewpoints (i.e. functional, information, concurrency, development, deployment and operational) and seven perspectives (e.g. security, performance, availability, usability, accessibility, location and regulation) are defined. Both viewpoints and perspectives are described in detail in a set of documents, which include information such as definition, concerns addressed, applicability, related stakeholders, activities, common problems and pitfalls, and a set of checklists to guide the architecture definition.

In previous sections we introduced the ACDM method and mentioned that architectural documentation is part of stage 2 that focuses on architectural design. Thus, the output of stage 2 comprises the initial, or the refined, architectural design and the associated documentation artifacts. ACDM considers static, dynamic and physical views, which are analogous to the module, component-and-connector and allocation views mentioned before, and suggests organizing them according to a specific template. The ACDM does not emphasize the use of a specific notation.

2.4 Architectural Evaluation Process and Methods

Software architecture evaluation focuses on assessing a software architecture design to determine whether it satisfies the required architectural requirements. Next, we describe some relevant methods that support this process.

The Software Architecture Analysis Method (SAAM) (Kazman et al., 1996) is a scenario-based evaluation method. Although SAAM works for scenarios related to different quality attributes requirements, it is considered that the main one SAAM analyzes is modifiability. SAAM can be used either for a single architecture or for comparison of multiple ones. For a single architecture, SAAM's activities are scenario development, which requires the presence of all stakeholders, SA description, individual scenario evaluation and scenario interaction. In this case, the cost of scenario modification is estimated by listing the components and the connectors that are affected and then counting the number of changes. In the case of using SAAM to compare multiple architectures, scenarios and the scenario interactions are weighted according to their importance. This metric is used to determine an overall evaluation of the candidate architectures.

The Architecture Tradeoff Analysis Method (ATAM) (Clements, Kazman, & Klein, 2002) is an evaluation method based on SAAM. However, and in contrast to the former, ATAM explores quality attribute scenarios of any type to discover sensitivity points, trade-off points and risks within a set of candidate architectural structures. In ATAM a sensitivity point is a property resulting from a design decision which directly impacts the achievement of a particular quality attribute. A trade-off point is a property that affects multiple quality attributes. A risk is a design decision that was incorrectly taken or not taken at all. Finally, it is important to mention that ATAM is designed to support the evaluation of systems whose quality attribute requirements may not have been documented when the evaluation took place. Thus, ATAM considers, as part of its initial steps, the identification of the quality attributes requirements.

In ACDM, the eight-stage method introduced in the previous sections, stages 4-6 focus on evaluation. In stage 4, the architectural design is reviewed to discover issues that may compromise the satisfaction of the architectural drivers. In order to do so, the architecture design team evaluate the initial architectural design (or reevaluate the refined design after architectural evaluation and experimentation, see stages 5-6 below). Based on this review, it is determined in stage 5 whether the architectural design is ready for production or not. If it is not, some experimentation is carried out in stage 6 to address the issues that were discovered during the review. Based on the results of the experiments, the team refine the architecture design (ACDM stage 3 described in the architectural design section). This sequence of activities is repeated until all the issues have been addressed.

In RUP, within the architecture refinement activity there is a task named Review the Architecture whose focus is to perform an architectural evaluation. The review is conducted as a meeting and there are recommendations with respect to the approaches that can be used to do the review. These include reviewing the architectural model (representation-driven review), reviewing data and measurements (information-driven review) and reviewing scenarios (scenario-driven review). RUP does not provide more specific guidelines on how to conduct these particular reviews, and the Review the Architecture task script only emphasizes the fact that issues must be identified during the review and assigned to the person responsible for their resolution.

Some other methods that support software architecture are Architecture-Level Modifiability Analysis (ALMA) (Bengtsson et al., 2000) (Lassing et al., 2002), Performance Assessment of Software Architecture (PASA) (Ali Babar & Gorton, 2004) and Active Reviews for Intermediate Designs (ARID) (Clements, 2000).

3. PROBLEMS WITH ADOPTING SOFTWARE ARCHITECTURE PROCESSES AND METHODS

Unfortunately, despite a growing body of methods to support software architecture processes during the past years, at present we consider that not many organizations have adopted these methods in practice, at least, not as they are currently defined. The following list includes what we consider the main problems that have contributed to this:

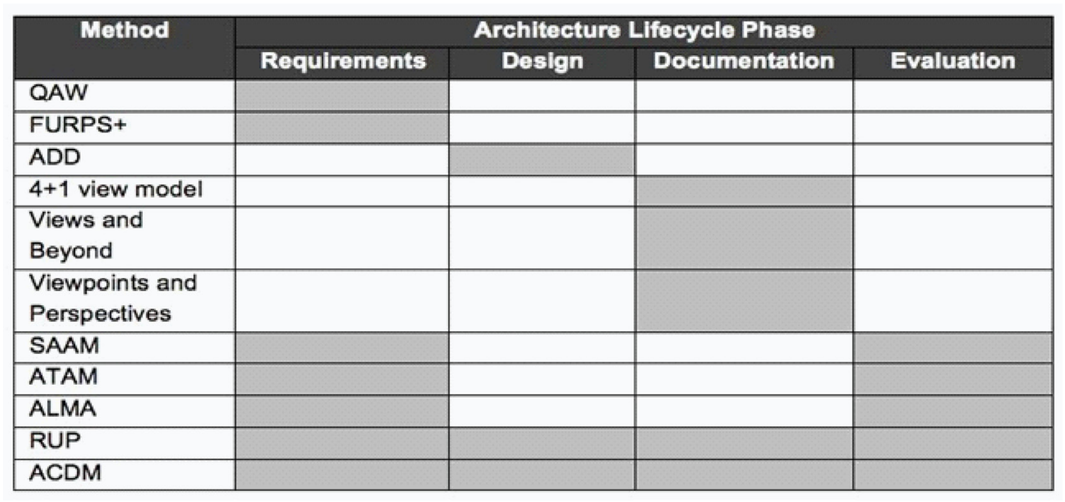
1. Selection of methods for the software architecture lifecycle.
2. Heterogeneity of the existing methods.
3. No consideration for the software development process.
4. Architectural design methods are decoupled from everyday practice.
5. Difficulty of organizational deployment.

It is important to highlight that we have heard about these problems from practitioners in the field as well as from our own experience with clients and industry contacts. In the following sections, we describe these problems in more detail.

3.1 Problem #1: Selection of Methods for the Software Architecture Lifecycle

Table 1 shows (in grey) the phases of the software architecture lifecycle covered by the methods reviewed in this chapter. As this table shows, only ACDM and RUP cover the complete lifecycle. RUP, however, is a general software development process and the guidance that it provides with

Table 1. Phases of the software architecture development lifecycle covered by the methods reviewed



Method	Architecture Lifecycle Phase			
	Requirements	Design	Documentation	Evaluation
QAW	▨			
FURPS+	▨			
ADD		▨		
4+1 view model			▨	
Views and Beyond			▨	
Viewpoints and Perspectives			▨	
SAAM	▨			▨
ATAM	▨			▨
ALMA	▨			▨
RUP	▨	▨	▨	▨
ACDM	▨	▨	▨	▨

respect to each of the phases in the architecture lifecycle is limited. The rest of the methods only cover specific phases of the architecture lifecycle.

The fact that architecture methods generally focus on particular phases of the lifecycle requires selecting an appropriate combination of methods. Table 1 also shows that there is more than one method to choose from for a particular phase of the lifecycle. As can be implied, not only can the number of available methods complicate the selection, but also the lack of knowledge of software architecture and experience in using these methods.

3.2 Problem #2: Heterogeneity of the Existing Methods

In the previous section we discussed the problem of selecting an adequate combination of methods to cover the architecture lifecycle. However, choosing the methods is not all that is needed. In order to progress beyond the selection of individual methods, it is necessary to stand back and identify how the selected methods should properly be used together. This is not a trivial task because these methods have usually been defined by different authors "in isolation," and therefore they are defined in terms of different activities, work products and terminology.

We have noticed that even methods that share a common heritage do not provide explicit support to combine them. To give an example, consider ATAM, the method to support the architecture evaluation process; and QAW, the method to support the architectural requirements analysis process, both developed by the Software Engineering Institute (SEI) (Software Engineering Institute, 2012). At the beginning, ATAM requires quality attributes for the system to be identified. This is because ATAM can be performed on a system whose quality attributes are not documented. However, if a requirements method such as QAW has been used previously, the initial steps of ATAM may be unnecessary.

Thus, once a particular combination of methods is chosen, the architect must often analyze and modify them to avoid mismatches, omissions or repetitions.

3.3 Problem #3: No Consideration of the Software Development Process

Another important problem is that architectural development methods are typically defined independently of a particular software development process. As far as we know, only the author of ACDM provides a detailed description of how to integrate it with different software development processes such as Extreme Programming, Scrum, Team Software Process (TSP), Rational Unified Process (RUP) and Agile Unified Process (AUP) (Lattanze, 2009). For the rest of the methods very little or no guidance is given to help architects to use them within the context of specific software development processes. Thus, the introduction of architectural development methods into an organization often requires adapting both the organization's development process and the architectural development methods to fit properly (Kazman, Nord, & Klein, 2003).

It is important to highlight that, when provided, the guidance is typically generic and therefore difficult to apply to specific situations. Success often depends on the context and characteristics of the organization interested in using the methods. The adaptation of the architectural development methods and the development are part of the activities of organizational deployment discussed in section 3.6.

3.4 Problem #4: Architectural Design Methods Are Decoupled from Everyday Practice

Architectural design is performed by applying design decisions to satisfy a set of architectural requirements. Examples of design decisions, within the context of the categories discussed in (Bass, Clements, & Kazman, 2012), are shown in Table 2. All the design decisions listed in this table are very important for the success of the system and for its evolution. However, the final category of design decisions and choice of technology are very critical to the success of the system.

Unfortunately, most software architecture design methods say very little on the choice of technology (Hofmeister et al., 2007) and often deal in abstract concepts such as tactics and patterns. These concepts are different from the ones that software architects use in their day-to-day work, which mostly come from development frameworks such as JSF (Java Server Faces), Spring, Hibernate or Axis (Cervantes, Velasco-Elizondo, & Kazman, 2013). Frameworks are related to patterns and tactics because they instantiate these concepts. However, as the mapping among all these concepts is not very evident in architectural design methods, software architects are often unwilling to use them.

3.5 Problem #5: Difficulty of Organizational Deployment

The introduction of architectural methods into an organization, whose processes are documented and used, often has a high initial cost due to the need to change several existing process elements. This cost is not only limited to the cost of making changes in the processes elements, it also often comprises the cost of training and technology investment.

Table 2. Examples of design decisions within the context of the categories discussed in (Bass, Clements, & Kazman, 2012)

Category	Examples
Allocation of responsibilities	• Determination of basic system functions. • Definition of the architectural infrastructure. • Determination of how responsibilities are allocated to architectural elements.
Coordination model	• Determination of the elements of the system that must be coordinated. • Definition of coordination properties, e.g. timeliness, currency, completeness, correctness, and consistency. • Selection of communication mechanisms to support coordination properties.
Data model	• Determination of main data abstractions. • Definition of operations and properties of data abstractions. • Definition of any metadata needed for consistent interpretation of data abstractions.
Management of resources	• Determination of the resources that must be managed. • Determination of the system elements that manage each resource. • Selection of the strategies employed when there is contention for or saturation of resources.
Mapping among architectural elements	• Specification of the mapping of runtime elements that are created from each module. • Specification of the modules that contain the code for each runtime element. • Specification of the assignment of runtime elements to processors and data items in the data model to data stores
Binding time decisions	• Establishment of the point in the life cycle and the mechanism for achieving a variation.
Choice of technology	• Determination of the available technologies to realize the decisions made in the other categories. • Determination of the available tools to support technology choices, e.g. IDEs, testing tools. • Determination of the side effects of technology choices.

Training is a fundamental aspect when introducing architecture development methods. Software architects are generally proficient developers with considerable experience. However, this does not guarantee that they are knowledgeable about software architecture concepts. Thus, training courses and coaching activities are often required not only for the software architect, but also for the people that the architect deals with within the organization. Technological support is also an important issue as the selection of appropriate tools is crucial to allow the architects to develop the architecture and communicate it in an easy and, ideally, in an automated or semi-automated manner.

It should also be noted that when an organization decides to use a new method to perform a specific activity, it is creating a change in the way people work. This can generally have a negative impact on (people's) productivity in the early stages. It should also be noted that an organization might need to invest a significant amount of effort to get people to adapt to the new processes.

4. USING ARCHITECTURE PROCESSES IN PRACTICE

This section discusses how the five issues listed previously were addressed in a large software development company in Mexico City through the introduction of software architecture development processes and methods. This company, which is currently rated at CMMI-DEV level 5, develops software for government and private customers using the Team Software Process (TSP) (Humphrey, 2000). In 2010 the architecture method introduction project was conducted on some aspects of the company as follows:

- The company had, at that time, a CMMI level 3 rating which means, among other things, that all of its processes associated with requirements and design were documented.
- There was a lack of experience in capturing quality attribute requirements. Furthermore, typical customers encountered difficulties while trying to express these types of requirements.
- The role of the software architect existed and the organization tried to assign a software architect to every team, although sometimes this was not possible due to the insufficient number of architects in the organization. People who took on the architect role were typically highly experienced developers with high technical proficiency, but usually little theoretical foundation in software architecture.
- The architect, along with the team leader and core developers, were selected at the beginning of the project and they usually worked together throughout the project where they participated in several activities such as requirements, high-level design, component development and testing.
- Development contracts typically required all of the requirements to be elicited initially.
- The project's cost and schedule were determined very early on before the actual requirements phase was performed. During this initial estimate, quality attributes were not frequently considered but an initial architecture proposal had to be established nonetheless.

The particular context of this company introduces specific constraints that affect the way the five problems discussed in section 3 were addressed.

4.1 Addressing Problem #1: Selection of Methods

As previously discussed, the first problem to be addressed involves the selection of methods for the software architecture lifecycle. Next, we discuss how methods for every phase of the architecture lifecycle were selected (see summary in Table 3).

4.1.1 Requirements Phase

For the requirements phase, the methods listed in section 2.1 were considered. While QAW is the most complete method with respect to quality attribute requirements, it was decided not to adopt it initially because of the lack of maturity in the company on elicitation of quality attributes as well as the difficulties associated in conducting meetings with relevant stakeholders. The decision, instead, was to define a custom method for requirements engineering of quality attributes, which would complement the existing requirements process of the company and which would help provide some initial level of maturity with respect to quality attribute elicitation. The scenario technique was retained along with an impact analysis technique associated with FURPS+. Prioritization was performed with the customer using a technique taken from ATAM where every scenario is given two ratings, which can take a Low, Medium or High value. The two ratings correspond to the importance of a quality attribute scenario for the customer and the difficulty of implementation from the architect's perspective.

4.1.2 Design Phase

For the design phase, only ADD was considered because this method provides the most detailed process for designing in a systematic way. Selecting ADD posed no significant problems since the company did not have any architectural design process in place.

4.1.3 Documentation Phase

For the documentation phase, the fact that the company already had several artifacts in place for documenting the software architecture had to be considered. These artifacts included a design document based on the 4+1 Views method. It was not necessary to make a complete change to the document so it was decided that only the concept of view packages and the associated templates from the V&B would be adopted.

Table 3. Summary of information on method selection

Architectural Lifecycle Phase	Constraints	Selected Method
Requirement analysis	- Lack of experience in quality attributes - Existing requirement process - Difficulty in involving customers	Custom quality attribute elicitation method
Design	- Existing architectural sketch from early estimation	ADD
Documentation	- Existing standard based on the 4+1 Views method	V&B (only view packages and templates)
Evaluation	- Availability of other architects for the evaluation team - Architects' limited time	ACDM stage 4 ("Evaluate the architectural design")

4.1.4 Evaluation Phase

For the evaluation phase, there was no equivalent activity in the existing organizational process. One benefit associated with the size of the company is that there was a reasonable number of architects that could participate as members of an architecture evaluation team so that performing scenario-based evaluations could be achieved. The constraint, however, was that their availability was limited, so the evaluation meeting had to be performed in a short time. Among the scenario-based evaluation methods, both ATAM and ACDM Stage 4 (Evaluate the Architectural Design) were considered. ATAM was discarded because it typically requires two days to carry out an evaluation and, furthermore, some steps of ATAM are rendered unnecessary because quality attributes are captured using the requirements method. The final decision was to select the process defined by ACDM's Stage 4.

4.2 Addressing Problem #2: Adapting and Connecting the Methods

The second problem involved adapting the methods to overcome heterogeneity, resulting from the fact that methods are defined in isolation. Next, we discuss how methods for every phase of the architecture lifecycle were adapted to overcome this heterogeneity.

4.2.1 Requirements Phase

Since the process for the requirements phase was a custom method, it required no particular adaptation of an existing method. However, one aspect that was considered, in addition to elicitation of quality attributes requirements, was the identification of other architectural drivers, including functional requirements and constraints. These architectural drivers were identified by adding a primary use case selection activity and listing constraints (which had not been formally identified previously). Complementing the quality attribute elicitation custom method with the selection of primary use cases and constraint identification provided all the necessary inputs for the design phase.

4.2.2 Design Phase

The ADD method was adopted with minor modifications. One important aspect that was considered was that the initial design iteration does not start "from scratch," but rather with a preliminary architecture sketch that is established as part of the early estimation process. This preliminary architecture constrains the decisions that the architect can make during the design process. Furthermore, the design process based on ADD emphasizes the use of technology, besides patterns and tactics, and the creation of an executable architecture as one of the outputs of the design process. Other aspects that were considered were guidelines to model the architecture in a case tool so that the documentation packages could be produced in a very straightforward way.

4.2.3 Documentation Phase

Regarding the documentation method, the view template from V&B was adopted without modifications. Since the company already had an architecture document based on 4+1 Views which mandated the inclusion of module, allocation and component-and-connector views, the view selection activity from V&B was not adopted. The original views were replaced with view packages and at least one view package associated with the module, component-and-connector and allocation was included.

4.2.4 Evaluation Phase

The process defined in ACDM Stage 4 was used without modifications. This process, however, was complemented by adding a preparation phase where an "evaluation package" was assembled. This package includes information concerning the business goals, the architectural drivers and the views produced in the documentation phase. Furthermore, once the evaluation method is performed, a follow-up activity is performed to support the architect in dealing with the observations raised during the evaluation meeting.

4.3 Addressing Problem #3: Integrating the Methods with the Team Software Process

The Team Software Process, as its name suggests, is a development process oriented towards teams, which is built on top of the Personal Software Process (PSP) (Humphrey, 2005). Data collected from the TSP projects reveal that projects developed using TSP do indeed achieve substantially better results than typical projects (0.06 defects/KLOC versus 7.5 defects/KLOC after delivery) (Davis & Mullaney, 2003).

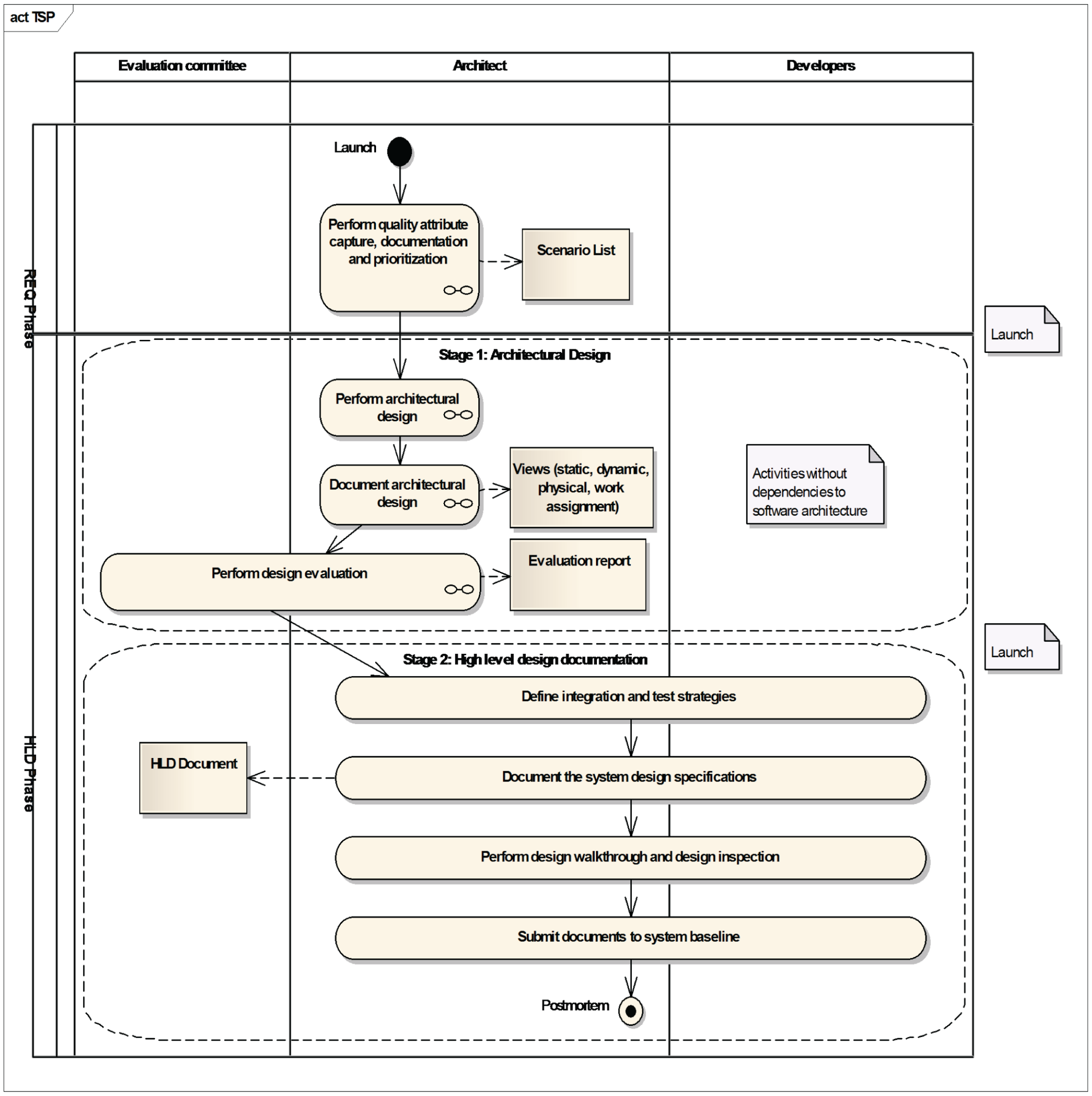
A TSP software project is performed as a series of development cycles, where each cycle begins with a planning process called a launch and ends with a closing process called a postmortem. Within each development cycle, activities belonging to different phases can be performed. These phases include: requirements (REQ), high-level design (HLD), implementation (IMPL) and testing (TEST). The REQ phase of TSP focuses on producing a complete System Requirements Specification document (SRS). The main goal of the HLD phase is to produce a high-level design that will guide product implementation. This high-level design must define the components that compose the system and that have to be designed and developed independently using PSP in the IMPL phase. Finally, the TEST phase focuses on performing integration and system testing and on preparing the delivery of the system. It must be noted that the lifecycle model of a particular project (waterfall, incremental) is defined by the phases that are performed in each cycle.

TSP does not give full consideration to software architecture development. None of the roles defined in TSP are that of software architect, which (generally speaking) denotes the person responsible for performing the process of software architecture development discussed previously. Furthermore, the script for the REQ phase does not provide specific guidelines to support the identification of architectural drivers, which are necessary to design the architecture. The HLD script focuses on designing a general structure to guide development, but no explicit consideration is given to satisfying quality attributes in this process. A further problem involves the fact that TSP does not mandate an architectural evaluation to be performed. The closest activities include a design walkthrough and the inspection of the design document. These activities, however, are performed by other team members, who may have less experience than the architect with respect to designing and, as a consequence, may not detect complex design problems.

The UML activity diagram in Figure 2 shows a general overview of the introduction strategy of software architecture development into the TSP (Cervantes, Martinez, Castillo, Montes de Oca, 2010). Vertical swimlanes represent the roles that participate in architecture development activities and horizontal swimlanes represent TSP phases (REQ and HLD). Within the HLD phase, two regions represent distinct stages. Composite activities, such as Perform Architectural Design, represent architectural development methods and objects represent artifacts produced by these methods.

As the diagram shows, the requirements method is included as part of the REQ phase of TSP and its execution produces a list of scenarios. The remaining methods are all part of the HLD phase and they are performed as the initial activities of this phase. The HLD phase is thus di-

Figure 2. Overview of architecture lifecycle phases introduced into TSP


act TSP
Evaluation committee
Architect
Developers
REQ Phase
HLD Phase
Launch
Perform quality attribute capture, documentation and prioritization
Scenario List
Launch
Stage 1: Architectural Design
Perform architectural design
Document architectural design
Views (static, dynamic, physical, work assignment)
Activities without dependencies to software architecture
Perform design evaluation
Evaluation report
Launch
Stage 2: High level design documentation
Define integration and test strategies
HLD Document
Document the system design specifications
Perform design walkthrough and design inspection
Submit documents to system baseline
Postmortem

vided into two stages: architectural design stage and high-level design and documentation stage. During the architectural design stage, the activities previously discussed that culminate in an evaluated architectural design are performed. This initial stage is performed mainly by the architect, but other architects from outside the project also participate during the evaluation of the architecture. In the high-level design and documentation stage, the team design and document the rest of the system based on the architectural design. This design typically involves creating sequence diagrams for all of the use cases, which allow the interfaces of all of the components to be specified. This specification is later used in the development phase (IMPL) for performing detailed design and development of the components.

The benefits of this approach is that the high-level design and documentation stage is performed using an architecture that has been evaluated.

Furthermore, the evaluation team participate in the architectural evaluation while other team members with less experience participate in the inspection of the architecture document at the end of the HLD phase.

4.4 Addressing Problem #4: Considering Frameworks during Architectural Design

We have discussed the problem that many software architecture design methods often deal with abstract concepts such as tactics and patterns, while software architects mostly use those that come from development frameworks. To address this problem we proposed an approach where frameworks are used as design concepts on par with tactics and patterns. The approach was realized as an extension to the ADD method. However, it can be applied to other architecture design methods as well.

Recalling section 2.2, the ADD assumes the existence of a set of architectural drivers and follows a top-down recursive decomposition-based approach where, at each iteration, tactics and patterns are applied to satisfy a subset of drivers. Table 4, shows an excerpt of what results from the first design iterations when frameworks are considered as design concepts. The iterations correspond to the greenfield development of a system to buy bus tickets: a typical enterprise application where large numbers of users interact with the system through a browser or mobile apps and perform processes such as checking bus schedules that act on data in a database. Functional architectural requirements include searching for bus schedules. The most important quality attribute scenario is performance: performing searches for timetables in less than 10 seconds, and constraints include time to market for the initial system release and having a small development team with experience in JSF, Spring and Hibernate.

In contrast to the traditional manner of performing the ADD, several frameworks are selected in early iterations. Although many frameworks exist, the ones selected were favored because of one of the architectural drivers in iteration 2. Once frameworks are chosen, further design iterations are impacted by this decision. To satisfy the performance scenario, in iteration 3, at the data layer, performance was addressed by configuring the parameters provided by the framework (Hibernate Community Documentation, 2004). In this case, Hibernate incorporates the Lazy Load Pattern, but it also incorporates tactics such as support for a cache (an instance of the "Maintain Multiple Copies" tactic) that allow performance to be improved. A detailed description of this design approach can be found in (Cervantes, Velasco-Elizondo, & Kazman, 2013).

Table 4. Excerpt of the initial ADD design iterations when using frameworks as design concepts

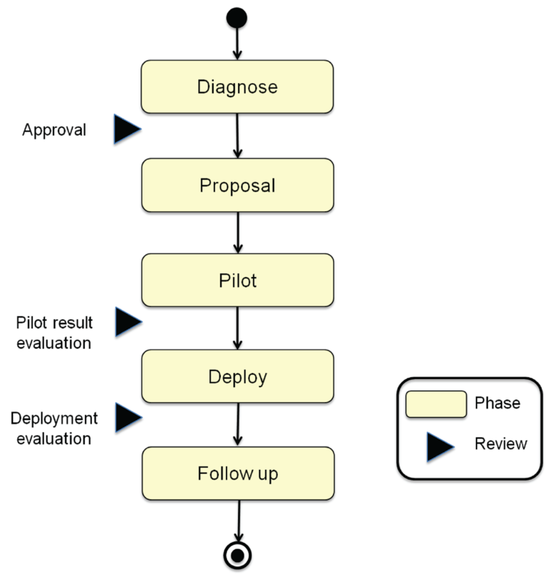
Iteration	Architectural Drivers	Element to Decompose and Designs Decisions
1	• Web access and support for mobile apps • Time to market for the initial release • Small development team	• Element: The whole system • Design Decisions: Apply the 3-Layers Pattern (presentation, business and data)
2	o Searching for bus schedules • Team experience with frameworks	• Element: The 3-Layers • Design Decisions: Apply the Application Service Pattern, Use of JSF, Spring and Hibernate for the presentation, business and data layers respectively.
3	• Performance scenario	• Element: The data layer • Design Decisions: Apply the Lazy Loading Pattern and the Maintain Multiple Copies tactic, both by configuring Hibernate support for lazy associations and caches.

4.5 Addressing Problem #5: Method Deployment

The deployment of architecture development methods in an organization is an endeavor that may be complicated depending on the scope of the changes and the size of the organization. The methods that were previously discussed were introduced into the company by following a systematic approach based on the Organizational Performance Management process area (OPM) of CMMi (Chrissis, Konrad & Shrum, 2010). Figure 3 shows the general steps that were followed. Next, we describe them.

- **Diagnose:** During this step, several activities were performed. These include analyzing the existing processes of the organization, observing development teams, interviewing architects, and studying work products. This step revealed many issues, an example is poor documentation of quality attribute requirements.

Figure 3. Steps followed in the introduction of architecture development methods



- **Proposal:** During this step, the activities discussed in sections 4.1, 4.2 and 4.3 were performed in order to address the problems identified during the diagnosis.
- **Pilot:** Piloting the proposal is necessary in order to 1) understand whether the proposal can really be applied in the context of a real project and 2) make adjustments to the proposal based on the results of its use. During the pilots, adjustments to the methods were made. An example of this is the modification of ADD to consider frameworks as design concepts, as discussed in section 4.4.
- **Deploy:** This is one of the most complex steps as it requires many activities to be performed. These activities include creating training materials and then training the architects, modifying the existing organizational processes and also championing the use of architecture methods.
- **Follow-up:** Follow-up involved coaching the architects and also collecting data about the results of the use of the methods.

Although the deployment of the methods is treated here very briefly, the aspects associated with organizational change management are complex and need to be given serious consideration in order to successfully introduce architectural development methods into an organization.

5. DISCUSSION

In the following sections we provide a discussion on (1) general observations and lessons learnt from the implementation of the actions described in this paper and (2) specific observations derived from coaching architects.

5.1 General Observations and Lessons Learnt

The material presented in the above sections provides an example of how the five problems listed were addressed in the software company. While it would be unwise to draw definitive conclusions from it, it is possible to make some general observations and discuss some valuable lessons learnt for most of the problems addressed.

With respect to problems 1 and 2, method selection is a complex task as it depends on the context of the organization. As there are many methods, both commercial and academic, each one of them defining heterogeneous activities and work products, it is necessary to properly use all this information not only to select the "best combination of methods" but also to adapt them. Method selection and, in particular, method adaptation also requires process engineers to work closely with software architects in order to make useful adaptations.

Regarding problem 3, it is important to consider the impact of the introduction of architectural methods into existing organizational processes. Although in the context of the company studied these changes seem significant, in the end the number of affected process elements was relatively small compared to the overall process repository. We believe, however, that to minimize the risk of process change, a process engineer should systematically perform analyses not only to identify the affected process elements but also to estimate the quantitative impact of the change on people performance.

Regarding problem 4, reducing the de-coupling between methods and everyday practice is often the result of performing pilot projects with the proposed methods. It is important to bridge this gap between theory and practice to facilitate the adoption of the methods.

Finally, with respect to problem 5, the aspects associated with organizational change management are complex and need to be given serious consideration in order to successfully introduce architectural development methods into an organization. If these aspects are not taken into consideration, no matter how well architectural methods are selected, connected and adapted to the development process, the possibility of them being deployed in the organization in a successful way is limited. The introduction of architecture development methods can be simplified by having somebody knowledgeable about the methods coaching the architects. In the organization studied, one of the authors performed this task, which allowed the observation of some specific aspects that are discussed in the next section.

5.2 Specific Observations from Coaching Architects

Regarding architectural drivers, specifying quality attribute requirements as scenarios was a difficult task. The identification of quality attribute types is not straightforward and deriving them from business goals requires certain experience. The most difficult part, however, is the definition of a measure to express the scenario's response in a quantitative manner. The SEI suggests the use of quality attribute scenario generation tables, which are templates that provide many choices for creating scenarios for a particular quality attribute category. To be effective these tables need, however, to be suited to the organization's type of products and this requires the study of many quality attribute scenarios produced by the organization, something that would not be possible during the initial introduction.

On the other hand, it was observed that the architects often needed to have clear criteria to establish how much architectural design is appropriate. Before the proposal was presented to the organization in question, there was great variation among architects with respect to the criteria used for end design. Currently, architects are asked to make a list of architectural drivers and to perform design activities until decisions have been taken for all of the architectural drivers. Although this may not always be possible depending on the time allocated to HLD in the project, this has proven to be a good criterion because even if an architect does not finish his design, he is aware of the drivers that he has not considered.

Some other aspects that are worthy of mention are the participation of customers in software architecture development activities and how software architecture supports software estimation and allocation of work. Regarding the first aspect, involving customers in activities related to software architecture development was not feasible in the early stages of the introduction of the methods into the organization. This is mainly due to the lack of maturity with respect to the use of these methods. For example, in the organization studied the identification and correct specification of quality attributes took some time. If QAW is used for architectural requirements analysis, this could build false expectations from customers. Regarding the second aspect, the availability of a software architecture helps not only to reduce the risks associated with software estimates but also to develop better work assignment. In the organization studied, many projects were planned and estimated based only on information pertaining to functional aspects of the system. Currently, there is also information about quality attributes, which is prioritized according to its importance for the customer and the difficulty of implementation. It should also be noted that in the context of the organization studied, the creation of a work assignment structure is essential in TSP to guide development in the IMPL phase. Furthermore, this structure also provides a clear guide to identify which interfaces must be documented thoroughly.

6. CONCLUSION

Over the past years, attention to the introduction of software architecture development has increased as software architecture has been recognized as an important artifact for high quality system development. Although software architecture development is supported by a variety of methods, their adoption is complicated because many of the methods have been defined without considering all the software architecture development activities, a specific organization environment or a particular software development life-cycle.

In this chapter we have described some actions to address the problems mentioned above and described the benefits observed when implementing them in an software development company in Mexico City, currently rated at CMMI-DEV level 5, which develops custom software for government and private customers. Although there is not enough data to evaluate quantitatively the real benefits of these actions, there are some preliminary positive results that have led to some valuable lessons learnt.

In order to improve the evaluation of the benefits of the actions described in this paper, we plan to carry out a systematic analysis of defects related to software architecture found during evaluations, system tests or after the system has been transitioned to customers. This type of analysis, however, may not be possible in the short term as it requires a long testing period and a significant number of projects to be performed so that sufficient data can be gathered.

ACKNOWLEDGMENT

The authors wish to thank CIMAT and the editors of this book for their kind invitation to be part of this project. We would also like to thank Quarksoft S.A. de C.V., the company where the software architecture methods were introduced.

REFERENCES

Ali Babar, M., & Gorton, I. (2004). Comparison of scenario-based software architecture evaluation methods. In *Proceedings of the Asia-Pacific Software Engineering Conference* (pp. 600-607). IEEE Computer Society.

Bachmann, F., Bass, L., Chastek, G., Donohoe, P., & Peruzzi, F. (2000). *The architecture based design method* (Technical Report CMU/SEI-2000-TR-001). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Barbacci, M., Ellison, R. J., Lattanze, A. J., Stafford, J. A., Weinstock, C. B., & Wood, W. G. (2003). *Quality attribute workshops (QAWs)* (Technical Report CMU/SEI-2003-TR-016). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Bass, L., Clements, P., & Kazman, R. (2012). *Software architecture in practice* (3rd ed.). Reading, MA: Addison-Wesley Professional.

Bengtsson, P., Lassing, N., Bosch, J., & Vliet, H. (2000). *Analyzing software architectures for modifiability* (Technical Report HK-R-RES–00/11-SE). Högskolan Karlskrona/Ronneby.

Buschmann, F., Henney, K., & Schmidt, D. (2007). Pattern-oriented software architecture: Vol. 4. *A pattern language for distributed computing*. Chichester, UK: Wiley.

Cervantes, H., Martinez, I., Castillo, J., & Montes de Oca, C. (2010). Introducing software architecture development methods into a TSP-based development company. In *Proceedings of SEI Architecture Technology User Network (SATURN 2010) Conference*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Cervantes, H., Velasco-Elizondo, P., & Kazman, R. (2013). A principled way of using frameworks in architectural design. *IEEE Software*, *30*(2), 46–53. doi:10.1109/MS.2012.175

Chrissis, M. B., Konrad, M., & Shrum, S. (2010). *CMMi for development: Guidelines for process integration and product improvement* (3rd ed.). Reading, MA: Addison-Wesley Professional.

Clements, P. (2000). *Active reviews for intermediate designs* (Technical Report CMU/SEI-2000-TN-009). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Reed, L., & Nord, R. (2011). *Documenting software architectures: Views and beyond* (2nd ed.). Reading, MA: Addison-Wesley Professional.

Clements, P., Kazman, R., & Klein, M. (2002). *Evaluating software architectures: Methods and case studies*. Reading, MA: Addison-Wesley Professional.

Davis, N., & Mullaney, J. (2003). *The team the team software ProcessSM (TSPSM) in practice: A summary of recent results (Technical Report, CMU/SEI-2003-TR-014)*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Eeles, P. (2012). *Capturing architectural requirements*. Retrieved from http://www.ibm.com/developerworks/rational/library/4710.html

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Reading, MA: Addison-Wesley Professional Computing Series.

Hibernate Community Documentation. (2004). *Improving performance*. Retrieved from http://docs.jboss.org/hibernate/orm/3.3/reference/en/html/performance.html

Hofmeister, C., Kruchten, P. B., Nord, R., Obbink, H., Ran, A., & America, P. (2007). A general model of software architecture design derived from five industrial approaches. *Journal of Systems and Software*, *80*(1), 106–126. doi:10.1016/j.jss.2006.05.024

Humphrey, W. (2000). *The team software process (TSP)* (Technical Report CMU/SEI-2000-TR-023). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Humphrey, W. (2005). *PSP, a self-improvement process for software engineers*. Reading, MA: Addison-Wesley Professional.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The unified software development process*. Boston, MA: Addison-Wesley.

Kazman, R., Abowd, G., Bass, L., & Clements, P. (1996). Scenario-based analysis of software architecture. *IEEE Software*, *13*(6), 47–55. doi:10.1109/52.542294

Kazman, R., Nord, R., & Klein, M. (2003). *A life-cycle view of architectural analysis and design methods (Technical Note CMU/SEI-2003-TN-026)*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Kroll, P., Kruchten, P. B., & Booch, G. (2003). *The rational unified process made easy*. Reading, MA: Addison-Wesley Professional.

Kruchten, P. B. (1995). The 4+1 view model of architecture. *IEEE Software*, *6*(12), 42–50. doi:10.1109/52.469759

Lassing, N., Bengtsson, P., Vliet, H., & Bosh, J. (2002). Experience with ALMA: Architecture-level modifiability analysis. *Journal of Systems and Software*, *61*, 47–57. doi:10.1016/S0164-1212(01)00113-3

Lattanze, A. J. (2009). *Architecting software intensive systems: A practitioners guide*. Boca Raton, FL: CRC Press.

OpenUP. (2012). Retrieved from http://epf.eclipse.org/wikis/openup

Rozanski, N., & Woods, E. (2012). *Software systems architecture: Working with stakeholders using viewpoints and perspectives*. Reading, MA: Addison-Wesley.

Shaw, M., & Clements, P. (2006). The golden age of software architecture. *IEEE Software*, *2*(23), 31–39. doi:10.1109/MS.2006.58

Software Engineering Institute. (2012). Retrieved from http://www.sei.cmu.edu/

KEY TERMS AND DEFINITIONS

Architectural Design: The phase of the software architecture development lifecycle that focuses on identifying and selecting the different structures that compose the architecture and that will allow architectural requirements to be satisfied.

Architectural Documentation: The phase of the software architecture development lifecycle that focuses on creating the documents that describe the different structures that compose the architecture for the purpose of communicating it efficiently to the different system stakeholders.

Architectural Evaluation: The phase of the software architecture development lifecycle that focuses on assessing a software architecture design to determine whether it satisfies the required architectural requirements.

Architectural Requirements Analysis: The phase of the software architecture development lifecycle that focuses on eliciting, analyzing, specifying and prioritizing architectural requirements so that they can later be used to drive the design of the architecture.

Software Architecture Development Lifecycle: It imposes a structure on the activities for software architecture development. The software architecture development lifecycle is composed of the following phases: architectural requirements analysis, architectural design, architectural documentation, and architectural evaluation. Each one of the phases involves principles, practices and methods used to develop software architecture.

Software Architecture Development: It is the set of activities that are typically performed early in a software development project, which contribute to creating the software architecture of a system.

Software Architecture: It is the structure (or structures) of this system, which comprises software elements, the externally visible properties of those elements, and the relationships among them (Clements et al, 2010).