A Framework for Software Reference Architecture Analysis and Review
— Thesis Proposal —

Silverio Martínez-Fernández
Advised by Xavier Franch and Claudia Ayala

Barcelona, Spain
June 2013
Abstract

Nowadays, the size and complexity of software systems, together with critical time-to-market needs, pushes organizations to continuously look for techniques to improve their IT services in general, and the design of software architectures in particular. The use of software reference architectures allows organizations to reuse architectural knowledge and software components in a systematic way and, hence, to reduce costs.

Software reference architectures mainly appear in organizations in which the multiplicity of software systems (i.e., systems developed at multiple locations, by multiple vendors and across multiple organizations) triggers a need for life-cycle support for all systems. Therefore, software reference architectures are very attractive when organizations become large and distributed in order to develop new systems or new versions of systems. In return, organizations face the need to analyze the return-on-investment in adopting software reference architectures, and to review these software reference architectures in order to ensure their quality and incremental improvement.

This thesis proposal presents the need of supporting organizations to decide on the adoption of software reference architectures and its subsequent suitability for the organization purposes. We propose an empirical framework aimed to help practitioners in such tasks by harvesting relevant evidence in software reference architecture projects from the wide spectrum of involved stakeholders and commonly available information and documentation. Such a framework comes from an action-research approach held in everis, an IT consulting firm.

This document presents the state-of-the-art and the first steps that have been carried out to achieve these goals, together with a schedule for forthcoming work.
# Contents

1 Introduction  
1.1 Problem .......................................................... 2  
1.2 Research Context ................................................. 3  
1.2.1 The “Cátedra everis-UPC” project ......................... 3  
1.3 Structure of the document ....................................... 5  

2 Background  
2.1 Architecture Disciplines Basic Concepts ..................... 7  
2.1.1 Definitions of architecture disciplines .................... 7  
2.1.2 Relationships between architecture disciplines and so-  
   lution architecture ............................................. 8  
2.1.3 Where Software Reference Architecture belongs to ...... 8  
2.1.4 Summary ..................................................... 9  
2.2 Software Reference Architecture Essentials ................. 11  
2.2.1 Definition of Software Reference Architecture .......... 11  
2.2.2 The Industrial Context of Software Reference Architec-  
tures ............................................................... 12  
2.2.3 Types of Software Reference Architecture ................ 15  
2.2.4 Which are the elements that compose a Reference Ar-  
   chitecture ..................................................... 16  
2.2.5 Concrete Software Architecture and Software Reference  
   Architecture .................................................. 17  
2.2.6 Product Line Architectures and Software Reference Ar-  
   chitectures .................................................... 19  

3 State-of-the-art  
3.1 State-of-the-art of Software Reference Architecture Economics  
3.1.1 Business Case Analysis and Return-On-Investment ...... 21  
3.1.2 Economic Models for Software Reference Architectures  
3.1.3 Summary ................................................... 25  
3.2 State-of-the-art of Evaluation Methods for Software Reference  
   Architecture .................................................... 25
Chapter 1

Introduction

Every software system has a software architecture [11, p. 6]: “the software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both” [11, p. 4].

Nowadays, the size and complexity of software systems, together with critical time-to-market needs, demand new software engineering approaches to design such software architectures [53]. One of these approaches is the use of software reference architectures that allows to systematically reuse knowledge and elements when developing a concrete software architecture [22] [30]. A software reference architecture is, then, the baseline for many software systems.

Hence, the purpose of software reference architectures is to serve as guidance for the development, standardization, and evolution of systems [53], as depicted in Figure 1.1. This is possible because software reference architectures are abstract enough to allow its usage in differing systems’ contexts [2]. It has been claimed that “reference architectures reduce the complexity of hardware and software architecture by systematically reducing environmental diversity [...] enables greatly increased speed and reduced operational expenses as well as quality improvements due to lowered complexity, greater investment and greater reuse” [65]. Thus, “IT organizations that lack architecture and configuration standards [...] have higher costs and less agility that those with enforced standards” [65].

Software reference architectures mainly appear in organizations where the multiplicity of software systems (i.e., systems developed at multiple locations, by multiple vendors and across multiple organizations) triggers a need for life-cycle support for all systems [22]. Therefore, software reference architectures are very attractive when organizations become large and distributed [51] in order to develop new systems or new versions of systems.

According to their expected benefits, software reference architectures have become widely studied and used in software architecture research and prac-
Figure 1.1: Software reference architectures serve as guidance for the development, standardization, and evolution of concrete software architectures of software systems.

Despite software reference architectures are widely recognized and used by practitioners [4], their analysis and review has received little attention in the literature. The next section describes this problem.

1.1 Problem

Currently, there is no specific support for the analysis and review of software reference architectures. While comprehensive methods have been defined for the analysis and review of (concrete) software architectures, software reference architecture have received relatively less attention in literature. The reasons for this can be probably traced in an assumption that theory on software architectures is directly applicable to software reference architecture. But in practice, as Angelov et al. point out [4], **practitioners face difficulties in working with software reference architectures.**

This problem originates from the specific features of software reference architectures with respect to software architectures [3], such as the need of an initial investment, their generic nature, the wide group of stakeholders that they involved, their high level of abstraction or their instantiation in the organization’s portfolio of software systems. This situation triggers specific questions that have not been addressed yet. Some questions of quantitative nature: *Is it worth to invest on the adoption of a software reference architecture? How is it possible to calculate the return-on-investment (ROI) of the adoption of a*
1.2. Research Context

This document represents my thesis proposal on the PhD in Computing, at Barcelona School of Informatics (Facultat d’Informàtica de Barcelona) of the Technical University of Catalonia (Universitat Politècnica de Catalunya). It is being done inside the software architecture research area of the Research Group of Software Engineering for Information Systems (GESSI).

The proposed thesis grows around software reference architectures and empirical research as a way of gaining knowledge in software reference architectures. The purpose of the thesis is to analyze quantitatively and qualitatively software reference architectures, mainly motivated by the "Cátedra everis-UPC" project.

1.2.1 The “Cátedra everis-UPC” project

This research has its origin in an ongoing action-research initiative among our research group and everis. As a consulting company, everis offers solutions for big businesses (e.g., banks, insurance companies, public administration,
utilities, and industrial organizations) that provide a wide spectrum of services to their clients. Given the complexity of the resulting software systems, which integrate bespoke applications with commercial packages, these organizations need high-quality software architectures, and this is the service that they hire to everis. The solution provided by everis is based on the adoption of a software reference architecture in the organization, from which concrete software architectures are derived and used in a wide spectrum of applications.

In this context, everis commissioned our research group two main tasks (respectively aligned with our research questions, later presented in Section 4.1):

- to calculate the ROI that organizations get after adopting a software reference architecture.

- and to systematically gather empirical evidence about the elements that compose the software reference architectures that they designed for their clients. In this way, the architectural knowledge of years of work could be captured in a congruent vision and can help everis’ employees in the inception, design, and application of prospective software reference architectures.

First, the architecture group of everis experienced the inability to calculate the ROI derived from software reference architectures that they create (or plan to create) for organizations. The purposes of our research are: on the one hand, to create guidelines for extracting costs and benefits of software reference architectures based on data that they already collected; and on the other hand, to create an economic model to make the software reference architecture business case.

Second, to gather such empirical evidence, it is necessary to contact software reference architecture’s stakeholders [44]. In everis, three essential roles are distinguished: software architects that cooperatively work to figure out a software reference architecture to accomplish the desired quality attributes and architecturally-significant requirements of the client organization; architecture developers that are responsible for coding, maintaining, integrating, testing and documenting the software reference architecture’s software components; and application builders that take reusable components from the software reference architecture and instantiate them to build concrete software architectures for software systems. The context in which these stakeholders work is explain in depth in Section 2.2.2.
1.3 Structure of the document

The rest of this document is structured in the following way:

**Chapter 2: Background.** Chapter 2 presents basic concepts related to software reference architecture. The first section introduces the different disciplines around the concept ‘architecture’. The second section includes the current state of software reference architecture theory.

**Chapter 3: State-of-the-Art.** Chapter 3 summarizes the current state-of-the-art in software reference architectures. On the one hand, the state-of-the-art about economic models for software reference architectures is analyzed. On the other hand, we study which empirical evidence about software reference architecture needs to be investigated. Finally, the last section describes the existing gaps in software reference architecture research that justify further work.

**Chapter 4: Thesis Proposal.** Chapter 4 presents the main goals of the PhD thesis, together with the main stages in the research. Finally, it includes bibliographic information about the work that has been published.
Chapter 2

Background

This chapter presents basic concepts related to software reference architecture. The first section introduces the different disciplines around the concept ‘architecture’. The second section includes the current state of software reference architecture theory.

2.1 Architecture Disciplines Basic Concepts

The term “architecture” has been used extensively, but not always together with software. Two architecture disciplines related to software architecture are system architecture and enterprise architecture [11, p. 7]. This section discusses:

- the definitions of software architecture, system architecture and enterprise architecture,
- the relationships and boundaries between these three architecture disciplines and solution architecture,
- where software reference architecture belongs to.

2.1.1 Definitions of architecture disciplines

As we have already stated, ‘the software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both’ [11, p. 4]

“A system’s architecture is a representation of a system in which there is a mapping of functionality onto hardware and software components, a mapping of the software architecture onto the hardware architecture, and a concern for the human interaction with these components” [11, p. 7].

“Enterprise architecture is a description of the structure and behavior of an organization’s processes, information flow, personnel, and organizational
subunits, aligned with the organization’s core goals and strategic direction” [11] p. 8].

2.1.2 Relationships between architecture disciplines and solution architecture

Software architecture is different from other architecture disciplines (system architecture and enterprise architecture). The main objects of study of a software architecture are: the abstraction of a software system that consists of three components: elements, form, and rationale [59]; and the set of significant decisions about the organization of such software system [39].

However, system architecture and enterprise architecture “have broader concerns than software and affect software architecture through the establishment of constraints within a software system must live” [11]:

- a system architecture is concerned with a total system, including hardware, software and humans [11] p. 7].
- an enterprise architecture is concerned with how an enterprise’s software systems support the business processes and goals of the enterprise [11] p. 8].

To sum up, software is only one concern of system architecture and enterprise architecture.

In spite of being different architecture disciplines, software architecture and system architecture share their support to software systems. Inside the context of enterprise architecture, they can be seen as solution architectures. The Open Group Architecture Framework (TOGAF) defines a solution architecture as “a description of a discrete and focused business operation or activity and how IS/IT supports that operation. A Solution Architecture typically applies to a single project or project release, assisting in the translation of requirements into a solution vision, high-level business and/or IT system specifications, and a portfolio of implementation tasks” [35]. Poort et al. [61] also use the term solution architecture to group various architecture “genres” with the common denominator of finding a solution to a particular set of stakeholders’ needs. Such common denominator is shared by software architecture and system architecture, so we can consider that both of them are solution architectures.

2.1.3 Where Software Reference Architecture belongs to

As defined by Bass et al. [10], reference models and reference architectures are different concepts.

“A reference model is a division of functionality together with data flow between the pieces. A reference model is a standard decomposition of a
2.1. Architecture Disciplines Basic Concepts

Figure 2.1: The relationships of reference models, architectural patterns, reference architectures, and software architectures [10].

known problem into parts that cooperatively solve the problem”. They arise in mature domains in which experience has lead to a standard solution for the problem, e.g., the standards parts of a compiler or a database management system and how such parts work together to accomplish their collective purpose.

On the other hand, a reference architecture is “a reference model mapped onto software elements (that cooperatively implement the functionality defined in the reference model) and the data flows between them. Whereas a reference model divides the functionality, a reference architecture is the mapping of that functionality onto a system decomposition”.

The relationship among reference models, reference architectures, and (concrete) software architectures are shown in Figure 2.1. Summarizing, “a reference architecture is a set of domain concepts mapped onto a standard set of software components and relationships” [19].

A Reference Architecture provides a prescriptive way (a template solution) for an architecture for a particular domain [5] [71]. Reference Architectures can be found in the aforementioned architecture disciplines, leading to:

- Software reference architecture, such as RACE [34], which addresses the engine software for document processing systems.
- System reference architecture, such as a distributed system reference architecture for Adaptive QoS and Resource Management [68].
- Enterprise reference architecture, such as PERA [70], which is a complete enterprise reference architecture as defined by the IFAC/IFIP Task Force on Enterprise Integration.

This thesis proposal focuses on software reference architectures, which are inside the software architecture field of research.

2.1.4 Summary

Figure 2.2 shows the relationship between software architecture, system architecture and enterprise architecture. Although all of them are different
architecture disciplines, they are interconnected, e.g., “the software architect for a system should be on the team that provides input into the decisions made about the system or the enterprise” [11, p. 7]. For example, an enterprise could adopt an enterprise architecture as a strategic activity, but needs also to have a solution architecture (i.e., system architecture and/or software architecture) that deepens in the structure of each system. This scenario is depicted in Figure 2.3. Another example is that the software architecture of a system needs to be in compliance with the system architecture (e.g., software systems' technologies need to be compatible with the hardware architecture).

To sum up, software reference architectures are attractive when enterprises have many software systems that have very similar structure and share a technological or business domain. Then, a software reference architecture common to such software systems can be designed. Such software reference architecture defines a standard structure of systems.
Figure 2.3: Coexistence of reference architectures from different architecture disciplines: enterprise, system and software (adapted from [24]).

2.2 Software Reference Architecture Essentials

This section focuses on software reference architecture and studies:

• the definitions of software reference architecture,
• the industrial context of software reference architectures,
• the types of software reference architecture,
• the elements of software reference architecture,
• the boundaries of software reference architectures with respect to concrete software architecture,
• and the boundaries of software reference architectures with respect to product line architecture.

2.2.1 Definition of Software Reference Architecture

Nakagawa et al. [53] define a software reference architecture as “an architecture that encompasses the knowledge about how to design concrete architectures of systems of a given application [or technological] domain; therefore, it must address the business rules, architectural styles (sometimes also defined as architectural patterns that address quality attributes in the reference architecture), best practices of software development (for instance, architectural decisions, domain constraints, legislation, and standards), and the software elements that support development of systems for that domain. All of this must be supported by a unified, unambiguous, and widely understood domain terminology”.
2.2.2 The Industrial Context of Software Reference Architectures

Software reference architectures mainly appear in organizations where the multiplicity of software systems (i.e., systems developed at multiple locations, by multiple vendors and across multiple organizations) triggers a need for life-cycle support for all systems [22]. Hence, software reference architectures are very attractive when organizations become large and distributed [51] in order to develop new systems or new versions of software systems. According to their expected benefits, software reference architectures have become widely studied and used in software architecture research and practice [2][54]. Nowadays, many software reference architecture vendors (such as everis) and acquisition organizations are involved in software reference architecture projects.

We are interested in the case in which an acquisition organization (or client organization) has adopted a software reference architecture from a development organization (or vendor) with the purpose of deriving concrete software architectures for each software system (or application). This usually happens when a vendor organization (e.g., IT consulting firm or a software development company) is regularly contracted to create or maintain information systems in that acquisition organizations. Each information system is built upon the software reference architecture and includes many software systems (see Fig. 2.4).

A software reference architecture can be designed with an intended scope of a single organization or multiple organizations that share a certain property. Although Fig. 2.4 shows software reference architectures that are used for the design of concrete software architectures in a single organization, there also exist software reference architectures for multiple organizations that share a market or technological domain such as web applications [2].
The use of software reference architectures allows development organizations to reuse the architectural knowledge of their reference model, and software components (normally associated to particular technologies) for the design of concrete software architectures in client organizations. Thus, software reference architectures inherit best practices from previous successful experiences and a certain level of quality. These software reference architectures provide a baseline that facilitates standardization and interoperability as well as the attainment of business goals during enterprise applications’ development and maintenance.

Types of projects

There are three types of projects with different targets (Fig. 2.5): 1) Reference model (RM) projects; 2) Software reference architecture (SRA) projects; and 3) Concrete software architecture (SA) projects. Each of these projects has a team formed by several stakeholders. Here, we focus on key stakeholders for software reference architecture analysis. Stakeholders need to be clearly defined for software reference architecture assessment purposes [3]. The people involved in a software reference architecture assessment are the evaluation team, which conducts the empirical studies, and stakeholders from architecture projects. In the three types of projects defined above performed by software reference architecture vendors (e.g., IT consulting firms), we consider the following five stakeholders essential for software reference architecture assessment: project business manager, project technological manager, software architect, developer, and application builder. Each of these stakeholders has a vested interest in different architectural aspects, which are important to analyze and reason about the appropriateness and the quality of the three types of projects [29]. However, there could be more people involved in an architectural evaluation, as Clements et al. indicate in [19]. As a consequence, although this context is generic for IT consulting firms, projects stakeholders may vary between firms. Below, we describe to which type of project essential stakeholders belong and their interests.

Reference model projects It is composed of software architects from the development organization (e.g., IT consulting firm) that worked in previous successful software reference architecture projects. They are specialized in architectural knowledge management. Their goal is to gather the best practices from previous software reference architecture projects. experiences in order to design and/or improve the corporate RM.

Software reference architecture projects Software reference architecture projects involve people from the development organization and likely from the client organization. Their members (project technological managers, soft-
Figure 2.5: Relevant stakeholders for software reference architecture analysis [44].

Software architects and architecture developers) are specialized in architectural design and have a medium knowledge of the organization business domain.

Project technological managers from the development organization are responsible for meeting schedule and interface with the project business managers from the client organization.

Software architects (also called as software reference architecture managers) usually come from the development organization, although it may happen that the client organization has software architects in which organization’s managers rely on. In the latter case, software architects from both sides cooperatively work to figure out a solution to accomplish the desired quality attributes and architecturally-significant requirements.

Architecture developers come from the development organization and are responsible for coding, maintaining, integrating, testing and documenting software reference architecture software components.
Concrete software architecture projects  Enterprise application projects can involve people from the client organization and/or subcontracted development organizations (which may even be different than the reference model owner) whose members are usually very familiar with the specific organization domain. The participation of the client organization in software reference architecture and concrete software architecture projects is one possible strategy for ensuring the continuity of their information systems without having much dependency on subcontracted IT consulting firm.

Project business managers (i.e., customer) come from client organizations. They have the power to speak authoritatively for the project, and to manage resources. Their aim is to provide their organization with useful applications that meet the market expectations on time.

Application builders take the software reference architecture reusable components and instantiate them to build an application.

2.2.3 Types of Software Reference Architecture

Angelov et al. [2] distinguish between five types of software reference architectures. They define a multi-dimensional space to classify these types of software reference architectures, which is composed of 3 dimensions (context, goal and design) that include 8 sub-dimensions in total.

- Context dimension (C)
  - C1: Where will it be used? Values: single organization, multiple organizations.
  - C2: Who defines it? Values: software groups, user groups, and independent groups.
  - C3: When is it defined? Values: preliminary, classical.

- Goal dimension (G)
  - G1: Why is it defined? Values: standardization, facilitation.

- Design sub-dimensions (D)
  - D1: What is described? Values: components and connectors, interfaces, protocols, algorithms, policies and guidelines.
  - D2: How detailed is it described? Values: detailed, semi-detailed, and aggregated.
  - D4: How is it represented? Values: informal, semi-formal, formal.
The values of a software reference architecture are mutually exclusive for the G1, C1, and C3 sub-dimension (i.e., a reference architecture can be attributed only one value from these sub dimensions). It leads to five types of software reference architectures [2]:

- Type 1) Classical, standardization architectures to be implemented in multiple organizations;
- Type 2) Classical, standardization architectures to be implemented in a single organization;
- Type 3) Classical, facilitation reference architectures for multiple organizations designed by a software organization in cooperation with user organizations;
- Type 4) Classical, facilitation architectures designed to be implemented in a single organization;
- Type 5) Preliminary, facilitation architectures designed to be implemented in multiple organizations.

2.2.4 Which are the elements that compose a Reference Architecture

Nakagawa et al. present in [54] RAModel. RAModel is a reference model for reference architectures that shows possibly all elements, organized by types and relationships, which could be contained in reference architecture. We think that this reference model is applicable in a high degree to software and systems reference architectures. RAModel is depicted in Figure 2.6. As Figure 2.6 shows, these elements are inside one of the following four groups:

- Domain: It contains elements related to self-contained, specific information of the space of human action in the real world, such as domain legislations, standards, and certification processes, which impact systems and related reference architectures of that domain;
- Application: It contains elements that provide a good understanding about the reference architecture, its capabilities and limitations. It also contains elements related to the business rules (or functionalities) that could be present in software systems built from the reference architecture;
- Infrastructure: It refers to elements that could be used to build the software systems based on the reference architecture. These elements are responsible to enable these systems to automate, for instance, processes, activities, and tasks of a given domain; and
2.2. Software Reference Architecture Essentials

Figure 2.6: RAModel: Reference model for reference architectures [54].

- Crosscutting Elements: It aggregates a set of elements that are usually spread across and/or tangled with elements of other three groups (domain, application, and infrastructure). We have observed that communication (that we have identified as internal and external) in the software systems built from the reference architecture, as well as the domain terminology and decisions are present in a spread and tangled way when describing other groups and are, therefore, crosscutting elements.

On the other hand, Cloutier et al. vision of reference architecture is more aligned with enterprise reference architecture, since it also includes mission, vision and strategy of the organization [22] (see Figure 2.7).

2.2.5 Concrete Software Architecture and Software Reference Architecture

There are many definitions of software architecture. We show below three of the most cited ones. The Software Engineering Institute keeps an up-to-date list of software architecture’s definitions at [36].
Figure 2.7: Summary of the role of reference architectures [22].

“The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both” [11].

“The fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution” [67].

“An architecture is the set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaborations among those elements, the composition of these structural and behavioral elements into progressively larger subsystems, and the architectural style that guides this organization—these elements and their interfaces, their collaborations, and their composition” [39].

The main difference between a software reference architecture and a software architecture is that the former one is much more generic for many software systems of a domain. The above definitions highlight the architecture of “a” [11] [67] [39] system whereas an reference architecture “encompasses the knowledge about how to design concrete architectures of systems” [53]. Angelov et al. point out [2] “the generic nature of reference architectures as a main feature distinguishing them from concrete architectures. Their generic nature implies their applicability in multiple, different contexts, reflecting the requirements of the stakeholders in these contexts. The generic nature of reference architectures is achieved by designing them at higher levels of abstraction (abstracting from the differences introduced by the contexts). Thus,
we can label an architecture as reference, only if it is defined to abstract from certain contextual specifics allowing its usage in differing contexts” [2]. Also, Galster et al. show that reference architectures “capture the essence of the architecture of similar systems in an application or technology domain” and “can be instantiated for different contexts and at the same time support a high degree of variability in instantiated architectures” [31].

2.2.6 Product Line Architectures and Software Reference Architectures

The terms software reference architecture and software product line architecture are sometimes used indistinctly inside the software product line engineering context, in which the term software reference architecture is used to refer to “a core architecture that captures the high-level design for the applications of the SPL” [60] p. 124] or “just one asset, albeit an important one, in the software product line’s asset base” [21] p. 12].

However, out of the software product line context, software reference architecture and product line architecture are considered different types of artifacts [2] [24] [31] [53]. In Fig. 1 we show the main similarities and differences:

• Product line architectures are software reference architectures whereas not all software reference architectures are product line architectures [2], i.e. product line architectures are one type of software reference architectures [31]. Product line architectures are just one asset of SPL [21] p. 12].

• Software reference architectures are more generic and abstract than product line architectures that are more complete architectures [2] [31]. Hence, "software reference architectures can be designed with an intended scope of a single organization or multiple organizations that share a certain property" [2] whereas product line architectures are produced for a single organization [31].

• Software reference architectures provide standardized solutions for a broader domain (i.e., "spectrum of systems in a technology or application domain" [31]) whereas product line architectures provide standardized solutions for a smaller subset of the software systems of a domain [53] (i.e., "group of systems that are part of a product line" [31]). Therefore, product line architectures give a coherent and more congruent view of the products in a project (i.e., possible to track the status of) [24] whereas by means of software reference architectures it is more difficult to obtain congruence [2], since they can only provide guidelines for applications’ development.
• Product line architectures specifically address points of variability and more formal specification in order to ensure clear and precise behavior specifications at well-specified extension points [2]. In contrast, software reference architectures have less focus on capturing variation points [2] [24] [53]. Although variability is not typically addressed by software reference architectures in a systematic manner, it is also a key fact for software reference architectures [30], and it can be treated as a quality attribute, rather than explicitly as 'features' and 'decisions' [30].

• Software reference architectures include "the reuse of knowledge about software development in a given domain, in particular with regard to architectural design" [53] and dictate the patterns and principles to implement, i.e. "what the design should be" [24]. Conversely, product line architectures specifically indicate deviations, i.e. "what the design is" [24].

• Software reference architectures include architectural knowledge and the instantiation of this architectural knowledge (i.e., reference model) into software elements [10]. In this sense, both software reference architectures and product line architectures are "a superset, a tool box, with every possible architecture element described, which can be used in the design of a product architecture" [24].

Figure 2.8: Similarities and differences between software reference architecture, product line architectures, and software product lines [45].
Chapter 3

State-of-the-art

This chapter summarizes the current state-of-the-art in software reference architectures. On the one hand, the state-of-the-art about economic models for software reference architectures is analyzed. On the other hand, we study which empirical evidence about software reference architecture needs to be investigated. Finally, the last section describes the existing gaps in software reference architecture research that justify further work.

3.1 State-of-the-art of Software Reference Architecture Economics

This section analyzes the state-of-the-art about economic models for software reference architectures.

3.1.1 Business Case Analysis and Return-On-Investment

Reifer defines a business case as the “materials prepared for decisions makers to show that the idea being considered is a good one and that the numbers that surround it make financial sense” [63]. That is, business cases enable to justify investments in technology. For instance, spending in the adoption of a software reference architecture without a previous and trustworthy analysis seems to be reckless and can lead to a disaster. There are more types of analysis techniques. Reifer has identified the following types of analysis techniques for business case [63]:

- Breakeven Analysis. Analysis performed to compute the value at which the solution will recover expenditures when comparing alternative use of resources.
- Cause-and-Effect Analysis. Analysis to explore solutions to problems.
- Cost/Benefit Analysis. Analysis performed to compute the net benefits (can be plus or minus) resulting from an investment decision.
Value Chain Analysis. Analysis to evaluate alternatives and assess the impact of each option using a form of decision tree.

Investment Opportunity Analysis. Analysis to assess the attractiveness of a range of alternatives. Example of financial measures are return on capital, after-tax rate of return.

Pareto Analysis. Analysis based on the premise that most effects are generated from relatively few causes (sometimes called the 80-20 rule).

Payback Analysis. Analysis to calculate the amount of time required to recover the costs of the initial investment.

Sensitivity Analysis. Analysis conducted to determine to which of the input parameters the solution is sensitive.

Trend Analysis. Statistical procedure used for estimating the mathematical relationship of the cost and benefit.

The most common form of business case analysis is cost-benefit analysis. It involves determining the relative financial costs, benefits, and ROI across a system’s life-cycle as [15]:

\[
ROI = \frac{Benefits - Costs}{Costs}
\]

Three additional factors may be important in business case analysis: the time value of money, unquantifiable benefits, and uncertainties and risk.

First, money has value that increases over time due to inflation, i.e., today’s money is less worth tomorrow. To calculate such value increase, present value is used to enable decision makers to view future investments in terms of what money is worth today [63].

Second, there are benefits that may be difficult to quantify. These benefits should also been taken into account when making a business case. To cope with them, there are two possibilities: determine an estimated quantity.

Third, to adjust cost and benefits to risk, they can be multiplied by percentages that generally increase the costs and reduce the benefits (assuming the worst case). For instance, TEI [27] propose to multiple costs by values that range from 98% to 150% and benefits by values between 50% and 110%.

3.1.2 Economic Models for Software Reference Architectures

Current research on software reference architecture evaluation consists of analysis methods [3] [29] [34] that involve the analysis of risks, non-risks, benefits and trade-offs. Although they facilitate the analysis of those aspects based on the most important and critical scenarios, they have little support to analyze the cost and benefits of software reference architectures based on economics.
Introducing an software reference architecture into an organization involves making a decision of a greater degree than only considering the aforementioned aspects, since it should not only include quality, but it should also include productivity issues. Whereas architectural quality is usually estimated in relation to eliciting implicit and explicit requirements of the different stakeholders affected by the development of the system, productivity is actually measured in terms of effort, cost, and economic benefits. Nevertheless, both views are necessary to achieve a comprehensive analysis of the system.

Up to our knowledge, there is no specific economic model for estimating whether it is worth or not to invest in a software reference architecture for an organization. Due to the lack of research in this specific area, we have aimed at adopting and adapting existing results in related areas: economic models for software product lines (SPL), cost-benefit analysis methods for software architectures, and more generic metrics about cost savings.

**Economic models for software product lines and software reuse**

Although we also consider that software reference architectures and product line architectures are different (see Section 2.2.6), some perceived benefits of software reference architecture (e.g., cost avoidance from reusing software elements) and cost-benefit factors (e.g., common software costs, unique development costs) are applicable to both, since both have reuse as their core strategy. For this reason, we studied the applicability of some economic models originally conceived for SPL to software reference architectures. Below, we summarize our results with respect to cost and benefit factors. To see more models, the reader is referred to [1], in which Ali et al. surveyed twelve economic models for SPL, and to [28] [50] in which the authors surveyed economic models for software reuse.

**Cost and Benefit Factors of Economic Models.** SIMPLE [20], Poulin’s model [62], and COPLIMO [14] are some of the most widespread economic models for SPLs. SIMPLE [20] comprises a set of seven cost factors:

- \( C_{\text{org}} \), upfront investments to establish a SPL infrastructure.
- \( C_{\text{cabr}} \), the cost to build reusable assets of the SPL.
- \( C_{\text{unique}} \), the cost to develop unique parts of products in a SPL.
- \( C_{\text{reuse}} \), the cost of reusing reusable assets in a product inside the SPL.
- \( C_{\text{cabu}} \), the cost to evolve the core asset in a SPL.
- \( C_{\text{prod}} \), the cost to build a product in a stand-alone fashion.
- \( C_{\text{evo}} \), the cost to evolve a product in a stand-alone fashion.
These cost factors and benefit functions can be used to construct equations that can answer a number of questions such as whether the SPL approach is the best option for development and what is the ROI for this approach. Ganesan et al. extended SIMPLE by considering infrastructure degeneration over time [32].

On the other hand, Poulin [62] and Boehm et al. [14] base their reuse-based models in two parameters: RCR and RCWR.

• RCR (Relative Cost of Reuse). Assuming that the cost to develop a reusable asset equals one unit of effort, RCR is the portion of this effort that it takes to reuse a reusable asset without modification (black-box reuse).

• RCWR (Relative Cost of Writing for Reuse). Assuming that the cost to develop a new asset for one-time use equals one unit of effort, RCWR is the portion of this effort that it takes to write a similar "reusable" asset.

For those cases in which there are difficulties to obtain historical data of building and evolving products in a stand-alone fashion ($C_{\text{prod}}$, $C_{\text{evo}}$), we consider more adequate the use of RCR and RCWR (see Section 4.1, step 2).

Finally, we must note two models (Schmid [64], InCoME [55]) that integrate cost and investment models in different layers, which make them more comprehensive.

Value of software architecture design decisions

There exist a few economics-based software architecture analysis methods that drive the decision-making process during software architecture review and design. In this direction, CBAM [37] is a useful method for prioritizing architectural decisions that bring higher value. In addition, Ozkaya et al. proposed an economic valuation of architectural patterns [58].

These approaches help to find the optimal set of decisions that maximizes the ROI [26]. They pursue to solve the same problem of this paper, but their scope is broader and general for any kind of software architecture decision and do not reflect fundamental characteristics of adopting a software reference architecture. Therefore, their applicability for studying the ROI of software reference architecture adoption would require more effort, since specific cost-benefit factors for architecture-centric reuse are not considered. Hence, they are not the most convenient approaches for making the business case of adopting a software reference architecture and calculating its payback time.

Generic software metrics

There exist several approaches that propose metrics for estimating cost savings in software development and maintenance. Metrics as dependency structure matrices (DSM) have been applied to assist architecture-level analysis,
such as value of modular designs, and they have proven to be particularly insightful for validating the future value of architecting modular systems [16]. Mac-Cormack et al. extracted coupling metrics from an architecture DSM view for inferring the likelihood of change propagation and, hence, future maintenance costs [11]. Baldwin et al. presented a generic expression for evaluating the option to redesign a module also based on DSMs [9]. In addition, the concept of technical debt (either architecture-focused [56] or code-based [40]) is a way to measure unexpected rework costs due to expediting the delivery of stakeholder value in short.

3.1.3 Summary

Although there is a lack of research in evaluating the economic viability of software reference architecture adoption, there is a strong base of research in related areas. The most important related area is economic models that identify cost and benefit factors for product line architecture adoption. Although there is a significant amount of research in this direction, it falls short in:

- Validation in industry. “Very few [economic models for SPL] actually have been used as a basis for further development or adopted in industry” [23]. Thus, “there is a clear need for many more empirical studies to validate existing models” [1].

- Easy adoption of models in industry by identifying realistic metrics to collect and report. “It is difficult for the practitioners to evaluate the usability and usefulness of a proposed solution [economic model for SPL] for application in industry” [38]. Not guidelines exist to fully operationalize the models in practice [64].

Economics-driven software architecture analysis methods do not specifically aim at making an investment analysis of the adoption of an architecture-centric program. Software reference architecture adoption is a sub area inside their generic decision-making context.

At a lower level, more simple metrics like DSM, could also be adequate for calculation the cost and benefit factors of software reference architecture adoption and make more complete models.

3.2 State-of-the-art of Evaluation Methods for Software Reference Architecture

This section studies which empirical evidence about software reference architecture needs to be investigated. To do so, it firstly presents evaluation methods to review software architectures and software reference architectures; and, then, it studies the relevant aspects used to assess software reference architectures, about which it is needed to extract evidence.
3.2.1 Why is it important to review software reference architectures?

As Clement et al. point out [19], the software architecture of a system is important because it allows or precludes nearly all of the system’s quality attributes. Software architecture is also a vehicle for communication among stakeholders, the manifestation of the earliest design decisions, and a reusable abstraction of a system.

Software reference architecture serves as a baseline for deriving the software architecture of systems of a domain. For instance, an exemplar software reference architecture for the information system domain details in which consists of the software architecture of systems (i.e., a database, business logic, and a user interface mapped onto a three-tier client-server structure), and provides software components to support systems’ software architecture.

A software reference architecture can be applied to many software systems exhibiting similar requirements (i.e., systems that share a domain) and can promote large-scale reuse. Once built, a software reference “architecture and the components that populate it can be one of an organization’s key assets for many years. An artifact of such far-ranging, long-lasting importance deserves to be evaluated to make sure it will serve its organization as intended” [19].

A software reference architecture is the result of early design decisions that are necessary before a group of people can collaboratively build the software architecture of a software system. Clement et al. indicate that the most fundamental truth about architecture evaluation is: “if architectural decisions determine a system’s quality attributes, then it is possible to evaluate architectural decisions with respect to their impact on those attributes” [19].

Evaluations are a risk-mitigation effort. In other words, “architecture evaluation is a cheap way to avoid disaster” [19]. Their aim is to analyze software architectures to identify potential risks and to verify that the quality requirements have been addressed in the design [23].

Evaluation Methods

Table 3.1 shows a summary of well-known evaluation methods of software architectures, and Table 3.2 shows how these methods have been extended to evaluate software reference architectures.

Current evaluation methods for software reference architectures extend from well-known evaluation methods for software architecture. As Table 3.2 indicates, Angelov’s method [3] extends from ATAM, Graaf’s method [34] extends from SAAM and Gallager’s method [29] extends from ATAM.

3.2.2 Relevant Aspects to Assess Software Reference Architectures

Prior to analysis and review software reference architecture, it becomes necessary to previously identify relevant aspects to assess software reference

Table 3.1: Evaluation Methods for Software Architecture.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Evaluation Criteria</th>
<th>Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATAM</td>
<td>Quality attributes (e.g., modifiability, security, reliability and performance)</td>
<td>[19]</td>
</tr>
<tr>
<td>SAAM</td>
<td>Modifiability and functionality</td>
<td>[19]</td>
</tr>
<tr>
<td>ARID</td>
<td>Suitability of design approach</td>
<td>[19]</td>
</tr>
<tr>
<td>TARA</td>
<td>Requirements</td>
<td>[23]</td>
</tr>
<tr>
<td>LAAAM</td>
<td>Design options</td>
<td>[17]</td>
</tr>
<tr>
<td>Scenario-based peer review</td>
<td>Risks associated with a single scenario</td>
<td>[8]</td>
</tr>
</tbody>
</table>

Table 3.2: Evaluation Methods for Software Reference Architecture.

<table>
<thead>
<tr>
<th>Authors</th>
<th>Based on</th>
<th>Evaluation Criteria</th>
<th>Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>S. Angelov et al.</td>
<td>ATAM</td>
<td>Quality attributes (extended with specific in reference architecture, e.g., applicability)</td>
<td>[3]</td>
</tr>
<tr>
<td>B. Graaf et al.</td>
<td>SAAM</td>
<td>Modifiability</td>
<td>[34]</td>
</tr>
<tr>
<td>B.P. Gallagher</td>
<td>ATAM</td>
<td>Quality attributes (e.g., modifiability, security, reliability and performance)</td>
<td>[29]</td>
</tr>
</tbody>
</table>

architectures. However, a commonly accepted set of criteria to assess software reference architectures does not exist [3] [29] [30] [34]. Thus, in this section we identify important aspects to assess software reference architectures out of practice and out of the literature. These aspects should be seen as a primary input for their further refinement based on the evidence from organizations (throughout this thesis).

In [3], Angelov et al. state that software architectures and software reference architectures have to be assessed for the same aspects. For this reason, we started by analyzing some available works on software architecture assessment [12] [25]. However, existing evaluation methods for software architectures are not directly applicable to software reference architectures because they do not cover the generic nature of software reference architectures [3]. Therefore, we elaborated further this analysis considering both the specific characteristics of software reference architectures as described in [3] [29] [30] [53] and our own experience in the field. The resulting aspects for assessing software reference architecture are detailed below and summarized in Table 3.3.

Aspect 1 refers to the need of having an overview of the software reference architecture. It includes an analysis of its generic functionalities, its domain [3], its origin and motivation, its correctness and utility, and its sup-
port for efficient adaptation and instantiation [30]. Since software reference architectures are defined to abstract from certain contextual specifics allowing its usage in differing contexts [2], their support for efficient adaptation and instantiation while deriving concrete software architectures is an aspect to assess [30].

Many prominent researchers [3] [19] [25] [29] highlight the importance of quality attributes, as well as architectural decisions for the software architecture design process and the architectural assessment. These two aspects should also be considered for the software reference architecture assessment because, as we said, software architectures and software reference architectures have to be assessed for the same aspects [3]. Thus, we considered them as Aspects 2 and 3 respectively. However, since an software reference architecture has to address more architectural qualities than an software architecture (e.g., applicability) [3], this analysis could be wider for software reference architectures in this sense. A list of quality attributes that are strictly determined by software architectures is defined in [19]. This list consists of the following ten quality attributes: performance, reliability, availability, security, modifiability, portability, functionality, variability, subsetability and conceptual integrity.

Software architectures also address business qualities [3] (e.g., cost, time-to-market) that are business goals that affect their competence [12]. It is considered as Aspect 4.

To improve the software architecture design process, there also exist supportive technologies such as methods, techniques and tools [25] [53]. Thus, it is not only important for an software reference architecture to collect data to assess its design process, but also its supportive technologies, which are assessed by Aspects 5 and 6.

As stated in [25], a crucial aspect to define the goodness of a software architecture is related to the ROI. The optimal set of architectural decisions is usually the one that maximizes the ROI. Aspect 7 is intended to quantify benefits and costs of software reference architectures to calculate their ROI.

These architectural aspects can be divided in two areas of different nature. First, Aspects 1 to 6 are qualitative architectural concerns. Second, Aspect 7 consists of quantitative metrics to calculate the benefits and costs of deriving software architectures from software reference architectures.

### 3.2.3 Summary

We recommend gathering evidence about all these aspects, which are summarized in Table 3.3 while assessing an software reference architecture. Existing methods for software architecture assessment have been previously applied for software reference architecture assessment, such as in [3] [29] [34]. However, none of them cover all the aspects of Table 3.3, especially Aspect 7.
### 3.3 Conclusions

Table 3.3: Summary of relevant aspects for software reference architecture assessment.

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Description of the Architectural Aspect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Overview: functionalities, origin, utility and adaptation</td>
</tr>
<tr>
<td>2</td>
<td>Requirements and quality attributes analysis</td>
</tr>
<tr>
<td>3</td>
<td>Architectural knowledge and decisions</td>
</tr>
<tr>
<td>4</td>
<td>Business qualities and architecture competence</td>
</tr>
<tr>
<td>5</td>
<td>Software development methodology</td>
</tr>
<tr>
<td>6</td>
<td>Technologies and tools</td>
</tr>
<tr>
<td>7</td>
<td>Benefits and cost metrics to derive software architectures from software reference architectures</td>
</tr>
</tbody>
</table>

Hence, new approaches to assess software reference architectures considering these aspects altogether are required. This has motivated our work.

**3.3 Conclusions**

The state-of-the-art summarized in Section 3.1.3 drove us to **the formulation of an economic model for software reference architectures**. The formulation of the model aims to:

- Adapt cost and benefit factors from SPL models that are easy-to-apply by industry. The goal is to provide guidelines to fully operationalize the model in practice.

- Fill the gap of software reference architecture economics inside the software architecture decision-making context.

- Look for generic software metrics that can quantify new cost and benefit factors.

On the other hand, in spite of the existence of evaluation methods for software reference architectures, “the software engineering community rarely adopts the methods and techniques available to support disciplined architecture review processes” [7]. We posit two possible reasons for this:

- As Ali Babar et al. point out, we think that “there remains a need for systematically accumulating and widely disseminating evidence about the factors that may influence the selection and use of different methods, techniques, and tools for architecture evaluation” [6].
Evaluation teams need to have the vision from all stakeholders (e.g., project managers, software architects, developers, etc.). When they do not have such vision, they experience problems while conducting architectural reviews. Each of these stakeholders has a vested interest in different architectural aspects, which are important to analyze and reason about the appropriateness and the quality of the reference architecture [29].

In order to overcome these two issues, we propose conducting empirical studies to accumulate real evidence about the relevant aspects for evaluating software reference architectures from essential type of stakeholders. Such empirical studies should gather data about all software reference architecture relevant aspects (see them in Section 3.2.2). The gathering of all these relevant aspects from essential stakeholders is novel regarding the current state-of-the-art summarized in Section 3.2.3.
Chapter 4

Thesis Proposal

This chapter presents the main goal and objectives of the PhD thesis, the research methodology that is and will be used in the development of the thesis, together with the research plan for the work that remains to be done. Finally, it presents bibliographic information about the work that has been published.

4.1 Goal

This section exposes the goal of this thesis:

To support organizations to decide on the adoption of software reference architectures and its subsequent suitability for the organization purposes.

This research goal is divided into two Research Questions (RQ):

- **RQ 1: Is it worth to invest on the adoption of a software reference architecture?**

  The objective of the RQ 1 is to provide guidelines to support organizations to quantitatively analyze if it is worth to adopt a software reference architecture. Such an objective consists of constructing an economic model for software reference architectures that enables to make business case for financial analysis. This analysis optimize the decision-making process when studying whether to make the strategic move to software reference architecture in an organization.

  Likewise, this research question is divided into four sub-research questions:
– RQ 1.1: Under which context are software reference architectures used in practice?
– RQ 1.2: Which commonly available data do organizations have to quantitatively calculate the costs and benefits of adopting a software reference architecture in an organization?
– RQ 1.3: Which are the cost and benefit factors of acquiring a software reference architecture in an organization?
– RQ 1.4: How is it possible to calculate the return-on-invest of the adoption of a software reference architecture in an organization?

• RQ 2: Once adopted, how the suitability of a software reference architecture for deriving concrete software architectures for an organization’s software systems can be ensured?

The objective of the RQ 2 is to provide guidelines to support practitioners to qualitatively review the suitability of a software reference architecture for deriving concrete software architectures for an organization’s software systems. This general objective consists of gathering, increasing and disseminating empirical evidence about relevant aspects of software reference architectures. This objective is twofold: to help researchers to assess current research and identify promising future research areas, and practitioners to choose appropriate methods and techniques for supporting the software reference architecture design and evaluation.

Specifically, this research question is divided into five sub-research questions:

– RQ 2.1: How different stakeholders perceive the potential benefits and drawbacks of software reference architectures?
– RQ 2.2: Which are the elements that compose a software reference architecture in the industrial practice and what is their potential reuse across domains?
– RQ 2.3: Which are the quality attributes that a software reference architecture enforce?
– RQ 2.4: How are architectural decisions taken and documented in real software reference architecture projects?
– RQ 2.5: Which supportive technologies (i.e., methodologies, tools) are currently being used in real software reference architecture projects?

The Section 4.1.1 and Section 4.1.2 describe these two research questions respectively.
4.1. Goal

4.1.1 RQ 1: Is it worth to invest on the adoption of a software reference architecture?

The creation and maintenance of complex software systems involves making a series of business-critical architecture design decisions [49]. Imagine that you are the CIO of an organization with a wide portfolio of software systems. You have read about the expected benefits than a software reference architecture may bring to the organization, e.g., standardization of concrete software architecture of systems, greater reuse, shorter time-to-market, reduced costs, reduced risk, support for system development at multiple locations, by multiple vendors and across multiple organizations, and so on. Therefore, you are considering to adopt a software reference architecture to create and maintain your organization’s software systems. However, how do you know if it is worth to invest on the adoption of a software reference architecture? How many software systems are necessary before savings pay off for the up-front investment in building a software reference architecture? Which is the return-on-investment (ROI) of a software reference architecture? These questions could be answered by making a business case with the help of an economic model for software reference architectures.

Reifer defines a business case as the “materials prepared for decision makers to show that the idea being considered is a good one and that the numbers that surround it make financial sense” [63]. That is, business cases enable to justify investments in technology. Spending in the adoption of a software reference architecture without a previous and trustworthy analysis seems to be reckless and can lead to a disaster.

In the software reference architecture context, an economic model is needed to help making business cases. An economic model should take into account costs, benefits, risks, and schedule implications. An economic model to perform cost-benefit analysis on the adoption of software reference architecture is a key asset for optimizing architectural decision-making.

4.1.2 RQ 2: Once adopted, how the suitability of a software reference architecture for deriving concrete software architectures for an organization’s software systems can be ensured?

Introducing a software reference architecture into an organization not only involves making a decision considering the aforementioned productivity issues, but also involves the analysis of risks, non-risks, benefits and trade-offs. Whereas productivity is actually measured in terms of effort/cost and economic benefits, architectural quality is usually estimated in relation to eliciting implicit and explicit requirements of the different stakeholders affected by the development of the system. Nevertheless, both views are necessary to achieve a comprehensive analysis of the system.
The software architecture of a software system is an early result of the development cycle. Therefore, evaluating every aspect of software architecture in advance (e.g., identifying improvements that can dramatically improve any system’s performance, security, reliability, and maintainability) implies a remarkably low cost, since by then the implementation of the system has not been started [19]. As a result, the practice of evaluating software architectures has matured, with well-known evaluation methods such as ATAM [19], TARA [73], LAAAM [17] and scenario-based peer reviews [8]. Besides, these methods have been investigated and extended to evaluate reference architectures, such as already presented in [3] [29] [34]. However, there is a lack of recent research that proposes means to evaluate reference architectures [52].

In spite of the existence of evaluation methods, “the software engineering community rarely adopts the methods and techniques available to support disciplined architecture review processes” [7]. To overcome it, we propose conducting empirical studies to accumulate real evidence about the relevant aspects for evaluating software reference architectures from key stakeholders.

4.2 Research Methodology

Empirical research is a way of gaining knowledge by means of direct and indirect observation or experience [69]. One of the objectives of Empirical Software Engineering is to gather and utilize evidence to advance software engineering methods, processes, techniques, and tools [25].

Figure 4.1 shows the relationships among software architecture theory, empirical theory, empirical assessments, challenges, lessons learned, and empirical results [25]. “When researchers attempt to empirically assess software architecture theory, they face challenges from both empirical theory and software architecture theory. Empirical theory provides the means to gather and disseminate evidence in order to support the claims of efficiency or efficacy of a particular technology. Software architecture theory provides the hypothesis which will be accepted or rejected. Empirical research can provide the results on which to build and/or assess the theoretical foundations underpinning various software architecture-related technologies. Experiences and lessons learned from empirically assessing software architecture research represent a valuable -but often underestimated- means of improving the application of the empirical paradigm to software architecture research and practice” [25].

This thesis proposal fosters the conduction of empirical studies as a way to build up software reference architecture theory. Therefore, we instantiated this methodology for the sub-area of software reference architecture inside software architecture field of research. Figure 4.1 shows such instantiation. The next section describes the challenges and expected contribution of this thesis: A Framework for Software Reference Architecture Analysis and Review.
4.3 Contribution: A Framework for Analysis and Review of Software Reference Architectures

To accomplish the goal of this research, we plan to devise a framework by providing procedural guidelines for setting up and carrying out empirical
studies. Such a framework has two challenges, presented in Table 4.1. In order to achieve these challenges, the conduction of empirical studies pursues two main expected results (second column of Table 4.1).

Table 4.1: Research Questions’ Challenges and Expected Results.

<table>
<thead>
<tr>
<th>RQ</th>
<th>Challenge</th>
<th>Expected Result and Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Calculate the ROI of software reference architecture adoption for an organization.</td>
<td>An economic model for software reference architectures. Its purpose is to support business case to make financial analysis that optimize the decision-making process when studying whether to make the strategic move to software reference architecture or not.</td>
</tr>
<tr>
<td>2</td>
<td>Gain knowledge about relevant aspects of software reference architectures.</td>
<td>Gathering and disseminate empirical evidence about relevant aspects of software reference architectures. It helps researchers to assess current research and identify promising future research areas, and practitioners to choose appropriate methods and techniques for supporting the software reference architecture adoption.</td>
</tr>
</tbody>
</table>

The framework, aimed to analyze and review software reference architectures, is composed of an assortment of empirical studies. These empirical studies are classified by two dimensions:

- The step in which the framework is being applied. Wohlin et al. state that “it is in most cases impossible to start improving directly” [72]. The framework has three steps: understand, evaluate and improve. The framework deals with the two former steps, in which surveys are inside the understanding step, and models and methods in the evaluation step. The improving step is achieved by iteratively applying the evaluation studies and considering the lessons learned.

- The Research Question that the study approaches. Therefore, there are two possible values: RQ 1 (quantitative study) and RQ 2 (qualitative study).

Figure 4.2 describes the dimensions and the studies that compose the framework. The rows indicate the step in which the framework is being applied whereas the columns show the research question that the study approaches.
As shown by Figure 4.2, the framework is composed of four studies, two studies to cope with quantitative analysis (see details in Table 4.2) and two studies to cope with qualitative analysis (see details in Table 4.3):

- **RQ 1 supported by two studies:**
  - A *Survey to check existing value-driven data in organizations*. It aims to provide support or guidelines to check existing value-driven data in the organization in order to perform a quantitative evaluation.
  - An *Economic model to calculate the ROI of adopting an RA* (quantitative). It aims to provide an economic model to calculate the return-on-investment of adopting a software reference architecture.

- **RQ 2 supported by two studies:**
  - A *Survey to understand the impact of using a software reference architecture*. It aims to provide support or guidelines to understand the impact of using a software reference architecture in the organization in order to perform a qualitative evaluation.
  - An *Architectural evaluation method specific for software reference architecture*. The above survey helps to provide support for the selection of an architectural evaluation method for software reference architecture and easy its conduction with information from key stakeholders. Currently, there exist evaluation methods. Therefore, the framework integrates existing evaluation methods.

The studies are complementary and support each other (e.g., results from a preceding study can be used to corroborate or develop further these results). For this reason, the suggested studies should be conducted sequentially.
Table 4.2: Studies of the framework for the RQ 1.

<table>
<thead>
<tr>
<th>Study of the Framework</th>
<th>Context</th>
<th>Objective</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surveys to check existing value-driven data in software reference architecture projects</td>
<td>Typically, organizations do not have resources to compare the real cost of creating applications with and without a software reference architecture. Thus, alternatives should be envisaged.</td>
<td>To discover existing data that organizations have to quantitatively calculate the costs and benefits of adopting a software reference architecture in an organization.</td>
<td>Exploratory surveys with personalized questionnaires applied to relevant stakeholders (e.g., manager, architect, developer) to find out the quantitative data that has been collected in software reference architecture projects and application projects.</td>
</tr>
<tr>
<td>Applying an economic model to calculate the ROI of adopting a software reference architecture</td>
<td>Before deciding to launch a software reference architecture, organizations need to analyze whether undertaking or not the investment. Offering organizations an economic model that is based on former projects data can help them to make more informed decisions.</td>
<td>To assess whether it is worth investing in a software reference architecture.</td>
<td>An economic model that quantifies the potential advantages and limitations of using a software reference architecture. Some related works explain how to calculate the ROI of a product [27], and software reuse [62]. We suggest using the economic model for software reference architectures presented in [45].</td>
</tr>
</tbody>
</table>
Table 4.3: Studies of the framework for the RQ 2.

<table>
<thead>
<tr>
<th>Study of the Framework</th>
<th>Context</th>
<th>Objective</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surveys to understand the impact of using a software reference architecture</td>
<td>To refine the set of review criteria for software reference architectures, it is needed to understand software reference architectures' characteristics, as well as its potential benefits and limitations. Assessing previous software reference architecture projects is a feasible way to start gaining such an understanding.</td>
<td>To understand the impact and suitability of a reference model for the elaboration of software reference architectures, and of a software reference architecture for the creation of concrete software architectures. Improvement insights can also be identified from different stakeholders.</td>
<td>Exploratory surveys with personalized questionnaires applied to relevant stakeholders (e.g., architects, developers) to gather their perceptions and needs.</td>
</tr>
<tr>
<td>Applying an architectural evaluation method to prove software reference architecture effectiveness</td>
<td>Architecture is the product of the early design phase [19]. Software reference architecture evaluation is a way to find potential problems before implementing software reference architecture’s modules, and to gain confidence in the software reference architecture design provided to concrete software architecture projects.</td>
<td>To analyze the software reference architecture strengths and weaknesses and to determine which improvements should be incorporated in the software reference architecture.</td>
<td>To apply an existing evaluation method specific for software reference architectures such as [3] [29] [34]. The selection of the method would depend on the organization context [13].</td>
</tr>
</tbody>
</table>
4.4 Tasks and Working Plan

We have identified 7 main tasks. Table 4.4 shows the task list, in which a Task is represented by the acronym T. Next, we explain these tasks.

**T1: State-of-the-art and state-of-the-practice of Software Reference Architectures, Economic Models and Empirical Software Engineering.** T1 consists of a start-of-the-art on the topic of the thesis: software reference architectures and empirical methods. This state-of-the-art also includes the state-of-the-practice, from the feedback in our action-research in eversis and other sources usually consulted by practitioners, such as market research companies (e.g., Gartner and Forrester).

This state-of-the-art and state-of-the-practice is not a definitive result, but is going to be updated throughout the thesis. Indeed, the *Initial stage of the state-of-the-art and state-of-the-practice* (T1.1) only considered software reference architectures and empirical software engineering. Throughout the development of the thesis, it becomes necessary to *Maintain an up-to-date state-of-the-art and state-of-the practice* (T1.2). This incremental task does not only consider new work, but can also includes new topics as the research advances. For this reason, we added the study of economic models.

With this work we got two results: the identification of relevant aspects for evaluating software reference architectures and selecting appropriate empirical methods to understand and evaluate software reference architectures. These results are useful in order to fulfill the next task, the design of a framework for software reference architectures analysis and review. A initial version of this work is described in a technical report [42].

**T2: A Framework to Software Reference Architectures Analysis and Review.** T2 consists of the construction of a framework that supports organizations to cope with the research questions previously stated in Section 4.1

*T2.1: Definition of the Framework.* This task consists of gluing the empirical methods designed to understand and evaluate software reference architectures. The result is a complete framework composed of an assortment of studies that help organizations to understand, evaluate and improve their software reference architectures. This framework (published in [44]) is not a definitive result, and is going to be incrementally updated throughout the thesis. Next tasks show how to deal with this iterative evolution of the framework.

*T2.2: Formative stage of the Framework.* T3, T4 and T5 represent the studies that compose the framework. As their conduction advances, their feedback will contribute to incrementally design the framework. For this reason, T3, T4 and T5 are best characterized as formative studies due to their central role in shaping the framework.
Table 4.4: Task list.

<table>
<thead>
<tr>
<th>Task</th>
<th>Sub-Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T1.1</td>
<td>Initial stage of the state-of-the-art and state-of-the-practice.</td>
</tr>
<tr>
<td></td>
<td>T1.2</td>
<td>Maintain an up-to-date state-of-the-art and state-of-the-practice.</td>
</tr>
<tr>
<td>T2</td>
<td></td>
<td>A Framework to Software Reference Architectures Analysis and Review</td>
</tr>
<tr>
<td></td>
<td>T2.1</td>
<td>Definition of the Framework.</td>
</tr>
<tr>
<td></td>
<td>T2.2</td>
<td>Formative stage of the Framework.</td>
</tr>
<tr>
<td></td>
<td>T2.3</td>
<td>Summative stage of the Framework.</td>
</tr>
<tr>
<td>T3</td>
<td></td>
<td>Survey to check existing value-driven data in organizations.</td>
</tr>
<tr>
<td></td>
<td>T3.1</td>
<td>Definition, Design and Implementation of the Survey.</td>
</tr>
<tr>
<td></td>
<td>T3.2</td>
<td>Execution of the Survey (at everis).</td>
</tr>
<tr>
<td></td>
<td>T3.3</td>
<td>Analysis and Packaging of the Survey.</td>
</tr>
<tr>
<td>T4</td>
<td></td>
<td>An Economic Model to Calculate the Return-on-Investment of adopting a Software Reference Architecture (REARM)</td>
</tr>
<tr>
<td></td>
<td>T4.1</td>
<td>Definition of REARM</td>
</tr>
<tr>
<td></td>
<td>T4.2</td>
<td>Application of REARM in Organization X.</td>
</tr>
<tr>
<td></td>
<td>T4.3</td>
<td>Update of REARM with feedback from T4.2</td>
</tr>
<tr>
<td></td>
<td>T4.4</td>
<td>Application of REARM in Organization Y.</td>
</tr>
<tr>
<td></td>
<td>T4.5</td>
<td>Update of REARM with feedback from T4.4.</td>
</tr>
<tr>
<td></td>
<td>T4.6</td>
<td>Update of REARM with feedback from T5.</td>
</tr>
<tr>
<td>T5</td>
<td></td>
<td>Survey to understand the impact of using a software reference architecture</td>
</tr>
<tr>
<td></td>
<td>T5.1</td>
<td>Definition, Design and Implementation of the Survey.</td>
</tr>
<tr>
<td></td>
<td>T5.2</td>
<td>Execution of the Survey (at everis).</td>
</tr>
<tr>
<td></td>
<td>T5.3</td>
<td>Analysis and Packaging of Survey’s Group of Questions about Benefits and Drawbacks.</td>
</tr>
<tr>
<td></td>
<td>T5.4</td>
<td>Analysis and Packaging of Survey’s Group of Questions about Elements and Guidelines.</td>
</tr>
<tr>
<td></td>
<td>T5.5</td>
<td>Analysis and Packaging of Survey’s Group of Questions about Requirements.</td>
</tr>
<tr>
<td></td>
<td>T5.6</td>
<td>Analysis and Packaging of Survey’s Group of Questions about Architectural Decisions.</td>
</tr>
<tr>
<td></td>
<td>T5.7</td>
<td>Analysis and Packaging of Survey’s Group of Questions about Methodologies and Technologies.</td>
</tr>
<tr>
<td>T6</td>
<td></td>
<td>Write the thesis.</td>
</tr>
<tr>
<td>T7</td>
<td></td>
<td>Design the presentation of the thesis.</td>
</tr>
</tbody>
</table>
T2.3: Summative stage of the Framework. Once that the previous task (T2.2) will be finished, it will be necessary to validate the results got and to analyze lessons learned. This task is best characterized as summative, since its primary role in this thesis is to develop a final version of the framework from the output of the formative studies.

T3: Survey to check existing value-driven data in organizations. This task represents one study of the framework, inside the understanding step and that approaches the RQ 1. It aims to discover existing data that organizations have to quantitatively calculate the costs and benefits of adopting a software reference architecture.

As defined in [18], there are several steps along the conduction of a survey: definition, design, implementation, execution, analysis and packaging. For this reason, this task has three sub-tasks: Definition, Design and Implementation of the Survey (T3.1); Execution of the Survey at everis (T3.2); and Analysis and Packaging of the Survey (T3.3).

Results from this survey are published in [44].

T4: An Economic Model to Calculate the Return-on-Investment of adopting a Software Reference Architecture (REARM). This task represents one study of the framework, inside the evaluation step and that approaches the RQ 1. It aims to assess whether it is worth or not investing in a software reference architecture.

The first sub-task of T4 is the Definition of REARM (T4.1). REARM, whose acronym comes from REference ARchitecture Model, is an economic model to quantitatively analyze the adoption of software reference architectures in organizations. A preliminary version of REARM was published in a technical report [46].

We plan to use REARM in two client organizations of everis that respectively have adopted and will adopt a software reference architecture. These tasks are: Application of REARM in Organization X (T4.2) and Application of REARM in Organization Y (T4.4). With the feedback and lessons learned from such applications of REARM in real projects, we plan to incrementally update it (if necessary) in tasks T4.3 (Update of REARM with feedback from T4.2) and T4.5 (Update of REARM with feedback from T4.4). The first application of REARM has already been done and published in [45].

Finally, throughout the thesis, REARM also benefits from results in qualitative surveys that help to understand cost and benefit factors. This task is T4.6 (Update of REARM with feedback from T5).

T5: Survey to understand the impact of using a software reference architecture. This task represents one study of the framework, inside the understanding step and that approaches the RQ 2. It aims to understand the impact
4.4. Tasks and Working Plan

and suitability of a reference model for the elaboration of software reference architectures, and of a software reference architecture for the creation of concrete software architectures. Improvement insights can also be identified from different stakeholders.

As we also did in T3, we have created sub-tasks to manage the complexity of a long task as the conduction of a survey.

The first two tasks are: Definition, Design and Implementation of the Survey (T5.1) and Execution of the Survey (T5.2). These two tasks have already been done. We worked on nine projects in which everis defined and implemented a software reference architecture for different organizations. We conducted face-to-face interviews with the software architects for each project. In addition, there were two on-line surveys for the architecture developers and application builders. In total twenty-eight people participated in the study. Currently, we are analyzing and packaging the data collected.

For analysis and packaging of the survey, we decided to create five sub-tasks, since we conducted long face-to-face interviews (with average duration of one hour and a half) dealing with several concepts and also long on-line questionnaires (with more than 50 questions). Moreover, the survey include three different types of stakeholders with personalized questions. The division of analysis and packaging activities has been done by topic (i.e., group of questions). We plan to conduct it in five phases, that have lead to the next sub-tasks: Analysis and Packaging of Survey’s Group of Questions about Benefits and Drawbacks (T5.3), Analysis and Packaging of Survey’s Group of Questions about Elements and Guidelines (T5.4), Analysis and Packaging of Survey’s Group of Questions about Requirements (T5.5), Analysis and Packaging of Survey’s Group of Questions about Architectural Decisions (T5.6), and Analysis and Packaging of Survey’s Group of Questions about Methodologies and Technologies (T5.7). Preliminary results about the T5.3 has been published in [43].

T6: Write the thesis. In this task, we are going to write the documentation of the thesis.

T7: Design the presentation of the thesis. This task includes the time between handing the documentation in and the defense of the thesis.

Besides these tasks, other tasks related with the thesis (such as stays on foreign universities and attendance to conferences, workshops, seminars, courses and so on) have been and will be also done.
4.4.1 Relationship of Tasks with Research Questions

The tasks presented above, have been carefully envisaged to fulfill the research questions defined in Section 4.1.

$T_1$, $T_6$ and $T_7$ are the habitual tasks on every thesis (state-of-the-art, writing the thesis and preparation of defense), whereas $T_2$, $T_3$, $T_4$ and $T_5$ stem from the stated goal and research questions of the thesis.

Specifically, Table 4.5 shows how the research questions are covered by the latter group of tasks ($T_2$, $T_3$, $T_4$ and $T_5$).

Table 4.5: Relationship of Tasks with Research Questions.

<table>
<thead>
<tr>
<th>Task</th>
<th>RQ</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>None</td>
<td>General task on every thesis.</td>
</tr>
<tr>
<td>T2</td>
<td>RQ 1</td>
<td>The framework copes with both research questions by an assortment of studies.</td>
</tr>
<tr>
<td>T3</td>
<td>RQ 1</td>
<td>Study of the framework that is inside the understanding step and that approaches the $RQ_1$. Specifically, it covers the following sub research questions: $RQ_{1.1}$, $RQ_{1.2}$.</td>
</tr>
<tr>
<td>T4</td>
<td>RQ 1</td>
<td>Study of the framework that is inside the evaluation step and that approaches the $RQ_1$. Specifically, it covers the following sub research questions: $RQ_{1.3}$, $RQ_{1.4}$.</td>
</tr>
<tr>
<td>T5</td>
<td>RQ 2</td>
<td>Study of the framework that is inside the understanding step and that approaches the $RQ_2$. Each sub-task of the analysis and packaging of data, namely $T5.3$, $T5.4$, $T5.5$, $T5.6$, and $T5.7$, respectively covers one the following sub research questions: $RQ_{2.1}$, $RQ_{2.2}$, $RQ_{2.3}$, $RQ_{2.4}$, and $RQ_{2.5}$.</td>
</tr>
<tr>
<td>T6</td>
<td>None</td>
<td>General task on every thesis.</td>
</tr>
<tr>
<td>T7</td>
<td>None</td>
<td>General task on every thesis.</td>
</tr>
</tbody>
</table>
4.4. Tasks and Working Plan

4.4.2 Working Plan

The PhD thesis has a full duration of 4 years, starting October 18th 2011. Figure 4.3 shows a timeline with the general schedule for the tasks defined in Table 4.4. Figure 4.4 shows Gantt diagram with more details that include sub-tasks.

Figure 4.3: Timeline.

![Timeline Diagram]

Figure 4.4: Gantt Diagram.

![Gantt Diagram]
4.5 Publications

This section enumerates and summarizes the work that has already been published. For a better representation of the work done, the following subsections show these publications grouped by topic.

1. Publications with both approaches (qualitative and quantitative) that present the Framework for Software Reference Architecture Analysis and Review.

2. Publications that deal with the RQ 1 and present REARM, our Software Reference Architecture Economic Model.

3. Publications that deal with the RQ 2 and present reports about qualitative empirical studies.

4. Out of the Cátedra UPC-everis project, we have studied in the predictive service domain. The results have been a reference model and its application for the weather forecast domain.

Besides the publication of research papers, I am going to use my personal page of UPC, [http://www.essi.upc.edu/~smartinez/](http://www.essi.upc.edu/~smartinez/) to disseminate the work as it progresses.
4.5.1 Framework for Software Reference Architecture

<table>
<thead>
<tr>
<th>Title</th>
<th>A Framework for Software Reference Architecture Analysis and Review [44]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authors</td>
<td>Silverio Martinez-Fernández, Claudia Ayala, Xavier Franch, Helena Martins, David Ameller</td>
</tr>
<tr>
<td>Published in</td>
<td>In Proceedings 10th Workshop on Experimental Software Engineering (ESELAW), Montevideo (Uruguay), 10th April, 2013</td>
</tr>
<tr>
<td>Year</td>
<td>2013</td>
</tr>
<tr>
<td>Involved Thesis Tasks</td>
<td>T1.1, T2.1, T2.2, T3, T5.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Title</th>
<th>Conducting Empirical Studies on Reference Architectures in IT Consulting Firms [42]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authors</td>
<td>Silverio Martinez-Fernández, David Ameller, Claudia Ayala, Xavier Franch and Xavier Terradellas</td>
</tr>
<tr>
<td>Published in</td>
<td>Technical Report ESSI-TR-12-2, April 2012</td>
</tr>
<tr>
<td>Year</td>
<td>2012</td>
</tr>
<tr>
<td>Involved Thesis Tasks</td>
<td>T1.1, T2.1</td>
</tr>
</tbody>
</table>
### 4.5.2 Software Reference Architecture Economics

<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th><em>REARM: A Reuse-Based Economic Model for Software Reference Architectures</em> [45]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Authors</strong></td>
<td>Silverio Martínez-Fernández, Claudia Ayala, Xavier Franch, Helena Martins</td>
</tr>
<tr>
<td><strong>Published</strong></td>
<td>In Proceedings 13th International Conference on Software Reuse (ICSR), Pisa (Italy), 18th-21th June, 2013</td>
</tr>
<tr>
<td><strong>Year</strong></td>
<td>2013</td>
</tr>
<tr>
<td><strong>Involved Thesis Tasks</strong></td>
<td>T1.2, T4.1, T4.2, T4.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th><em>A Reuse-Based Economic Model for Software Reference Architectures</em> [46]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Authors</strong></td>
<td>Silverio Martínez-Fernández, Claudia Ayala, and Xavier Franch</td>
</tr>
<tr>
<td><strong>Published in</strong></td>
<td>Technical Report ESSI-TR-12-6, November 2012.</td>
</tr>
<tr>
<td><strong>Year</strong></td>
<td>2012</td>
</tr>
<tr>
<td><strong>Involved Thesis Tasks</strong></td>
<td>T4.1, T4.2</td>
</tr>
</tbody>
</table>
### Empirical Research on Software Reference Architecture

<table>
<thead>
<tr>
<th>Title</th>
<th>Benefits and Drawbacks of Reference Architectures [43]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authors</td>
<td>Silverio Martínez-Fernández, Claudia Ayala, Xavier Franch, Helena Martins</td>
</tr>
<tr>
<td>Published in Year</td>
<td>In Proceedings 7th European Conference on Software Architecture (ECSA), Montpellier (France), 1st-5th July, 2013.</td>
</tr>
<tr>
<td>Involved Thesis Tasks</td>
<td>T1.2, T5.3</td>
</tr>
</tbody>
</table>
### Application of a Reference Model for Predictive Services Selection

<table>
<thead>
<tr>
<th>Title</th>
<th>Verifying Predictive Services’ Quality with Mercury</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authors</td>
<td>Silverio Martínez-Fernández, Xavier Franch, Jesús Bisbal.</td>
</tr>
<tr>
<td>Published</td>
<td>In Proceedings 4th International Workshop on Academic Software Development Tools and Techniques (WASDeTT), Montpellier (France), 1st July, 2013.</td>
</tr>
<tr>
<td>Year</td>
<td>2013</td>
</tr>
<tr>
<td>Involved Thesis Tasks</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Title</th>
<th>QuPreSS: A Service-Oriented Framework for Predictive Services Quality Assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authors</td>
<td>Silverio Martínez-Fernández, Jesús Bisbal and Xavier Franch.</td>
</tr>
<tr>
<td>Published</td>
<td>In Proceedings 7th International Knowledge Management, Services and Cloud Computing Conference (KMO), Salamanca (Spain), 12th July, 2012.</td>
</tr>
<tr>
<td>Year</td>
<td>2012</td>
</tr>
<tr>
<td>Involved Thesis Tasks</td>
<td>-</td>
</tr>
</tbody>
</table>
References


Appendix A

Glossary

**Enterprise architecture**  Enterprise architecture (EA) is a discipline for proactively and holistically leading enterprise responses to disruptive forces by identifying and analyzing the execution of change toward desired business vision and outcomes. EA delivers value by presenting business and IT leaders with signature-ready recommendations for adjusting policies and projects to achieve target business outcomes that capitalize on relevant business disruptions. EA is used to steer decision making toward the evolution of the future state architecture [33].

**Reference architecture**  A Reference Architecture provides a prescriptive way (a template solution) for an architecture for a particular domain [5]. A reference architecture is a set of domain concepts mapped onto a standard set of software components and relationships [19].

**Return-on-Invest (ROI)**  Measure of how much profit an investment earns computed by dividing net income by the assets used to generate it [63].

**Software architecture**  The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both [11, p. 4].

**Software reference architecture**  An architecture that encompasses the knowledge about how to design concrete architectures of systems of a given application [or technological] domain; therefore, it must address the business rules, architectural styles (sometimes also defined as architectural patterns that address quality attributes in the reference architecture), best practices of software development (for instance, architectural decisions, domain constraints, legislation, and standards), and the software elements that support development of systems for that domain. All of this must be supported by a unified, unambiguous, and widely understood domain terminology [53].
System architecture  A system’s architecture is a representation of a system in which there is a mapping of functionality onto hardware and software components, a mapping of the software architecture onto the hardware architecture, and a concern for the human interaction with these components [11] p. 7].
A Framework for Software Reference Architecture
Analysis and Review

Silverio Martínez-Fernández¹, Claudia Ayala¹, Xavier Franch¹, Helena Martins Marques², and David Ameller¹

¹GESSI Research Group, Universitat Politècnica de Catalunya, Barcelona, Spain
{smartinez,cayala,franch,dameller}@essi.upc.edu
²everis, Barcelona, Spain
hmartinm@everis.com

Abstract. Tight time-to-market needs pushes software companies and IT consulting firms to continuously look for techniques to improve their IT services in general, and the design of software architectures in particular. The use of software reference architectures allows IT consulting firms reusing architectural knowledge and components in a systematic way. In return, IT consulting firms face the need to analyze the return on investment in software reference architectures for organizations, and to review these reference architectures in order to ensure their quality and incremental improvement. Little support exists to help IT consulting firms to face these challenges. In this paper we present an empirical framework aimed to support the analysis and review of software reference architectures and their use in IT projects by harvesting relevant evidence from the wide spectrum of involved stakeholders. Such a framework comes from an action research approach held in everis, an IT consulting firm. We report the issues found so far.

Keywords: Software architecture, reference architecture, architecture analysis, architecture evaluation, empirical software engineering.

1 Introduction

Nowadays, the size and complexity of information systems, together with critical time-to-market needs, demand new software engineering approaches to design software architectures (SA) [17]. One of these approaches is the use of software reference architectures (RA) that allows to systematically reuse knowledge and components when developing a concrete SA [8][13].

As defined by Bass et al. [3], a reference model (RM) is “a division of functionality together with data flow between the pieces” and an RA is “a reference model mapped onto software elements (that cooperatively implement the functionality defined in the reference model) and the data flows between them”.

A more detailed definition of RAs is given by Nakagawa et al. [17]. They define an RA as “an architecture that encompasses the knowledge about how to design concrete architectures of systems of a given application [or technological] domain; therefore, it
must address the business rules, architectural styles (sometimes also defined as architectural patterns that address quality attributes in the reference architecture), best practices of software development (for instance, architectural decisions, domain constraints, legislation, and standards), and the software elements that support development of systems for that domain. All of this must be supported by a unified, unambiguous, and widely understood domain terminology”.

In this paper, we use these two RA definitions. We show the relationships among RM, RM-based RA and RA-based concrete SA in Fig. 1. Throughout the paper, we use the term RA to refer to RM-based RA and SA to refer to RA-based concrete SA. Angelov et al. have identified the generic nature of RAs as the main feature that distinguishing them from concrete SAs. Every application has its own and unique SA, which is derived from an RA. This is possible because RAs are abstract enough to allow its usage in differing contexts. [2]

![Fig. 1. Relationships among RM, RA and SA.](image)

**Research problem.** The motivations behind RAs are: to facilitate reuse, and thereby harvest potential savings through reduced cycle times, cost, risk and increased quality [8]; to help with the evolution of a set of systems that stem from the same RA [13]; and to ensure standardization and interoperability [2]. Due to this, RAs are becoming a key asset of organizations [8].

However, although the adoption of an RA might have plenty of benefits for an organization, it also implies several challenges, such as the need for an initial investment [13] and ensuring its adequacy for the organization’s portfolio of applications. Hence, in order to use RAs, software companies and information technology consulting firms face two fundamental questions:

1) Is it worth to invest on the adoption of an RA?
2) Once adopted, how the suitability of an RA for deriving concrete SAs for an organization’s applications can be ensured?
Currently, organizations lack of support for dealing with these questions. On the one hand, there is a shortage of economic models to precisely evaluate the benefit of architecture projects in order to take informed decisions about adopting an RA in an organization. On the other hand, although there are qualitative evaluation methods for RAs, they do not systematize how these RAs should be evaluated regarding certain quality attributes (for instance, their capability to satisfy the variability in applications developed from RAs).

In this context, the goal of this research is to devise a framework that supports organizations to deal with the aforementioned questions by providing procedural guidelines for setting up and carrying out empirical studies aimed to extract evidence for: 1) supporting organizations to assess if it is worth to adopt an RA, and 2) ensuring the suitability of an RA for deriving concrete SAs for an organization’s applications.

It is worth mentioning that this research has its origin in an ongoing action-research initiative among our research group and everis, a multinational consulting company based in Spain. The architecture group of everis faced the fundamental questions stated above and the framework proposed in this paper was mainly originated and shaped throughout our involvement for helping everis to envisage a suitable solution. The idea behind devising such a framework is twofold: to help other organizations dealing with similar problems as everis; and to improve the guidelines of the framework by the experience gained in each application of the framework in order to consolidate architectural knowledge from the industrial practice.

The paper is structured as follows. In Section 2 we describe the fundamental aspects of RAs that are suggested to be assessed. In Section 3 we describe the empirical studies that compose the framework. In Section 4 we present the context of IT consulting firms and show how the framework can be applied in the context of an IT consulting firm. In Sections 5 and 6 we present preliminary results of two studies of the framework applied in everis. In Section 7 we end up with conclusions and future work.

2 Practical Review Criteria for Reference Architectures

In order to devise the framework for RA analysis and review, it becomes necessary to previously identify relevant aspects to assess RAs. However, a commonly accepted set of criteria to assess RAs does not exist. Thus, in this section we identify important aspects to assess RAs out of practice and out of the literature. The framework presented in this paper envisions these aspects as a primary input for their further refinement based on the evidence from organizations.

In [1], Angelov et al. state that SAs and RAs have to be assessed for the same aspects. For this reason, we started by analyzing some available works on SA assessment. However, existing evaluation methods for SAs are not directly applicable to RAs because they do not cover the generic nature of RAs. Therefore, we elaborated further this analysis considering both the specific characteristics of RAs as described in [1][12-13][17] and our own experience in the field. The resulting aspects for assessing RA are detailed below and summarized in Table 1.
Aspect 1 refers to the need of having an overview of the RA. It includes an analysis of its generic functionalities, its domain [1], its origin and motivation, its correctness and utility, and its support for efficient adaptation and instantiation [13]. Since RAs are defined to abstract from certain contextual specifics allowing its usage in differing contexts [2], their support for efficient adaptation and instantiation while deriving concrete SAs is an aspect to assess [13].

Many prominent researchers [1][7][10][12] highlight the importance of quality attributes, as well as architectural decisions for the SA design process and the architectural assessment. These two aspects should also be considered for the RA assessment because, as we said, SAs and RAs have to be assessed for the same aspects [1]. Thus, we considered them as Aspects 2 and 3 respectively. However, since an RA has to address more architectural qualities than an SA (e.g., applicability) [1], this analysis could be wider for RAs in this sense. A list of quality attributes that are strictly determined by SAs is defined in [7]. This list consists of the following ten quality attributes: performance, reliability, availability, security, modifiability, portability, functionality, variability, subsetability and conceptual integrity.

SAs also address business qualities [1] (e.g., cost, time-to-market) that are business goals that affect their competence [4]. It is considered as Aspect 4.

To improve the SA design process, there also exist supportive technologies such as methods, techniques and tools [10][17]. Thus, it is not only important for an RA to collect data to assess its design process, but also its supportive technologies, which are assessed by Aspects 5 and 6.

As stated in [10], a crucial aspect to define the goodness of a SA is related to the ROI. The optimal set of architectural decisions is usually the one that maximizes the ROI. Aspect 7 is intended to quantify benefits and costs of RAs to calculate their ROI.

We recommend gathering evidence about all these aspects, which are summarized in Table 1, while assessing an RA. Existing methods for SA assessment have been previously applied for RA assessment, such as in [1][12][14]. However, none of them cover all the aspects of Table 1, especially Aspect 7. Hence, new approaches to assess RAs considering these aspects altogether are required. This has motivated our work.

These architectural aspects can be divided in two areas of different nature. First, Aspects 1 to 6 are qualitative architectural concerns. Second, Aspect 7 consists of quantitative metrics to calculate the benefits and costs of deriving SAs from RAs.

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Description of the Architectural Aspect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qualitative 1</td>
<td>Overview: functionalities [1], origin, utility and adaptation [13]</td>
</tr>
<tr>
<td>Qualitative 2</td>
<td>Requirements and quality attributes analysis [1][10][12]</td>
</tr>
<tr>
<td>Qualitative 3</td>
<td>Architectural knowledge and decisions [10][12][17]</td>
</tr>
<tr>
<td>Qualitative 4</td>
<td>Business qualities [1] and architecture competence [4]</td>
</tr>
<tr>
<td>Qualitative 5</td>
<td>Software development methodology [10][17]</td>
</tr>
<tr>
<td>Qualitative 6</td>
<td>Technologies and tools [10][17]</td>
</tr>
<tr>
<td>Quantitative 7</td>
<td>Benefits and costs metrics to derive SAs from RAs [10]</td>
</tr>
</tbody>
</table>
3 An Empirical Framework to Review Reference Architectures

In this section, we present the ongoing version of our empirical framework. It is composed of an assortment of empirical studies. Each empirical study reviews a subset of the relevant architectural aspects presented in Table 1.

Current economic models [16] and RAs evaluation methods (e.g., [1][12-14]) suggest to gather Aspect 7, and Aspect 1-6 respectively, directly from the organizations. However, they do not provide support or guidelines for doing so. Thus, our framework is aimed to provide support for such gathering process while suggests to apply any of the existing methods to evaluate the economic costs of adopting an RA or its quality based on the empirically obtained data. The selection of the method used in each situation would depend on the organization context [5].

Regarding to Aspect 7, an economic model becomes necessary. The data needed to feed such an economic model depends on the existing value-driven data in the organization (see Sections 3.1 and 5). Such data may be gathered by conducting post-mortem studies that collect real metrics or, when the organization does not have previous experience with RAs, by estimating these metrics using historical data.

In order to gather data to cover Aspects 1-6, our framework suggests conducting surveys (see Sections 3.3 and 6). These studies gather information not only from RA projects, but also from SA projects as they are a direct outcome of the RA usage. This allows analyzing the RA’s suitability for producing the SAs of enterprise applications in organizations as well as detecting improvement opportunities.

Fig. 2 summarizes the studies that compose the framework. The studies are classified by their approach to assess the RA (qualitative or quantitative), depending on which question of Section 1 they answer.

Moreover, the studies have been designed by following the guidelines for empirical studies of Wohlin et al. [21]. They state that “it is in most cases impossible to start improving directly”. Therefore, the framework is based on three steps: understand, evaluate and improve. The current version of the framework deals with the two former steps, in which surveys are inside the understanding step, and models and methods in the evaluation step. Thus, studies are complementary and support each other (e.g., results from a preceding study can be used to corroborate or develop further these results). For this reason, the suggested studies should be conducted sequentially.

Fig. 2. Empirical studies of the framework to assess RAs.
3.1 Surveys to check existing value-driven data in RA projects

**Context.** Typically, organizations do not have resources to compare the real cost of creating applications with and without an RA. Thus, alternatives should be envisaged.

**Objective.** To discover existing data that organizations have to quantitatively calculate the costs and benefits of adopting an RA in an organization.

**Method.** Exploratory surveys with personalized questionnaires applied to relevant stakeholders (e.g., manager, architect, developer) to find out the quantitative data that has been collected in RA projects and application projects.

3.2 Applying an economic model to calculate the ROI of adopting an RA

**Context.** Before deciding to launch an RA, organizations need to analyze whether undertaking or not the investment. Offering organizations an economic model that is based on former projects data can help them to make more informed decisions.

**Objective.** To assess whether it is worth investing in an RA.

**Method.** Depending on the maturity of the organization, two methodologies can be applied. If the organization does not have an RA, the economic model should be fed with estimated data. Nevertheless, when the organization already has an RA, real data can be gathered by means of an exploratory quantitative post-mortem analysis. Then, the economic model quantifies the potential advantages and limitations of using an RA. Some related works explain how to calculate the ROI of a product [11], and software reuse [19]. We suggest using the economic model for RAs presented in [14].

3.3 Surveys to understand the impact of using an RA

**Context.** To refine the set of review criteria for RAs, it is needed to understand RA’s characteristics, as well as its potential benefits and limitations. Assessing previous RA projects is a feasible way to start gaining such an understanding.

**Objective.** To understand the impact and suitability of an RM for the elaboration of RAs, and of an RA for the creation of SAs. Improvement insights can also be identified from different stakeholders.

**Method.** Exploratory surveys with personalized questionnaires applied to relevant stakeholders (e.g., architects, developers) to gather their perceptions and needs.

3.4 Applying an architectural evaluation method to prove RA effectiveness

**Context.** Architecture is the product of the early design phase [7]. RA evaluation is a way to find potential problems before implementing RA modules, and to gain confidence in the RA design provided to SA projects.

**Objective.** To analyze the RA strengths and weaknesses and to determine which improvements should be incorporated in the RA.

**Method.** To apply an existing evaluation method specific for RAs such as [1][12-14]. The selection of the method would depend on the organization context [5].
4 Use of the framework in an IT consulting firm

4.1 Context of Information Technology Consulting Firms

Motivation. We are interested in the case in which an IT consulting firm has designed an RA with the purpose of deriving concrete SAs for each application of a client organization. This usually happens when the IT consulting firm is regularly contracted to create or maintain information systems in client organizations. Each information system is built upon the RA and includes many enterprise applications (see Fig. 3).

An RA can be designed with an intended scope of a single organization or multiple organizations that share a certain property. Although Fig. 3 shows RAs that are used for the design of SAs in a single organization, there also exist RAs for multiple organizations that share a market or technological domain such as web applications [2].

The use of RAs allows IT consulting firms to reuse the architectural knowledge of their RM, and software components (normally associated to particular technologies) for the design of SAs in client organizations. Thus, RAs inherit best practices from previous successful experiences and a certain level of quality. These RAs provide a baseline that facilitates standardization and interoperability as well as the attainment of business goals during enterprise applications’ development and maintenance.

Types of projects. There are three types of projects with different targets (Fig. 3): 1) RM projects; 2) RA projects; and 3) SA projects.

Stakeholders for RA analysis. Stakeholders need to be clearly defined for RA assessment purposes [1]. The people involved in an RA assessment are the evaluation team, which conducts the empirical studies of the framework, and stakeholders from architectural projects. In the three types of projects defined above performed by IT consulting firms, we consider the following five stakeholders essential for RA assessment: project business manager, project technological manager, software architect, developer, and application builder. Each of these stakeholders has a vested interest in different architectural aspects, which are important to analyze and reason about the appropriateness and the quality of the three types of projects [12]. However, there could be more people involved in an architectural evaluation, as Clements et al. indicate in [7]. As a consequence, although this context is generic for IT consulting firms, projects’ stakeholders may vary between firms. Below, we describe to which type of project essential stakeholders belong and their interests.

RM projects. It is composed of software architects from the IT consulting firm that worked in previous successful RA projects. They are specialized in architectural knowledge management. Their goal is to gather the best practices from previous RA projects’ experiences in order to design and/or improve the corporate RM.

RA projects. RA projects involve people from the IT consulting firm and likely from the client organization. Their members (project technological managers, software architects and architecture developers) are specialized in architectural design and have a medium knowledge of the organization business domain.

Project technological managers from the IT consulting firm are responsible for meeting schedule and interface with the project business managers from the client organization.
Software architects (also called as RA managers) usually come from the IT consulting firm, although it may happen that the client organization has software architects in which organization’s managers rely on. In the latter case, software architects from both sides cooperatively work to figure out a solution to accomplish the desired quality attributes and architecturally-significant requirements.

Architecture developers come from the IT consulting firm and are responsible for coding, maintaining, integrating, testing and documenting RA software components.

**SA projects.** Enterprise application projects can involve people from the client organization and/or subcontracted IT consulting firms (which may even be different than the RM owner) whose members are usually very familiar with the specific organization domain. The participation of the client organization in RA and SA projects is one possible strategy for ensuring the continuity of their information systems without having much dependency on subcontracted IT consulting firm.

Project business managers (i.e., customer) come from client organizations. They have the power to speak authoritatively for the project, and to manage resources. Their aim is to provide their organization with useful applications that meet the market expectations on time.

Application builders take the RA reusable components and instantiate them to build an application.

---

**Fig. 3.** Relevant stakeholders for RA analysis.
4.2 Instantiation of the Framework

The presented empirical framework is currently being applied at *everis*. The main motivation of *everis* for conducting the empirical studies is twofold: 1) strategic: providing quantitative evidence to their clients about the potential benefits of applying an RA; 2) technical: identifying strengths and weaknesses of an RA.

As an IT consulting firm, *everis* fits into the context described in Section 4.1 (e.g., they carry out the three types of projects described there). Following the criteria found in [1], RAs created by *everis* can be seen as Practice RAs, since they are defined from the accumulation of practical knowledge (the architectural knowledge of their corporate RM). According to the classification of [2], they are also classical, facilitation architectures designed to be implemented in a single organization. They are classical because their creation is based on experiences, and their aim is to facilitate guidelines for the design of systems, specifically for the information system domain.

All the studies suggested in Section 3 are planned to be conducted to understand and evaluate RAs defined by *everis*. In this paper, we present the protocol and preliminary results of the two surveys of the understanding step. Section 5 describes the available value-driven data in projects in order to create or choose an economic model to calculate the ROI of adopting an RA. In Section 6, an excerpt of the survey protocol, which has already been designed and reviewed, is presented. The survey is still in the analysis step. However, the data about the Aspect 4 (business qualities) have already been processed. Preliminary results about this aspect show the benefits and aspects to consider in *everis*’ RA projects, and we think that such aspects might be similar in other IT consulting firms that adopt RAs in client organizations.

Table 2 shows how the roles are covered by the different studies in the *everis* case.

<table>
<thead>
<tr>
<th>Project</th>
<th>Business Manager</th>
<th>Technical Manager</th>
<th>Software Architect</th>
<th>Architecture Developer</th>
<th>Application Builder</th>
</tr>
</thead>
<tbody>
<tr>
<td>RM</td>
<td>n/a</td>
<td>n/a</td>
<td>S1, ROI, S2</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>RA</td>
<td>S2, Eva</td>
<td>S1, S2, Eva</td>
<td>ROI, S2, Eva</td>
<td>S2, Eva</td>
<td>S2, Eva</td>
</tr>
<tr>
<td>SA</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>S1, S2</td>
</tr>
</tbody>
</table>

a. Legend: Survey to study existing data (S1), Economic model for RA’s ROI (ROI), Survey to understand RA projects (S2), and RA evaluation (Eva).

5 Survey to check existing value-driven data in projects

5.1 Protocol

Objectives of this study. The objective of this survey is to identify the quantitative information that can be retrieved from past projects in order to perform a cost-benefit analysis. The cost-benefit analysis, which is the evaluation step of the framework, needs this kind of data to calculate the ROI of adopting an RA in an organization.

Sampling. A sample of 5 *everis*’ RA projects and 5 SA projects built upon their RAs have been selected.
Approach for data collection. The main perceived economic benefit on the use of RAs are the cost savings in the development and maintenance of systems due to the reuse of software elements and the adoption of best practices of software development that increase the productivity of developers [16]. We use online questionnaires to ask project technical managers and application builders about existing information in past projects for calculating these cost savings. When the client organization has no experience in RAs, these data need to be estimated, what could be potentially error-prone.

5.2 Preliminary results: costs and benefits metrics for RAs

In this section we describe the information that was available in order to calculate the costs and benefits of adopting an RA. We divide existing information in two categories: effort and software metrics. On the one hand, the invested effort from the tracked activities allows the calculation of the costs of the project. On the other hand, software metrics help to analyze the benefits that can be found in the source code.

**Effort metrics to calculate projects’ costs.** In RA projects, 4 out 5 client organizations tracked development efforts, while maintenance effort was tracked in 5. In SA projects, 4 client organizations tracked development and maintenance effort.

The development effort is the total amount of hours invested in the development of the RA and the SAs of applications. It could be extracted from the spent time for each development’s activity of the projects. The maintenance effort is the total amount of hours invested in the maintenance of the RA and the SAs of applications. Maintenance activities include changes, incidences, support and consults.

**Software metrics to calculate benefits in reuse and maintainability.** Code in RA and SA projects was obviously available in all projects. However, due to confidentiality issues with client organizations, it is not always allowed to access source code.

The analysis of the code from RA and SA projects allow quantifying the size of these projects in terms of LOC or function points (number of methods). Having calculated the project costs as indicated above, we can calculate the average cost of a LOC or a function point. Since the cost of applications’ development and maintenance is lower because of the reuse of RA modules, we can calculate the benefits of RA by estimating the benefits of reusing them. Poulin defines a model for measuring the benefits of software reuse [19]. Maintenance savings due to a modular design could be calculated with design structured matrices [15]. For a detailed explanation about how such metrics can be used in a cost-benefit analysis, the reader is referred to [16].

5.3 Lessons learned

Architecture improvements are extremely difficult to evaluate in an analytic and quantitative fashion as the efficacy of the business (e.g., sales) [6]. This is because software development is a naturally low-validity environment and reliable expert intuition can only be acquired in a high-validity environment [9]. In order to evaluate RAs based on an economics-driven approach, software development needs to move to a high-validity environment. The good news is that it could be done with the help of good practices like time tracking, continuous feedback, test-driven development and
continuous integration. In order to get the metrics defined in the Section 5.2, tools such as JIRA\(^1\) and Redmine\(^2\) allow managing the tasks and their invested time, general software metrics (like LOC) and percentages of tests and rules compliance can be calculated by Sonar\(^3\) and Jenkins\(^4\). We think that adopting good practices to collect data is the basis for moving software development to a high-validity environment and consequently being able of performing an accurate cost-benefit analysis.

6 Survey to understand the impact of using an RA

6.1 Protocol

Objectives of this survey. The purpose of the survey is to understand the impact of using RAs for designing the SAs of the applications of an information system of a client organization. This is a descriptive survey that measures what occurred while using RAs rather than why. The following research questions are important in order to review relevant Aspects 1 to 6 of RAs (defined in Section 2):

1. How is an RA adapted for creating SAs of an organization’s applications?
2. What is the state of practice on requirements engineering for RAs?
3. What is the state of practice on architectural design for RAs?
4. How does the adoption of RAs provide observable benefits to the different involved actors?
5. What methodologies are currently being used in RA projects by everis?
6. Which tools and technologies are currently being used in RAs projects by everis?

Sampling. The target populations of this survey are RA projects and SA projects executed by everis. A sample of 9 representative everis’ projects in client organizations were selected. All these projects were from Europe (seven from Spain).

Approach for data collection. On the one hand, semi-structured interviews are used for project technological managers, software architects, and client’s project business managers. The reason of using interviews is that these roles have higher knowledge than the other roles about the architectural aspects of the Table 1, or another perspective in the case of client’s project business managers, so we want to collect as much information as possible from them. Prior to the interviews, questionnaires are delivered to collect personal information about the interviewee and to inform him/her about the interview. On the other hand, online questionnaires are used for RA developers and application builders, since most of their questions are about supportive technologies and their responses can be previously listed, simplifying the data collection process.

This is an excerpt of the survey protocol. The complete version of the protocol is available at http://www.essi.upc.edu/~gessi/papers/eselaw13-survey-protocol.pdf.

---

2 Redmine, http://www.redmine.org/
3 Sonar, http://www.sonarsource.org/
4 Jenkins, http://jenkins-ci.org/
6.2 Preliminary results: strengths and weaknesses of RAs

In this section we present preliminary results about the business quality section of the survey, which are the answer to the fourth research question of the protocol: “How does the adoption of RAs provide observable benefits to the different involved actors?” Below, the resulting benefits and aspects to consider are reported, followed of further statements. **Benefits in everis’ RA projects are:**

- 4 out of 9 projects mentioned “increased quality of the enterprise applications”.
  - An RA helps to accomplish business needs by improving key quality attributes.
  - An RA helps to improve the business processes of an organization.
  - An RA reuses architectural knowledge of previous successful experiences.
- 7 out of 9 projects stated “reduction of the development time and faster delivery of applications”.
  - An RA allows starting developing applications since the first day by following architectural decisions already taken.
  - An RA decreases the development time of applications since the RA’s modules that implement needed functionality are reused in the application.
- 7 out of 9 projects mentioned “increased productivity of application builders”.
  - An RA facilitates material and tools for the development, testing and documentation of applications, and for training application builders.
  - An RA generates or automatizes the creation of code in the applications.
  - An RA indicates the guidelines to be followed by the application builders.
  - An RA reduces the complexity of applications’ developments because part of the functionality is already resolved in the RA.
  - An RA facilitates the configuration of its modules and the integration with legacy systems or external systems.
- 6 out of 9 projects stated “cost savings in the maintenance of the applications”.
  - An RA increases the control over applications through their homogeneity.
  - An RA maintains only once reused services by all applications.
  - An RA allows adding or changing functionalities by means of a modular design.
  - An RA establishes long term support standards and “de facto” technologies.

**Aspects to consider** that eventually could become risks in everis’ RA projects are:

- 5 out of 9 projects considered “additional learning curve”. An RA implies an additional training for their own tools and modules, even if its technologies are standard or “de facto” already known by the application builder.
- 3 out of 9 projects stated “dependency on the RA”. Applications depend on the reused modules of the RA. If it is necessary to make changes in a reused module of the RA or to add a new functionality, application builders have to wait for the RA developers to include it in the RA for all the applications.
- 2 out of 9 projects considered “limited flexibility of the applications”. The use of an RA implies following its guidelines during the application development and adopting its architectural design. If business needs require a different type of application, the RA would limit the flexibility of that application.
6.3 Lessons learned

During the pilot of the survey, we learnt the following lessons about its design:

- The same term could have slightly different meaning in the academia and in the industry (for instance, the term “enterprise architecture” is sometimes used in the industry to mean “software reference architecture for a single organization”).
- Questions that deal with several variables disconcert the interviewee and make the analysis more difficult. It is better to split them to cover only one variable.
- If a survey targets several stakeholders, their questionnaires should be designed having into account their knowledge and interest about architectural concerns.
- In online questionnaires, it is recommendable to allow the interviewee to write any comments or clarifications in some field and also include an “n/a” option when necessary. Besides, a previous button is useful to make changes in prior questions.
- Contacting stakeholders from client organizations was harder than contacting interviewees from the IT consulting firm. This is mainly because it was the IT consulting firm who requested the study, so they had a clear interest on it.

7 Conclusions and Future Work

Driving empirical studies is becoming one of the main sources of communication between practitioners and the academia. The main contribution of this ongoing work intends to be the formulation of a framework to conduct empirical studies for supporting decision making and assessment related to RAs. It consists of a list of relevant aspects for RAs assessment, and an assortment of four complementary empirical studies that allow understanding and evaluating these aspects.

It is a practical framework that can be adapted to the specific context of software companies and IT consulting firms. Consequently, organizations that apply the framework could benefit from a common reference framework to review RAs.

The framework is being applied in everis. This allows getting feedback for assessing its effectiveness and gathering industrial evidence. Preliminary results of this application indicate the importance of good practices like time tracking, continuous feedback, test-driven development and continuous integration in order to quantitatively evaluate RAs. Another result is that the adoption of an RA could bring as main benefits cost savings in the development and maintenance of applications.

Future work spreads into two directions. In terms of validation, we are also conducting the evaluation step of the framework in everis. With respect to this first version of the framework, we aim to extend it considering Wohlin’s improvement step in order to build preliminary guidelines for improving RAs in IT consulting firms.

Acknowledgements. This work has been supported by “Cátedra everis” and the Spanish project TIN2010-19130-C02-00. We would also like to thank all participants of the surveys for their kindly cooperation.
References

Conducting Empirical Studies on Reference Architectures in IT Consulting Firms

Silverio Martínez-Fernández¹, David Ameller¹, Claudia Ayala¹, Xavier Franch¹, Xavier Terradellas²

¹ Software Engineering for Information Systems Group
Universitat Politècnica de Catalunya (GESSI-UPC)
Barcelona, Spain
{smartinez, dameller, cayala, franch}@essi.upc.edu

² Architecture Centre of Excellence
Everis (ARCHEX)
Barcelona, Spain
xavier.terradellas.fernandez@everis.com

April 2012

Report ESSI-TR-12-2
Departament d’Enginyeria de Serveis i Sistemes d’Informació
Conducting Empirical Studies on Reference Architectures in IT Consulting Firms

Silverio Martínez-Fernández, David Ameller, Claudia Ayala, Xavier Franch
Software Engineering for Information Systems Group
Universitat Politècnica de Catalunya (GESSI-UPC)
Barcelona, Spain
{smartinez, dameller, cayala, franch}@essi.upc.edu

Xavier Terradellas
Architecture Centre of Excellence
Everis (ARCHEX)
Barcelona, Spain
xavier.terradellas.fernandez@everis.com

Abstract— Tight time-to-market needs pushes IT consulting firms (ITCFs) to continuously look for techniques to improve their IT services in general, and the design of software architectures in particular. The use of reference architectures allows ITCFs reusing architectural knowledge and components in a systematic way. In return, ITCFs face the need to assess these reference architectures in order to ensure their quality, return on investment and incremental improvement. Little support exists to help ITCFs to face this challenge. In this work-in-progress paper we present an empirical framework aimed to assess ITCFs’ reference architectures and their use in IT projects by harvesting relevant evidence from the wide spectrum of involved stakeholders. We are currently applying this framework in an ITCF and we report the issues found so far.

Keywords-Software architecture, reference architecture, empirical software engineering.

I. INTRODUCTION

Nowadays, the size and complexity of information systems (IS), together with critical time-to-market needs, demand new software engineering approaches to design software architectures (SA) [13]. One of these approaches is the use of reference architectures (RA) that allows to systematically reuse knowledge and components when developing a concrete SA [5][11].

As defined in [13], an RA “encompasses the knowledge about how to design concrete architectures of systems of a given application [or technological] domain; therefore, it must address the business rules, architectural styles […], best practices of software development […], and the software elements that support development of systems for that domain”.

Due to their reusable nature, RAs are becoming a key asset of information technology consulting firms (ITCFs). Therefore, their exhaustive assessment (e.g., in terms of quality, cost and time reduction) becomes necessary. The goal of this paper is to present an empirical framework aimed to assess the RAs used by ITCFs in their IT projects executed in client organizations. This framework could be used by ITCFs to drive improvements on their RAs.

The paper is structured as follows. In Section 2 we present the context of our proposal. In Section 3 we describe the fundamental aspects of RAs that are suggested to be assessed. In Section 4 we describe the empirical studies that compose the framework. In Section 5 we present the ongoing application of the framework in the context of an ITCF. In Section 6 we end up with conclusions and future work.

II. CONTEXT OF IT CONSULTING FIRMS

We are interested in the case in which an ITCF has designed an RA with the purpose of deriving SAs for client organizations. This usually happens when the ITCF is regularly contracted to create or maintain ISs in client organizations. Each IS is built upon the derived SA (we call it RA-based SA) and includes many enterprise applications implemented on top of this SA (SA-based enterprise applications), see Fig. 1.
The use of RAs allows ITCFs to reuse their architectural knowledge and software components (normally associated to particular technologies) for the design of RA-based SAs in client organizations. Thus, a good RA guarantees a certain level of quality for each RA-based SA. Resulting RA-based SAs provide a baseline that facilitates standardization and interoperability as well as the attainment of business goals during enterprise applications’ development and maintenance.

In the scenario depicted in Fig.1, there are three kinds of projects with different targets: 1) RA projects; 2) RA-based SA projects; 3) SA-based enterprise application projects. Each kind of project has its own stakeholders who need to be clearly defined for assessment purposes [1]. RA projects are run exclusively by an ITCF team, specialized in architectural knowledge management. RA-based SA projects involve one ITCF team and likely another team from the client organization; their members are specialised in architectural design and have relevant knowledge of the organisation business domain. Finally, SA-based enterprise application projects can involve teams from the client organization and/or subcontracted ITCFs (which may even be different than the RA owner) whose members are usually very familiar with the specific organisation domain. The participation of the client organization in these two last types of projects is one possible strategy for ensuring the continuity of their ISs without having much dependency on the ITCF.

![Relationship among RAs, SAs and enterprise applications.](image)

**III. RELEVANT ASPECTS OF REFERENCE ARCHITECTURES**

In this section we identify important aspects to assess RAs. In [1], Angelov et al. state that SAs and RAs have to be assessed for the same aspects. For this reason, we started by analysing some available works on SA assessment [3][7]. However, due to the generic nature of RAs, some of the aspects found for SA assessment were not directly applicable to RA assessment. Therefore, we elaborated further this analysis considering both the specific characteristics of RAs as described in [1][10][11][13] and our own experience in the field. The resulting aspects for assessing RA are detailed below and summarized in Table I.

Aspect 1 refers to the need of having an *overview of the RA*. It includes an analysis of its generic functionalities, its domain [1], its origin and motivation, its correctness and utility, and its support for efficient adaptation and instantiation [11].

Falesi et al. [7] and other studies such as [10] highlight the importance of *requirements analysis and quality attributes*, as well as *decision-making and architectural evaluation* for the SA design process. These two aspects should also be considered for the RA assessment because, as we said, SAs and RAs have to be assessed for the same aspects [1]. Thus, we considered them as Aspects 2 and 3 respectively. However, since an RA has to address more architectural qualities than an SA (e.g., applicability) [1], this analysis could be wider for RAs in this sense.

SAs also address *business qualities* [1] (e.g., cost, time-to-market) that are business goals that affect their *competence* [3]. It is also applicable to RAs, so it is considered as Aspect 4.

To improve the SA design process, there also exist supportive technologies such as *methods*, and *techniques and tools* [7][13]. Thus, it is not only important for an RA to collect data to assess its design process, but also its supportive technologies, which are assessed by Aspects 5 and 6.
As stated in [7], a crucial aspect to define the goodness of a SA is related to the Return on Investment (ROI). The optimal set of architectural decisions is usually the one that maximizes the ROI. Aspect 7 is intended to quantify benefits and costs of RAs to calculate their ROI.

We recommend gathering evidence about all these aspects, which are summarised in Table I, while assessing an RA. Existing methods for SA assessment have been previously applied for RA assessment, such as in [1] and [10]. However, up to our knowledge none of them cover all the aspects of Table I. Hence, new approaches to assess RAs considering these aspects altogether are required. This has motivated our work.

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Description of the Architectural Aspect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Overview: functionalities [1], origin, utility and adaptation [11]</td>
</tr>
<tr>
<td>2</td>
<td>Requirements analysis [7], also called quality attributes [1][10]</td>
</tr>
<tr>
<td>3</td>
<td>Architectural knowledge and decisions [7][10][13]</td>
</tr>
<tr>
<td>4</td>
<td>Business qualities [1] and architecture competence [3]</td>
</tr>
<tr>
<td>5</td>
<td>Software development methodology [7][13]</td>
</tr>
<tr>
<td>6</td>
<td>Technologies and tools [7][13]</td>
</tr>
<tr>
<td>7</td>
<td>Benefits and costs metrics to derive SAs from RAs [7]</td>
</tr>
</tbody>
</table>

IV. AN EMPIRICAL FRAMEWORK TO ASSESS REFERENCE ARCHITECTURES

In this section, we present the ongoing version of our empirical framework to assess RAs (considering the relevant architectural aspects presented in Table I) in the context described in Section II. The framework aims to serve as a point of reference for practitioners (mainly ITCFs) that need to assess their RAs, who can apply it by themselves or with the support of a research team, as in our case (see Section V).

The pillars of the framework are the guidelines for conducting empirical studies in software engineering recommended by Wohlin et al. [17]. These guidelines suggest envisaging activities to understand, evaluate and improve the main object of a study, which is RAs in our case. Owing to it is impossible to start improving directly in most cases [17], the current version of the framework has mainly addressed the activities of understanding and evaluating RAs.

On the one hand, in order to understand the ITCF’s RA setting and how well such RA is working, the framework suggest three different and complementary types of empirical studies. First, qualitative surveys and case studies aimed to gather information related to the Aspects 1 to 6 (as defined in Section III). Second, a quantitative post-mortem analysis to target the collection of metrics related to the Aspect 7. These studies gather information not only from RA projects, but also from RA-based SA projects as they are a direct outcome of the RA usage, and from SA-based enterprise application projects as they are a direct outcome of the RA-based SA usage. This allows analysing the RA’s suitability for producing the RA-based SAs for the ITCF’s client organizations as well as the detection of improvement opportunities.

On the other hand, in order to evaluate the ITCF’s RA, there exist some evaluation methods in the literature (e.g., [1], [10] and the last step of [11]). These evaluation methods mainly gather information from RA projects, mostly covering Aspects 1 to 6. In our framework, we propose to apply one of these methods to evaluate the quality of an RA.

Below, the empirical studies that have been envisaged for our framework are explained. We use the same structure as in [7]: context and motivation, objectives, method and expected results. Table II summarises the studies that compose the framework and gives some guidelines to support their conduction.

A. Qualitative surveys to understand the current situation

Context: Before deciding to launch an RA-based SA project (or improving an RA), it is needed to understand RA’s characteristics, as well as its potential benefits and limitations. Assessing previous RA-based SA projects is a feasible way to start gaining such an understanding.

Objective: To understand the impact of using an RA in RA-based SA projects in the client organisations.

Method: Exploratory surveys with personalised questionnaires applied to relevant stakeholders (e.g., leader, architect, developer) to gather their perceptions and needs.
**Expected results:** To get an understanding of the impact and suitability of the RA for the elaboration of RA-based SA projects. Improvement insights can also be identified from different stakeholders.

### B. Quantitative post-mortem analysis to calculate ROI

**Context:** Before investing in an RA-based SA, project business leaders from prospective client organisations need to analyse whether undertaking or not the investment. Offering them evidence from former projects can help them to make more informed decisions.

**Objective:** To assess whether it is worth investing in an RA-based SA.

**Method:** An exploratory quantitative post-mortem analysis. It is aimed to quantify the potential advantages and limitations of using an RA-based SA. Results from the qualitative survey detailed above are an important input for the design of this study. Some examples can be found at [9] [16].

**Expected results:** A quantitative report that supports business leaders to make informed decisions.

### C. Case studies to seek an explanation of the situation

**Context:** Potential actions need to be envisaged after identifying potential problems, improvement opportunities and ROI information from the previous empirical studies.

**Objective:** To formulate hypotheses from the two previous studies and run case studies to seek an explanation for the intended hypothesis.

**Method:** Explanatory case study design to seek and explanation of previously identified situations.

**Expected results:** Feedback and interpretation with respect to previously identified situations from a case study in which the RA has been used.

### D. RA Evaluation to prove its effectiveness

**Context:** A positive evaluation of the RA would prove its effectiveness and quality. Hence, an RA must be evaluated to justify its use. The three previous studies offer potential insights for RA improvements for that cases in which the quality of the RA is not sufficient.

**Objective:** To evaluate the RA.

**Method:** An existing empirical method to evaluate RAs such as [1], [10] and the last step of [11].

**Expected results:** An evaluation of the RA to analyse its effectiveness and to determine which improvements should be incorporated in the RA.

### E. Summary of the empirical studies of our framework

Table II summarises the characteristics of the empirical studies of our framework. The first column shows in which step of the guidelines of [17] the study applies. The second column indicates its type. The third column points out the stakeholders that are involved in the study. Characteristics of the empirical methodological approach are in the fourth column. It comprises three characteristics [17]: the purpose of the study, which could be exploratory, descriptive, explanatory or improving; the collected data that may be quantitative or qualitative; and the research design that may be fixed, semi-fixed or flexible. The fifth column describes the main goal of each study. The sixth column emphasizes the aspects from Table I that should be covered by the study, which depends on its data collected (i.e., qualitative or quantitative). In the last columns, existing guidelines (both general and specific for software engineering) to conduct the studies are recommended.

It is important to note that the empirical studies suggested by our framework are complementary and support each other. Our framework benefits from this combination of studies. For instance, collecting data from different studies allows triangulation (i.e., data validation). Also, results from a preceding empirical study can be used to corroborate or develop further these results (e.g., using an explanatory case study to find out why the results from an exploratory survey are as they are). For this reason, the suggested studies have been designed to be conducted sequentially.
TABLE II. EMPIRICAL STUDIES OF THE FRAMEWORK TO ASSESS REFERENCE ARCHITECTURES

<table>
<thead>
<tr>
<th>Step</th>
<th>Type of study</th>
<th>Study’s stakeholders</th>
<th>Methodology characteristics</th>
<th>Goal</th>
<th>Aspects targeted</th>
<th>General</th>
<th>For SE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understand</td>
<td>Survey</td>
<td>Several RA-based SA teams, several SA-based enterprise application teams</td>
<td>Exploratory</td>
<td>To understand the scenario from each stakeholder’s perspective</td>
<td>1, 2, 3, 4, 5, 6</td>
<td>[12]</td>
<td>[4] [17]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Semi-fixed</td>
<td>Qualitative</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fixed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Post-mortem analysis</td>
<td>The RA team, several RA-based SA teams</td>
<td>Exploratory</td>
<td>To calculate the ROI for RA-based SAs</td>
<td>7</td>
<td>[12]</td>
<td>[6] [9]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Quantitative</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fixed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Case study</td>
<td>An RA-based SA team</td>
<td>Explanatory</td>
<td>To seek an explanation of the situation found out in the two previous studies</td>
<td>1, 2, 3, 4, 5, 6</td>
<td>[12]</td>
<td>[15] [17]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Qualitative</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Flexible</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evaluate</td>
<td>RA Evaluation</td>
<td>The RA team</td>
<td>Improving</td>
<td>To evaluate the RA quality</td>
<td>1, 2, 3, 4, 5, 6</td>
<td>n/a</td>
<td>[1] [7]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Qualitative</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fixed</td>
<td></td>
<td></td>
<td></td>
<td>[10] [11]</td>
</tr>
</tbody>
</table>

V. INSTANTIATION OF THE FRAMEWORK: THE EVERIS CASE

The presented empirical framework is currently being applied at the Architecture Centre of Excellence of Everis, a multinational ITCF. The main motivation of Everis for conducting the empirical studies is twofold: 1) technical: identifying strengths and weaknesses for their RA; 2) strategic: providing evidence to their clients about the potential benefits of applying their RA. Everis fits into the context described in Section II, e.g., they carry out the three types of projects described there.

Following the criteria found in [1], Everis’ RA can be seen as a Practice RA, since it is defined from the accumulation of practical knowledge. According to the classification of [2], it is also a classical, facilitation RA for multiple organisations designed by a software organisation in cooperation with user organisations. It is classical because its creation is based on experiences, and its aim is to facilitate guidelines for the design of systems, specifically for the IS domain. Fattah presents in [8] another classification scheme that would consider it as an enterprise RA because it is “a blueprint for the Solution Architecture [RA-based SA] of a number of potential projects [SA-based enterprise applications projects] within an organisation that embodies […] principles, policies, standards and guidelines”.

All the studies presented in this paper are planned to be conducted to assess Everis’ RA. The survey protocol has already been designed and reviewed. On the other hand, the post-mortem analysis to calculate the RA’s ROI is currently under design. The roles of the different stakeholders and an excerpt of the survey protocol are shown below. The complete version of the survey protocol is available at http://www.essi.upc.edu/~gessi/ecs12-survey-protocol.pdf .

A. Mapping between studies and stakeholders

As it was already said, stakeholders need to be clearly defined for RA assessment purposes [1]. In Everis’ projects, there are four kinds of stakeholders in both ITCF and client organisation teams: project business leader, project technological leader, software architect and developer. Each of these stakeholders has a vested interest in different architectural aspects, which are important to analyse and reason about the appropriateness and the quality of the three kinds of projects [10]. Table III shows how the roles are covered by the different studies in the Everis case.

B. The survey protocol of the Everis case

1) Sampling. The target population of this survey are RA-based SA projects and SA-based enterprise application projects. A representative sample of these projects in several client
organisations has been selected. Table III indicates with an ‘S’ the roles that will be interviewed in each project.

2) Approach for data collection. On the one hand, semi-structured interviews will be used for Project Technological Leaders and Software Architects, and Client’s Project Business Leaders. The reason of using interviews is that these roles have higher knowledge than the other roles about the architectural aspects of the Table I, or another perspective in the case of Client’s Project Business Leaders, so we want to collect as much information as possible from them. Prior to the interviews, questionnaires might be delivered to collect personal information about the interviewee and to inform him/her about the interview. On the other hand, online questionnaires will be used for RA-based SA Developers and SA-based enterprise application Developers, since most of their questions are about supportive technologies and their responses can be previously listed, simplifying the data collection process.

<table>
<thead>
<tr>
<th>Project</th>
<th>ITCF Team</th>
<th>Client Organization Team</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA</td>
<td>E</td>
<td>PBL</td>
</tr>
<tr>
<td>SA</td>
<td>C</td>
<td>E</td>
</tr>
<tr>
<td>Application</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>

a. Legend: Project Business Leader (PBL), Project Technological Leader (PTL), Software Architect (Arc), Developer (Dev), Survey (S), ROI study (R), Case study (C), and RA evaluation (E).

VI. CONCLUSIONS

Driving empirical studies is becoming one of the main sources of communication between practitioners and the academia. The main contribution of this work-in-progress paper intends to be the formulation of a framework to conduct empirical studies for assessing RAs. It consists of a list of relevant aspects for RAs assessment, and an assortment of four complementary empirical studies that allow assessing these aspects. The framework can be adapted to the specific context of ITCFs. Consequently, practitioners that apply the framework in their ITCFs, either by themselves or through collaboration with researchers, could benefit from a common reference framework to assess RAs.

Future work spreads into two directions. In terms of validation, we are conducting the Everis case using our framework, getting feedback for assessing its effectiveness. With respect to this first version of the framework, we aim to extend it considering Wohlin’s improvement step (see Section IV) in order to build preliminary guidelines for improving RAs in ITCFs.

ACKNOWLEDGMENTS

This work has been supported by the “Cátedra Everis” project. We would also like to thank Matthias Galster for his comments and suggestions.

REFERENCES

REARM: A Reuse-Based Economic Model for Software Reference Architectures

Silverio Martínez-Fernández¹, Claudia Ayala¹, Xavier Franch¹, Helena Martins Marques²

¹ GESSI Research Group, Universitat Politècnica de Catalunya, Barcelona, Spain
{smartinez,cayala,franch}@essi.upc.edu
² everis, Barcelona, Spain
hmartinn@everis.com

Abstract. To remain competitive, organizations are challenged to make informed and feasible value-driven design decisions in order to ensure the quality of their software systems. However, there is a lack of support for evaluating the economic impact of these decisions with regard to software reference architectures. This damages the communication among architects and management, which can result in poor decisions. This paper aims at ameliorating this problem by presenting a pragmatic preliminary economic model to perform cost-benefit analysis on the adoption of software reference architectures as a key asset for optimizing architectural decision-making. The model is based on existing value-based metrics and economics-driven models used in other areas. A preliminary validation based on a retrospective study showed the ability of the model to support a cost-benefit analysis presented to the management of an IT consulting company. This validation involved a cost-benefit analysis related to reuse and maintenance; other qualities will be integrated as our research progresses.

Keywords: Software architecture, reference architecture, economic model, architecture evaluation, cost-benefit analysis, quality attributes.

1 Introduction and motivation

Nowadays, the size and complexity of software systems, together with critical time-to-market needs, demand new software engineering approaches to software development. One of these approaches is the use of software reference architectures (RA), which are becoming widely studied and adopted in research and practice [3][19].

As defined by Bass et al. [5], an RA is “a reference model mapped onto software elements (that cooperatively implement the functionality defined in the reference model) and the data flows between them”. An RA encompasses the knowledge about how to design concrete software architectures (SA) of systems of a given domain; it must address the business rules, architectural styles, best practices of software development, and the software elements that support development of systems [28].

The motivations behind RAs are: to systematically reuse knowledge and software elements when developing concrete SA for new systems and thereby harvest potential
savings through reduced cycle times, cost, risk and increased quality [11]; to help with the evolution of a set of systems that stem from the same RA [18]; and to ensure standardization and interoperability [3].

However, although the adoption of an RA might have plenty of benefits for an organization, it also implies several challenges, among them the need for an initial investment [18]. Hence, in order to use RAs, organizations face a fundamental question: “Is it worth to invest on the adoption of an RA?”

Thus, organizations need to ensure the feasibility of adopting an RA by assessing their goals, the resources they can invest and the expected benefits. In spite of this need, there is a lack of research methods for economics-driven RA evaluation [29]. Besides, there is a shortage of economic models to “precisely evaluate the benefit of ‘architecture projects’ - those that aim to improve one or more quality attributes of a system” [8]. Thus, the adoption of RAs is usually made without evaluating their economic impact. To make informed decisions, it becomes necessary to make a business case in order to know how many instantiations (i.e., applications) are necessary before savings pay off for the up-front investment in building an RA.

The goal of this paper is to present a pragmatic preliminary economic model to perform cost-benefit analysis on the adoption of RAs as a key asset for optimizing architectural decision-making (referred to as REARM, REference ARchitecture Model). This goal is of interest for researchers for the need of formulating accurate models and practitioners for the opportunity of making more informed decision-making about whether to implement the strategic move to RA adoption. Due to the aforementioned lack of research in this specific area, we have aimed at adopting and adapting existing results in related areas, from classical software reuse to product line engineering.

It is worth mentioning that the paper has its origin in an ongoing action-research [36] initiative among our research group and everis, a multinational consulting company based in Spain. The architecture group of everis experienced the inability to calculate the return on investment (ROI) derived from RAs that they create for organizations. The model stemming from this collaboration is currently under formative evaluation [36], but results so far are already triggering change in some development processes in the organization (e.g., bug reporting). As part of the collaboration, we had the chance to provide an initial validation of the economic model. It comprises a retrospective evaluation of an RA created by everis for the IT department of a public administration center in Spain.

2 Background and related work

Current research on RA evaluation consists of analysis methods [2][17][21] that involve the analysis of risks, non-risks, benefits and trade-offs. Although they facilitate the analysis of those aspects based on the most important and critical scenarios, they have little support to analyze the cost and benefits of RAs based on economics.

Introducing an RA into an organization involves making a decision of a greater degree than only considering the aforementioned aspects, since it should not only include quality, but it should also include productivity issues. Whereas architectural
quality is usually estimated in relation to eliciting implicit and explicit requirements of the different stakeholders affected by the development of the system, productivity is actually measured in terms of effort, cost, and economic benefits. Nevertheless, both views are necessary to achieve a comprehensive analysis of the system.

Up to our knowledge, there is no specific economic model for estimating whether it is worth or not to invest in an RA for an organization. Due to the lack of research in this specific area, we have aimed at adopting and adapting existing results in related areas: economic models for software product lines (SPL), cost-benefit analysis methods for SAs, and more generic metrics about cost savings.

**Economic models for software product lines and software reuse.** The terms RA and product line architecture (PLA) are sometimes used indistinctly inside the SPL engineering context, in which the term RA is used to refer to “a core architecture that captures the high-level design for the applications of the SPL” [33, p. 124] or “just one asset, albeit an important one, in the SPL’s asset base” [9, p. 12].

However, out of the SPL context, RA and PLA are considered different types of artifacts [3][12][19][28]. In Fig. 1 we show the main similarities and differences:

- PLAs are RAs whereas not all RAs are PLAs [3], i.e. PLAs are one type of RAs [19]. PLAs are just one asset of SPL [9, p. 12].
- RAs are more generic and abstract than PLAs that are more complete architectures [3][19]. Hence, “RAs can be designed with an intended scope of a single organization or multiple organizations that share a certain property” [3] whereas PLAs are produced for a single organization [19].
- RAs provide standardized solutions for a broader domain (i.e., “spectrum of systems in a technology or application domain” [19]) whereas PLAs provide standardized solutions for a smaller subset of the software systems of a domain [28] (i.e., “group of systems that are part of a product line” [19]). Therefore, PLAs give a coherent and more congruent view of the products in a project (i.e., possible to track the status of) [12] whereas by means of RAs it is more difficult to obtain congruence [3], since they can only provide guidelines for applications’ development.
- PLAs specifically address points of variability and more formal specification in order to ensure clear and precise behavior specifications at well-specified extension points [3]. In contrast, RAs have less focus on capturing variation points [3][12][28]. Although variability is not typically addressed by RAs in a systematic manner, it is also a key fact for RAs [18], and it can be treated as a quality attribute, rather than explicitly as ‘features’ and ‘decisions’ [18].
- RAs include “the reuse of knowledge about software development in a given domain, in particular with regard to architectural design” [28] and dictate the patterns and principles to implement, i.e. “what the design should be” [12]. Conversely, PLAs specifically indicate deviations, i.e. “what the design is” [12].
- RAs include architectural knowledge and the instantiation of this architectural knowledge (i.e., reference model) into software elements [5]. In this sense, both RAs and PLAs are “a superset, a tool box, with every possible architecture element described, which can be used in the design of a product architecture” [12].
Although we also consider that RA and PLA are different, some perceived benefits of RA (e.g., cost saving from reusing software elements) and cost-benefit factors (e.g., common software costs, unique development costs) are applicable to both, since both have reuse as their core strategy. For this reason, we studied the applicability of some economic models originally conceived for SPL to RAs. Below, we summarize our results with respect to cost and benefit factors. To see more models, the reader is referred to [1], in which Ali et al. surveyed twelve economic models for SPL, and to [16][27] in which the authors surveyed economic models for software reuse.

Cost and Benefit Factors of Economic Models. SIMPLE [10], Poulin’s model [34], and COPPLIMO [7] are some of the most widespread economic models for SPLs. SIMPLE [10] comprises a set of seven cost factors:

- $C_{org}$, upfront investments to establish a SPL infrastructure.
- $C_{cab}$, the cost to build reusable assets of the SPL.
- $C_{unique}$, the cost to develop unique parts of products in a SPL.
- $C_{reuse}$, the cost of reusing reusable assets in a product inside the SPL.
- $C_{cabu}$, the cost to evolve the core asset in a SPL.
- $C_{prod}$, the cost to build a product in a stand-alone fashion.
- $C_{evo}$, the cost to evolve a product in a stand-alone fashion.
These cost factors and benefit functions can be used to construct equations that can answer a number of questions such as whether the SPL approach is the best option for development and what is the ROI for this approach. Ganesan et al. extended SIMPLE by considering infrastructure degeneration over time [20].

On the other hand, Poulin [34] and Boehm et al. [7] base their reuse-based models in two parameters: RCR and RCWR.

- **RCR (Relative Cost of Reuse).** Assuming that the cost to develop a reusable asset equals one unit of effort, RCR is the portion of this effort that it takes to reuse a reusable asset without modification (black-box reuse).
- **RCWR (Relative Cost of Writing for Reuse).** Assuming that the cost to develop a new asset for one-time use equals one unit of effort, RCWR is the portion of this effort that it takes to write a similar “reusable” asset.

For those cases in which there are difficulties to obtain historical data of building and evolving products in a stand-alone fashion ($C_{\text{prod}}, C_{\text{evo}}$), we consider more adequate the use of RCR and RCWR (see Section 4.1, step 2).

Finally, we must note two models (Schmid [37], InCoME [30]) that integrate cost and investment models in different layers, which make them more comprehensive.

**Value of software architecture design decisions.** There exist a few economics-based SA analysis methods that drive the decision-making process during SA review and design. In this direction, CBAM [22] is a useful method for prioritizing architectural decisions that bring higher value. In addition, Ozkaya et al. proposed an economic valuation of architectural patterns [32].

These approaches help to find the optimal set of decisions that maximizes the ROI [15]. They pursue to solve the same problem of this paper, but their scope is broader and general for any kind of SA decision and do not reflect fundamental characteristics of adopting an RA. Therefore, their applicability for studying the ROI of RA adoption would require more effort, since specific cost-benefit factors for architecture-centric reuse are not considered. Hence, they are not the most convenient approaches for making the business case of adopting an RA and calculating its payback time.

**Generic software metrics.** There exist several approaches that propose metrics for estimating cost savings in software development and maintenance. Metrics as dependency structure matrices (DSM) have been applied to assist architecture-level analysis, such as value of modular designs, and they have proven to be particularly insightful for validating the future value of architecting modular systems [8]. MacCormack et al. extracted coupling metrics from an architecture DSM view for inferring the likelihood of change propagation and, hence, future maintenance costs [25]. Baldwin et al. presented a generic expression for evaluating the option to redesign a module also based on DSMs [4].

In addition, the concept of technical debt (either architecture-focused [31] or code-based [24]) is a way to measure unexpected rework costs due to expediting the delivery of stakeholder value in short.
Summary. Although there is a lack of research in evaluating the economic viability of RA adoption, there is a strong base of research in related areas. The most important related area is economic models that identify cost and benefit factors for PLA adoption. Although there is a significant amount of research in this direction, it falls short in:

- Validation in industry. “Very few [economic models for SPL] actually have been used as a basis for further development or adopted in industry” [23]. Thus, “there is a clear need for many more empirical studies to validate existing models” [1].
- Easy adoption of models in industry by identifying realistic metrics to collect and report. “It is difficult for the practitioners to evaluate the usability and usefulness of a proposed solution [economic model for SPL] for application in industry” [23]. Not guidelines exist to fully operationalize the models in practice [37].

Economics-driven SA analysis methods do not specifically aim at making an investment analysis of the adoption of an architecture-centric program. RA adoption is a subarea inside their generic decision-making context.

At a lower level, more simple metrics like DSM, could also be adequate for calculation the cost and benefit factors of RA adoption and make more complete models.

This state of the art drove us to the formulation of an economic model for RAs, which is currently on its formative stage. The formulation of the model aims to:

- Adapt cost and benefit factors from SPL models that are easy-to-apply by industry.
- Fill the gap of RA economics inside the SA decision-making context.
- Look for generic software metrics that can quantify new cost and benefit factors.

3 Industrial context

The architecture group of everis is an initiative to manage architectural knowledge, best practices and lessons learned from previous experiences; and to provide efficient solutions to a better cost, flexibility and agility to the demands of client organizations.

This architecture group offers solutions for big businesses (e.g., banks, insurance companies, public administration and service, and industrial organizations) that offer a wide spectrum of services to their clients. Often, already existing commercial packages are not completely aligned with the business needs of these organizations, thereby requiring custom development and maintenance of applications. In this scenario, everis foster the use RAs for managing a wide spectrum of applications.

The architecture group of everis experienced the inability to calculate the ROI derived from RAs that they create for organizations. The purpose of our research is to create a method for extracting costs and benefits of RAs based on data that they were already collected.
4 An economic model for reference architectures

4.1 Method for formulating the economic model

An RA cost-benefit analysis should be based on giving an economic value to its activities. We designed our economic model through the three following steps:

1. **Identify the costs and benefits stemming from the use of an RA.** Although cost modeling is already a mature field within software engineering, benefits have traditionally been far more elusive to quantify [8]. For this reason, it is necessary to identify the RA quality attributes that bring more benefit to the development and maintenance of applications, and the costs of constructing these applications [22]. These attributes may vary depending on the architecturally-significant requirements coming from the applications based on the RA. It is crucial to involve relevant stakeholders to ensure the trustworthiness of the collected information [38].

   The outputs of this step are, therefore, the costs factors of adopting an RA and the list of quality attributes in which the RA brings more benefit.

2. **Adopt metrics that quantify the costs and benefits identified in the first step in order to convert them into a monetary value.** The metrics to quantify them may vary depending on the data available in the organization involved.

   The output of this step is providing guidelines for collecting simple metrics that make possible to calculate the cost and benefits factors in practice.

3. **Make the business case for the adoption of the RA.** Add the costs and benefits calculated in the second step to the formula for calculating the ROI (proposed by Boehm [6]), where the benefits are the improvements of applications quality attributes, and the costs are the expenses in constructing the systems and the RA.

   The output of this step is a business case that captures the reasoning for adopting an RA. The RA business case analysis involves determining the relative financial costs, benefits, and ROI across its life-cycle.

   \[
   \text{ROI} = \frac{\text{Benefits} - \text{Costs}}{\text{Costs}}
   \]  

4.2 Execution of the method for formulating the economic model

The action-research collaboration with *everis* provided us the opportunity of implementing this general-purpose method in a particular case.

**Step 1.** We conducted a survey involving project managers, architects and developers of 9 organizations in Europe (7 from Spain) [26]. The survey pointed out that the main perceived economic benefits on the use of RAs were: (1) an increased value from the improvement of quality attributes, since their reused architectural knowledge is incrementally improved with previous successful experiences from its application domain; (2) cost savings in the development and maintenance of systems due to the reuse of software elements and the adoption of best practices of software development that increase the productivity of developers. Therefore, RAs bring most of the benefit...
because of the improvement of reusability and maintainability quality attributes. One of the reasons why RAs were adopted in these organizations is that the most important architecturally-significant requirement was reusability. Thus, we decided to focus our cost-benefit analysis over reusability and maintainability.

We found that some of the potential metrics to be used were not as pragmatic as the organization needed. In other words, the organization should have been invested extra time which was not an option. Furthermore, we faced the problem that some of the required data to apply the proposed metrics was not previously registered by the organization. Thus, we stressed the emphasis on formulating a practical model that incrementally deals with diverse cost-benefit aspects.

We identified six cost-benefit factors for RA adoption. We started the formulation of factors by adopting Poulin’s method for measuring code reuse [34][35]. We adapted Poulin’s model because it has been applied in industry, offers parameters to operationalize it, and we could feed it with available data in everis (see Step 2 below).

We adopted its benefit factors (DCA, SCA) published in [35]. Conversely, we consider more appropriate for RAs to adopt the cost factors defined for SPL (CSW\textsubscript{dev\_costs}, CSW\textsubscript{service\_costs}) in [35], instead of the additional development costs [34].

To complete the model we add the unique development costs of applications. Also, with the help of the propagation cost metric [25], we also consider necessary changes to reusable elements (which are not considered by Poulin’s method) and, therefore, evolution. These two new factors include parameters to operationalize them.

The former three factors are for development and the latter ones for maintenance:

- **DCA (Development Cost Avoidance).** It is the benefit from reusing RA’s software modules in applications compared to building the applications independently.
- **UDC (Unique Development Costs).** It is the cost to develop the unique parts of an application that are not already implemented in the modules of the RA. UDC is equivalent to $C\text{reuse}+C\text{unique}$.
- **CSWD (Common Software Development costs).** It is the cost of the initial investment, i.e., developing an RA. CSWD is equivalent to $C\text{org}+C\text{cab}$.
- **SCA (Service Cost Avoidance).** It is the benefit of modifying reused code once.
- **CSWS (Common Software Service costs).** It is the cost of fixing bugs in the (reusable) RA modules. CSWS calculates the cost of changes due to bugs in $C\text{cabu}$.
- **CSWE (Common Software Evolution costs).** It is the cost of changing or adding functionalities to the RA modules. CSWE calculates the cost of evolutions in $C\text{cabu}$. Therefore, CSWS+CSWE are equivalent to $C\text{cabu}$.

Putting everything together, given a number $n$ of applications built in top of the RA, and a number $m$ of RA modules changed as it evolves, the benefits and costs of adopting an RA are defined as:

$$\text{Benefits}=\sum_{i=1}^{n}(DCA_i+S\text{CA}_i)$$  \hspace{1cm} (2)

$$\text{Costs}=\text{CSWD}+\text{CSWS}+\sum_{i=1}^{n}U\text{DC}_i+\sum_{j=1}^{m}\text{CSWE}_j$$  \hspace{1cm} (3)

**Step 2.** We divide the second step in two activities: checking the data available in practice and guide the information extraction from this data.
Data commonly available in practice that should be collected. The data that typically is available in order to calculate the aforementioned costs and benefits are effort and software metrics [26]. It allows converting cost-benefit factors into a monetary value.

On the one hand, the invested effort from the tracked activities allows the calculation of costs. We distinguish between three types of activities: training, development and maintenance. JIRA¹ and Redmine² are tools that support keeping track of activities and their invested time. Keeping track of activities is common in practice for project management and auditing. Activity tracking is also known as tickets [8].

On the other hand, software metrics help to analyze the benefits that can be found in the source code. For example, since the cost of applications’ development is lower because of the reuse of RA, we estimate the cost avoidance of reusing its LOC. Sonar³ offers tool support for obtaining general software metrics such as LOC, dependencies between modules, technical debt [24], and percentages of tests and rules compliance.

Software development is a naturally low-validity environment and reliable expert intuition can only be acquired in a high-validity environment. As stated by Erdogmus and Favaro [14], the adoption of practices like time tracking and tools to collect data is the basis for moving software development from its usual low-validity environment, to a high-validity environment.

We experienced difficulties collecting historical data (as in [20]), especially for the “before” state of adopting an RA. We noted that Cₚᵣₒᵈ and Cₑᵥₒ were seldom available since the “before” state did not exist. For this reason, we use RCR and RCWR.

Using commonly available data in practice to quantify the costs and benefits. In Table 1, we present ten basic parameters that are required for calculating the six cost-benefit factors of the Step 1. Table 2 shows the formulas to calculate this six cost-benefit factors as well as parameters that are needed for these calculations.

<table>
<thead>
<tr>
<th>Table 1. Basic parameters in order to feed the factors of Table 2.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description of the parameters (adapted for the RA context)</strong></td>
</tr>
<tr>
<td><strong>RCR</strong></td>
</tr>
<tr>
<td><strong>RCWR</strong></td>
</tr>
<tr>
<td><strong>ER</strong></td>
</tr>
<tr>
<td><strong>EC</strong></td>
</tr>
<tr>
<td><strong>NMSI</strong></td>
</tr>
<tr>
<td><strong>PC</strong></td>
</tr>
<tr>
<td><strong>CPKL</strong></td>
</tr>
</tbody>
</table>

¹ JIRA, http://www.atlassian.com/es/software/jira/overview
² Redmine, http://www.redmine.org/
³ Sonar, http://www.sonarsource.org/
USI | Unique Source Instructions: the amount of unique software (i.e., not reused) that was written or modified for an application
---|---
RSI | Reused Source Instructions: it is the total LOC of the RA’s modules that are reused in an application. It supports variability. In other words, reuse of RA might not be complete but partial, since different applications can reused different RA’s modules. Therefore RSI depend on each application [34].
TSI | Total Source Instructions: it is the total LOC of the RA that can be reused [34].

| Table 2. Cost-benefit factors to calculate the ROI of adopting an RA in an organization. |
| Description of the cost-benefit factors (adapted for the RA context) |
| DCA | Development Cost Avoidance: the benefits from reusing RA’s modules [34] \( DCA = RSI \times (1-RCR) \times CPKL \) |
| CSWD | Common Software Development costs: the costs to develop the RA [35] \( CSWD = RCWR \times TSI \times CPKL \) |
| UDC | Unique Development Costs: the costs to develop the unique part of an application \( UDC = USI \times CPKL \) |
| SCA | Service Cost Avoidance: benefits from maintaining only once RA’s modules [34] \( SCA = RSI \times ER \times EC \) |
| CSWS | Common Software Maintenance costs: cost of fixing bugs in reusable modules [35] \( CSWS = TSI \times ER \times EC \) |
| CSWE | Common Software Evolution costs: the costs of changing or adding a new functionality and maintaining it to the RA \( CSWE = (NMSI \times RCWR \times CPKL) + (NMSI \times ER \times EC) + (TSI \times CPKL \times PC) \) |

**Step 3.** As final step, we can use calculated factors in order to calculate the ROI:

\[
ROI = \frac{\left[\sum_{i=1}^{n_i} (DCA_i + SCA_i)\right] - \left[CSWD + CSWS + \sum_{i=1}^{n_i} UDC_i + \sum_{j=1}^{m} CSWE_j\right]}{CSWD + CSWS + \sum_{i=1}^{n_i} UDC_i + \sum_{j=1}^{m} CSWE_j}
\]  

(4)

We also suggest using these cost-benefit factors to make a business case for calculating the ROI of building an RA vs. building the applications independently. Table 3 shows an example of business case and how to calculate the cost and benefits for three years since the RA adoption. The parameters \( n_1, n_2, n_3 \) indicate the number of applications developed per year respectively, and \( m \) the number of evolved modules.

<p>| Table 3. Example of design of a business case with the cost-benefit factors of the model |</p>
<table>
<thead>
<tr>
<th>Year 1</th>
<th>Year 2</th>
<th>Year 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total benefit</td>
<td>( n_1 \times (DCA + SCA) )</td>
<td>( n_2 \times (DCA + SCA) )</td>
</tr>
<tr>
<td>Total cost</td>
<td>( CSWD + n_1 \times UDC + CSWS \times \frac{4}{5} )</td>
<td>( n_2 \times UDC + CSWS \times \frac{3}{5} + m \times CSWE )</td>
</tr>
</tbody>
</table>

As Boehm points out [6], two additional factors may be important in business case analysis: unquantifiable benefits, and uncertainties and risk.

First, the economic model that we propose promotes benefits in reusability and maintainability. However, other quality attributes, such as security, could be as rele-
vant as those for this analysis, even when they may be difficult to quantify. This other benefits should also been taken into account when adopting and RA. Unquantifiable benefits are also considered as “flexibility” in TEI\(^4\), the economic model of Forrester.

Second, to adjust cost and benefits to risk, they can be multiplied by percentages that generally increase the costs and reduce the benefits (assuming the worst case). For instance, TEI propose to multiple costs by values that range from 98% to 150% and benefits by values between 50% and 110%.

5 Preliminary validation

To assess the feasibility of the economic model, we conducted a retrospective analysis of a particular case. We calculated the costs and benefits (and hence the ROI) of an RA adoption driven by *everis* in the IT department of a public organization.

By the time we performed the validation, the public organization had already: (1) adopted an RA, (2) created an application using the RA—which we consider “exemplar” application—, and (3) fixed errors discovered in the RA software elements that were reused by the application.

The validation consisted of 4 parts. First, a post-mortem analysis in which our challenge was to extract the parameters of Table 1 from already collected data. The values that we got are shown in Table 4.

<table>
<thead>
<tr>
<th>RCR</th>
<th>RCWR</th>
<th>ER</th>
<th>EC</th>
<th>NMSI</th>
<th>PC</th>
<th>CPKL</th>
<th>USI</th>
<th>RSI</th>
<th>TSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.064</td>
<td>1.243</td>
<td>2,879</td>
<td>7.02</td>
<td>1.526</td>
<td>9.7%</td>
<td>75,22</td>
<td>2.885</td>
<td>8,364</td>
<td>41.189</td>
</tr>
</tbody>
</table>

\(^1\) In TSI, 9,231 LOC were refactored from previous project. So, 31,958 were new.

Recommended values for RCR range from 0.03 and 0.25, and for RCWR from 1 to 2.2 [34]. Therefore, with the values that we got in the study, we can see that both RCR and RCWR are low for RAs. A low RCR could show the trend of moving the complexity to the architecture in order to simplify the development of applications. We can also see this trend comparing the code of the RA software elements with the code of applications. RA code present higher values for complexity metrics such as coupling and cohesion. A reason why RCWR is low could be that RA architectural knowledge speeds up the development.

Second, with the data of Table 4, we had real data to calculate (see Table 5):

- CSWD, the RA initial investment, which lasted 6 months.
- DCA, the benefit of reusing RA code in the exemplar application development.
- SCA, the cost from fixing the errors of the reused code in the exemplar application.
- UDC, the cost of developing the application.

The above costs were accurately computed because *everis* keeps track of activities with their invested time. Third, it was necessary to estimate the rest of factors:

- **CSWS**, the cost of fixing all bugs in RA code. Since we knew the SCA for the exemplar application and the percentage of reuse, we calculated the error rate and error cost, which we used to estimate CSWS.
- **CSWE**, the cost of: (1) changing or developing a module with new functionality, (2) fixing its bugs, (3) making changes in the rest of the RA to integrate it.

Table 5. Values of the cost-benefit factors in the study.*

<table>
<thead>
<tr>
<th>Factor</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCA</td>
<td>589 hours</td>
</tr>
<tr>
<td>CSWD</td>
<td>2,988 hours</td>
</tr>
<tr>
<td>UDC</td>
<td>217 hours</td>
</tr>
<tr>
<td>SCA</td>
<td>169 hours</td>
</tr>
<tr>
<td>CSWS</td>
<td>832 hours</td>
</tr>
<tr>
<td>CSWE</td>
<td>474 hours</td>
</tr>
</tbody>
</table>

*Values in bold are real data. Values in italic are estimated.

Fourth, we made the business case analysis with two different scenarios:

**Scenario 1. Is it worth to invest on the adoption of an RA?** We constructed a business case for 3 years starting when the organization decided to adopt the RA, in order to calculate the ROI. For the first 8 months of those 3 years, we have real data about the RA development and the exemplar RA-based application. To estimate the costs and benefits for the rest of these 3 years, we conducted some additional interviews to the involved stakeholders. Stakeholders were carefully selected according to their knowledge and experience to increase the degree of confidence on the data gathered. After these interviews, we made the following assumptions:

- Future applications will have similar characteristics and complexity as the exemplar one.
- The public organization will develop 8 applications per year. Since the RA creation lasted 6 months, the first year they will develop just 4 applications.
- The totality of CSWS is computed proportionally starting the seventh month.
- A module is evolved (with new functionality) or added to the RA every year since the second year.

Under these assumptions, the costs and benefits in hours for the future can be calculated as shown in Table 3. They can be converted into a monetary value by multiplying them by an hourly rate. Assuming a rate of 30€ per hour for an application developer (which affects to DCA, SCA, UDC) and a rate of 40€ per hour for a developer and maintainer of the architecture (which affects to CSWD, CSWM, CSWE), Fig. 2 summarizes financial results for first three years of the RA. This organization will realize a ROI within 2 years through gains in systematic reuse.

**Scenario 2. How many instantiations (i.e., applications) are necessary before savings pay off for the up-front investment in building an RA?** In this scenario we calculated how many applications need to be build based on the RA to have a positive ROI. Fig. 3 shows the ROI due to developing and maintaining applications based on an RA rather than in a stand-alone fashion.
As Fig. 3 shows, after building 7 applications, savings pay off for the up-front investment in the RA. It must be noted that the exemplar application is small and only 20% of the RA is being reused (RSI/TSI). On the other hand, the application has a high reuse percentage of 74% (RSI/USI+RSI). The higher these percentages are (likely in medium to large applications), the greater the benefit from the RA is.

Moreover, applications are introduced into the market earlier from the seventh month on. This is due to the effort avoidance of 589 hours (DCA) of reusing the RA.

To sum up, this study illustrates the potential way in which an organization can evaluate the value of RA adoption. We calculated a three-year ROI of 42% with a payback period of 16.5 months and 7 applications.
6 Discussion

Once we applied the economic model and calculated the ROI, a last question remains: How accurate are these calculations and the obtained quantitative data? If the economic model is applied with existing data (as we have done in Section 5), the calculation of the ROI reaches a high degree of correctness, since the data that feeds the model is trustworthy. The metrics coming from code analysis (e.g., size in LOC) do not reflect any error. Also, we saw that time tracking is reliable. During data collection we found invested time in activities in two different sources: JIRA, which is optionally used by the project team and keeps the invested time of the project’s activities; and a mandatory corporate financial tool, which is used by the financial department. This data differ in 8.75%, being lower internally time tracking of the project. The reason could be that JIRA does not include other activities out of the scope of the project like traveling. To adjust the calculations to this risk, we have always considered the worst case (i.e., greater costs).

Contrary, when the economic model is used to predict the ROI of a completely new RA adoption in an organization, there is not real data since it does not exist yet. In this case, the accuracy totally depends on expert intuition and historical data. Historical data can be scarce in small and medium organizations; especially considering that reuse of architectures is still a research area in progress. In addition, historical data must be continuously updated, since some values of effort-related parameters (such as RCR) are expected to decrease each time a developer instantiates the RA.

As a final remark, the construction of an economic model from the data available in software companies is yet another instance of research question which needs to balance soundness with applicability. The awareness of this problem by the software engineering community is increasing and even dedicated events are being organized (See CESI 2013 @ ICSE, http://www.essi.upc.edu/~franch/cesi2013/).

7 Conclusions & next steps

Architecture improvements are extremely difficult to evaluate in an analytic and quantitative way, contrary to business efficacy (sales, marketing, and manufacturing) [8]. Methods and models for changing this state of the practice are demanded.

This paper has opened the path on the area of using economic models for RA assessment. We think that this area has a significant impact not just for researchers but also for practitioners in software development and organization’s executives. We presented REARM, an economic model to translate measured or estimated data (i.e., metrics) into monetary terms (i.e., cost-benefit analysis). Then, we use them as the basis for analyzing the economic value of an RA (i.e., valuation) that is adopted by an organization in the pursuit of its business strategies. Thus, our work aligns with Erdoganmus et al. vision on economic activities in software industry, that fall into 4 levels: metrics, cost-benefit analysis, valuation and business strategy [13].

We have conducted a preliminary validation to calculate the ROI of adopting an RA in a real organization. This organization will realize a return on their investment
within two years through gains in systematic reuse and applications maintainability. The method presented is generic enough to be used when other quality attributes are prioritized by relevant stakeholders. The presented economic model allows quantifying the value that an RA of Type 2 or 4 (those designed with an intended scope of a single organization) brings to an organization. Its strongest points are:

- It translates RA costs and benefits into monetary values, which can be considered an innovative approach in RA research and practice.
- The integration of different metrics from existing models that complement each other evaluating several RA-relevant aspects.
- It provides guidelines for easily collecting and reporting data for practitioners, and for using it to make a business case.
- The model has been applied in a public organization and validated with real data.

On the other hand, potential weaknesses of this approach are:

- It does not consider RA’s software elements degeneration over time [20].
- The risk increases when neither real nor historical data are available.

As future work, we plan to enrich the economic model by: (1) adding more metrics (such as technical debt [24], degeneration over time [20], risk metrics, homogeneity metrics [10]), and (2) validating it for bigger applications and in more organizations.

**Acknowledgements.** This work has been supported by “Cátedra everis” and the Spanish project TIN2010-19130-C02-00. We would also like to thank all participants of the data collection process for their kindy cooperation.

**References**

A Reuse-Based Economic Model for Software Reference Architectures

Silverio Martínez-Fernández¹, Claudia Ayala¹, Xavier Franch¹

¹ Software Engineering for Information Systems Group
Universitat Politècnica de Catalunya (GESSI-UPC)
Barcelona, Spain
{smartinez, cayala, franch}@essi.upc.edu

November 2012

Report ESSI-TR-12-6
Departament d'Enginyeria de Serveis i Sistemes d'Informació
A Reuse-Based Economic Model for Software Reference Architectures

Silverio Martínez-Fernández, Claudia Ayala, Xavier Franch
Universitat Politècnica de Catalunya (UPC)
Barcelona, Spain

Abstract—The growing size and complexity of software systems, together with critical time-to-market needs, demand new software engineering approaches for software development. To remain competitive, organizations are challenged to make informed and feasible value-driven design decisions in order to ensure the quality of the systems. However, there is a lack of support for evaluating the economic impact of these decisions with regard to software reference architectures. This damages the communication among architects and management, which can result in poor decisions. This paper aims at opening a path in this direction by presenting a pragmatic preliminary economic model to perform cost-benefit analysis on the adoption of software reference architectures as key asset for optimizing architectural decision-making. The model is based on existing value-based metrics and economics-driven models used in other areas. A preliminary validation based on a retrospective study showed the ability of the model to support a cost-benefit analysis presented to the management of an IT consulting company. This validation involved a cost-benefit analysis related to reuse and maintenance; other qualities will be incorporated as our research progresses.

Index Terms—Software architecture, reference architecture, architecture evaluation, cost-benefit analysis, quality attributes.

I. INTRODUCTION AND MOTIVATION

Nowadays, the size and complexity of software systems, together with critical time-to-market needs, demand new software engineering (SE) approaches to software development. One of these approaches is the use of software reference architectures (RA), which are becoming widely studied and adopted in SE research and practice [3]. An RA encompasses the knowledge about how to design concrete software architectures (SA) of systems of a given application domain; therefore, it must address the business rules, architectural styles, best practices of software development, and the software elements that support development of systems [18].

The motivation behind RAs is to systematically reuse knowledge and software elements when developing concrete SA for new systems, to help with the evolution of a set of systems that stem from the same RA, or to ensure standardization and interoperability. However, although the adoption of an RA might have plenty of benefits for an organization, it also implies several challenges, among them the need for an initial investment [11].

Thus, organizations need to ensure the feasibility of adopting an RA by assessing their goals, the resources they can invest and the expected benefits. In spite of this need, there is a lack of research methods for economics-driven RA evaluation [19]. In particular, there is a shortage of economic models to “precisely evaluate the benefit of ‘architecture projects’ - those that aim to improve one or more quality attributes of a system” [6]. Thus, the adoption of RAs is usually made without evaluating their economic impact.

The goal of this paper is to set a new research direction on RA evaluation attracting both: researchers for the need to formulate accurate models, and practitioners for the opportunity of making more informed decision-making about whether to make the strategic move to RA adoption.
Due to the aforementioned lack of research in this specific area, we have aimed at adopting and adapting existing results in related areas, from classical software reuse to product line engineering.

It is worth mentioning that the paper has its origin in an ongoing action-research initiative among our research group and the Center of Excellence on Software Architectures (ARCHEX) of Everis, a multinational consulting company based in Spain. ARCHEX experienced the inability to calculate the return on investment (ROI) derived from RAs that they create for organizations. The model stemming from this collaboration is currently under formative evaluation, but the current results are already triggering change in some development processes in the organization (e.g., bug reporting and management). As part of the collaboration, we had the chance to provide an initial validation of the economic model. It comprises a retrospective evaluation of an RA created by Everis for the IT department of a public administration center in Spain.

II. BACKGROUND

Current research on RA evaluation [2][10][12] has little support to analyze the cost and benefits of RAs based on economics. However, there exist a few SA evaluation methods that drive the decision-making process during SA review and design. In this direction, Moore et al. presented CBAM [17], Ozkaya et al. proposed an economic valuation of architectural patterns [21], and Ivanovic et al. quantified the customer value that stems from quality improvements [13]. Although these SA methods are useful for prioritizing architectural strategies that bring higher value, they can be applied only in a restricted way for RAs adoption. RAs involve fundamental assumptions that these methods do not reflect, especially when it comes to defining potential economic benefits and the payback time.

Another inspiring area is Software Product Lines (SPL). A survey by Ali et al. [1] summarized twelve economic models for SPL. Among them, we may remark SIMPLE [7]. SIMPLE comprises a set of cost and benefit functions that can be used to construct equations that can answer a number of questions such as whether the SPL approach is the best option for development and what is the ROI for this approach.

On the other hand, there exist several approaches that propose metrics for estimating cost savings in software development and maintenance. For instance, Poulin considered the reuse of assets in developing individual applications, and the potential costs savings that it implies to the development and maintenance [22]. Metrics as dependency structure matrices (DSM) have been applied to assist architecture-level analysis, such as value of modular designs, and they have proven to be particularly insightful for validating the future value of architecting modular systems [6]. MacCormack et al. extracted coupling metrics from an architecture DSM view for inferring the likelihood of change propagation and, hence, future maintenance costs [15]. Baldwin et al. presented a generic expression for evaluating the option to redesign a module also based on DSMs [4] that is used in [24]. In addition, the use of technical debt (either architecture-focused [20] or code-based [14]) is a way to measure unexpected rework costs due to expediting the delivery of stakeholder value in short.

Despite the existence of this inspiring body of work from the SPL, software reuse and modular design areas, the proposed models are not directly applicable to RAs. Although SPL and RAs have similarities (both have reuse as their core strategy) they have also important differences. RAs deal with the range of knowledge of an application domain, providing standardized solutions for such a broad domain, while SPL approaches are more specialized, focusing sometimes on a specific subset of the systems of a domain and providing standardized solutions for a smaller family of systems [18].

This state of the art drove us to the formulation of an economic model for RAs, which is currently on its formative stage.
III. AN ECONOMIC MODEL FOR REFERENCE ARCHITECTURES

A. Method for formulating the model

An RA cost-benefit analysis should be based on giving an economic value to its activities. This can be done in three steps:

1) Identify the costs and benefits stemming from the use of an RA: It is necessary to identify the RA quality attributes that bring more benefit to the development and maintenance of systems, and the costs of constructing these systems [17]. These attributes may vary depending on the characteristics of RA [3]. It is crucial to involve relevant stakeholders to ensure the trustworthiness of the collected information [23].

2) Adopt metrics that quantify the costs and benefits identified in the first step in order to convert them into a monetary value: The metrics to quantify them may vary depending on the data available in the organization involved.

3) Add the costs and benefits calculated in the second step to the formula for calculating the ROI:

\[
\text{ROI} = \frac{\text{Benefits} - \text{Costs}}{\text{Costs}}
\]  

where the benefits are the improvements of system quality attributes, and the costs are the expenses in constructing the systems and the RA. Eq. 1 has been proposed by Boehm [5].

B. Example of application

The action-research collaboration with Everis provided us the opportunity of implementing this general-purpose method in a particular case.

For the first step, we conducted a survey involving project managers, architects and developers of 9 organizations in Europe (7 from Spain) [16]. The survey pointed out that the main perceived economic benefits on the use of RAs were: (1) an increased value from the improvement of quality attributes, since their reused architectural knowledge is incrementally improved with previous successful experiences from its application domain; (2) cost savings in the development and maintenance of systems due to the reuse of software elements and the adoption of best practices of software development that increase the productivity of developers. These cost savings could be seen as the improvement of reusability and maintainability quality attributes. Thus, we decided to focus our cost-benefit analysis over these particular dimensions.

For the second step, we found that some of the potential metrics to be used were not as pragmatic as the organization needed. In other words, the organization should have been invested extra time which was not an option. Furthermore, we faced the problem that some of the required data to apply the proposed metrics was not previously registered by the organization. Thus, we stressed the emphasis on formulating a practical model that incrementally deals with diverse cost-benefit aspects. This helped us to prioritize the operationalization of these aspects into the model.

We started the formulation of metrics by adopting Poulin’s method for measuring code reuse [22]. With the help of the propagation cost metric [15], we also consider necessary changes to reusable elements (which are not considered by Poulin’s method) and, therefore, maintainability. For this purpose, we use six cost-benefit factors, which we classify into development and maintenance:

a) Benefits and costs of development:

- DCA (Development Cost Avoidance). It is the benefit from reusing RA’s software modules in applications compared to building the applications independently.
- UDC (Unique Development Costs). It is the cost to develop the unique parts of an application that are not already implemented in the modules of the RA.
- ADC (Additional Development Costs). It is the cost of developing an RA for a particular organization.
b) Benefits and costs of maintenance:

- **SCA (Service Cost Avoidance).** It is the benefit from modifying the reused code only once.
- **AMC (Additional Maintenance Costs).** It is the cost of fixing bugs in the (reusable) RA modules.
- **AEC (Additional Evolution Costs).** It is the cost of changing or adding functionalities to the RA modules.

Given a number \( n \) of applications built in top of the RA, and a number \( m \) of RA modules added or changed as it evolves, the benefits and costs of adopting an RA are defined as:

\[
\text{Benefits} = \sum_{i=1}^{n} (\text{DCA}_i + \text{SCA}_i) \\
\text{Costs} = \text{ADC} + \text{AMC} + \sum_{i=1}^{m} \text{UDC}_i + \sum_{j=1}^{m} \text{AEC}_j
\]

Table I shows the formulas to calculate the six factors as well as parameters that are needed for these calculations.

As final step, we can use the benefits of Eq. 2 and costs of Eq. 3 into the Eq. 1 in order to calculate the ROI.

### TABLE I. PARAMETERS OF THE ECONOMIC MODEL TO CALCULATE THE ROI OF ADOPTING A REFERENCE ARCHITECTURE IN AN ORGANIZATION

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description (adapted for the RA context)</th>
<th>Value in the study</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{RCR} ) [22]</td>
<td>Relative Cost of Reuse: effort that it takes to reuse a component without modification versus writing it new one-at-a-time</td>
<td>0.064</td>
</tr>
<tr>
<td>( \text{RCWR} ) [22]</td>
<td>Relative Cost of Writing for Reuse: effort that it takes to write a reusable component versus writing it for one-time use only</td>
<td>2.468</td>
</tr>
<tr>
<td>( \text{ER} ) [22]</td>
<td>Error Rate: the historical error rate in new software developed by your organization, in errors per thousand lines of code</td>
<td>2.015</td>
</tr>
<tr>
<td>( \text{EC} ) [22]</td>
<td>Error Cost: your organization’s historical cost to fix errors after releasing new software to the customer, in euros per error</td>
<td>1,036.38</td>
</tr>
<tr>
<td>( \text{NMSI} )</td>
<td>New Module Source Instruction: the LOC that the changed or new module has, which can be the average of previous ones</td>
<td>1,526</td>
</tr>
<tr>
<td>( \text{PC} ) [15]</td>
<td>Propagation Cost: the percentage of code affected in the RA when performing evolutions (i.e., changing modules)</td>
<td>9.7 %</td>
</tr>
<tr>
<td>( \text{CPL} ) [22]</td>
<td>Cost per LOC: the historical cost to develop a LOC of new software in your organization</td>
<td>2.5 €</td>
</tr>
<tr>
<td>( \text{USI} )</td>
<td>Unique Source Instructions: the amount of unique software (i.e., not reused) that was written or modified for an application</td>
<td>2,885</td>
</tr>
<tr>
<td>( \text{RSI} ) [22]</td>
<td>Reused Source Instructions: it is the total LOC of the RA’s modules that are reused in an application</td>
<td>8,364</td>
</tr>
<tr>
<td>( \text{TSI} ) [22]</td>
<td>Total Source Instructions: it is the total LOC of the RA that can be reused</td>
<td>41,189 LOC</td>
</tr>
<tr>
<td>( \text{DCA} ) [22]</td>
<td>Development Cost Avoidance: the benefits from reusing RA’s modules ( \Rightarrow \text{DCA} = \text{RSI} \ast (1-\text{RCR}) \ast \text{CPL} )</td>
<td>19,579 €</td>
</tr>
<tr>
<td>( \text{ADC} ) [22]</td>
<td>Additional Development Costs: the costs to develop the RA ( \Rightarrow \text{ADC} = (\text{RCWR}-1) \ast \text{TSI} \ast \text{CPL} )</td>
<td>151,213 €</td>
</tr>
<tr>
<td>( \text{UDC} )</td>
<td>Additional Unique Development Costs: the costs to develop the unique part of an application ( \Rightarrow \text{UDC} = \text{USI} \ast \text{CPL} )</td>
<td>7,212 €</td>
</tr>
<tr>
<td>( \text{SCA} ) [22]</td>
<td>Service Cost Avoidance: the benefits from maintaining only once RA’s modules ( \Rightarrow \text{SCA} = \text{RSI} \ast \text{ER} \ast \text{EC} )</td>
<td>17,467 €</td>
</tr>
<tr>
<td>( \text{AMC} )</td>
<td>Additional Maintenance Costs: the cost of fixing bugs in the (reusable) RA modules ( \Rightarrow \text{AMC} = \text{TSI} \ast \text{ER} \ast \text{EC} )</td>
<td>86,019 €</td>
</tr>
<tr>
<td>( \text{AEC} )</td>
<td>Additional Evolution Costs: the costs of changing or adding a new functionality and maintaining it to the RA ( \Rightarrow \text{AEC} = \text{evolution development} + \text{evolution maintenance} + \text{propagation} = (\text{NMSI} \ast \text{CPL}) + (\text{NMSI} \ast \text{ER} \ast \text{EC}) + (\text{TSI} \ast \text{CPL} \ast \text{PC}) )</td>
<td>17,028 €</td>
</tr>
</tbody>
</table>
IV. PRELIMINARY VALIDATION

To assess the feasibility of the economic model, we conducted a retrospective analysis of a particular case, in which we calculated the costs and benefits (and hence the ROI) of adopting an RA in the IT department of a public organization. This adoption was driven by Everis so that we applied the implementation of the model described in Section III.B.

By the time we performed the validation, the public organization had already: (1) adopted an RA, (2) created an application using the RA—which we consider “exemplar” application—and (3) fixed errors discovered in the RA software elements that were reused by the application.

The validation consisted of 3 parts. First, a post-mortem analysis in which our challenge was to extract metrics (see Table I) from already collected data. We calculated:

- ADC, RA initial investment, which lasted 6 months.
- DCA, the benefit of reusing code from the RA in the development of the exemplar application.
- SCA, the cost from fixing the errors of the reused code in the exemplar application.
- UDC, the cost of developing the application.

The above costs were accurately computed because this organization keeps track of project activities with their invested time. To identify the benefits, we interviewed application developers to estimate the costs of developing applications without using the RA. When there was no data, we asked the involved stakeholders as a light-weight solution [23].

Second, it was necessary to estimate the rest of factors:

- AMC, the cost of fixing all bugs in RA code. Since we knew the SCA for the exemplar application and the percentage of reuse, we calculated the error rate and error cost, which we used to estimate AMC.
- AEC, the cost of: (1) changing or developing a module with new functionality, (2) fixing its bugs, (3) making changes in the rest of the RA to integrate it.

Third, we constructed a business case for 3 years starting when the organization decided to adopt the RA, in order to calculate the ROI. For the first 8 months of those 3 years, we have real data about the RA development and the exemplar RA-based application. To estimate the costs and benefits for the rest of these 3 years, after some additional interviews, we have made the following assumptions:

- Future applications will have similar characteristics and complexity as the exemplar one.
- The public organization will develop 8 applications per year. Since the RA creation lasted 6 months, the first year they will develop just 4 applications.
- The totality of AMC is computed in the first year.
- A module is evolved (with new functionality) or added to the RA every year starting the second year.

Under these assumptions, the costs and benefits for the future can be calculated as shown in Table II. Table III summarizes the total costs and benefits for these 3 years. As we can see in Table III, this organization will realize a ROI within 2 years through gains in systematic reuse and application maintainability. The ROI after 3 years is 78%.

<table>
<thead>
<tr>
<th>Year</th>
<th>Total benefit</th>
<th>Total cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4*(DCA+S)</td>
<td>ADC+A+4*S</td>
</tr>
<tr>
<td>2</td>
<td>8*(DCA+S)</td>
<td>8<em>S+A+4</em>S</td>
</tr>
<tr>
<td>3</td>
<td>8*(DCA+S)</td>
<td>8<em>S+A+4</em>S</td>
</tr>
</tbody>
</table>

TABLE II. DESIGN OF THE BUSINESS CASE
TABLE III. RESULTS OF THE BUSINESS CASE

<table>
<thead>
<tr>
<th></th>
<th>Year 1</th>
<th>Year 2</th>
<th>Year 3</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total benefit</td>
<td>148,190 €</td>
<td>296,379 €</td>
<td>296,379 €</td>
<td>740,948 €</td>
</tr>
<tr>
<td>Total cost</td>
<td>266,083 €</td>
<td>74,729 €</td>
<td>74,729 €</td>
<td>415,540 €</td>
</tr>
<tr>
<td>Net cash flow</td>
<td>-117,894 €</td>
<td>221,651 €</td>
<td>221,651 €</td>
<td>325,408 €</td>
</tr>
<tr>
<td>Cumulative cash flow</td>
<td>-117,894 €</td>
<td>103,757 €</td>
<td>325,408 €</td>
<td></td>
</tr>
</tbody>
</table>

Data collection in this exemplar case was supported by the adoption of good practices with tool support: activity tracking with JIRA\(^1\) (keeping track of activities and their invested time); test-driven development with Sonar\(^2\) that informed about basic software metrics and percentages of tests compliance; and continuous integration with Jenkins\(^3\). The adoption of these practices and tools to collect data is the basis for moving software development from its usual low-validity environment, to a high-validity environment. As stated by Erdogmus and Favaro \[9\], high-validity environments allow acquiring reliable expert intuition and performing an accurate cost-benefit analysis, in this case of RAs.

V. CONCLUSIONS & NEXT STEPS

Architecture improvements are extremely difficult to evaluate in an analytic and quantitative way, contrary to the business efficacy (sales, marketing, and manufacturing) \[6\]. Methods and models for changing this state of the practice are demanded.

This paper has opened the path on the area of using economic models for RA assessment. We think that this area has a significant impact not just for researchers but also for practitioners in software development and organization’s executives. We presented an economic model to translate measured or estimated data (i.e., metrics) into monetary terms (i.e., cost-benefit analysis). Then, we use them as the basis for analyzing the economic value of an RA (i.e., valuation) that is adapted by an organization in the pursuit of its business strategies. Thus, our work aligns with Erdogmus et al. vision on economic activities in software industry, that fall into 4 levels: metrics, cost-benefit analysis, valuation and business strategy \[8\].

We have conducted a preliminary validation to calculate the ROI of adopting an RA in a real organization. This organization will realize a return on their investment within two years through gains in systematic reuse and applications maintainability. The method presented is generic enough to be used when other quality attributes are prioritized by relevant stakeholders. The presented economic model allows quantifying the value that an RA of Type 2 or 4 (those designed with an intended scope of a single organization) \[3\] brings to an organization. Its strongest point is the integration of different metrics that complement each other evaluating several RA-relevant aspects. On the other hand, a potential weakness of this approach is that it does not look at risks.

As future work, we plan to enrich the economic model with more metrics (like technical debt) to analyze more quality attributes, and coping with metrics risk. Also, we want to use homogeneity metrics to discover the RA modules that bring more benefit to an organization \[7\].

\(^1\) http://www.atlassian.com/es/software/jira/overview
\(^2\) http://www.sonarsource.org/
\(^3\) http://jenkins-ci.org/
REFERENCES


Benefits and Drawbacks of Reference Architectures

Silverio Martínez-Fernández1, Claudia Ayala1, Xavier Franch1, Helena Martins Marques2

1 GESSI Research Group, Universitat Politècnica de Catalunya, Barcelona, Spain
{smartinez,cayala,franch}@essi.upc.edu
2 everis, Barcelona, Spain
hmartins@everis.com

Abstract. Reference architectures (RA) have been studied to create a consistent notion of what constitutes them as well as their benefits and drawbacks. However, few empirical studies have been conducted to provide evidence that support the claims made. To increase this evidence, this paper investigates the actual industrial practice of using RAs. The study consists of a survey with 28 stakeholders from everis, a multinational consulting company based in Spain. We report the findings and contextualize them with previous research.

Keywords. Software reference architecture, empirical software engineering

1 Introduction

Software reference architectures (RA) have emerged as an approach to guide the development, standardization and evolution of concrete software architectures for new systems [5]. As in [1], we refer to the definition of RA as stated by Bass et al.: “a reference model mapped onto software elements and the data flows between them”.

RAs have become widely studied and used in research and practice [1], as they are claimed to increase speed, reduce operational expenses and improve quality in software systems development mainly due to reuse [6]. Nonetheless, limited evidence exists to support these claims [7]. Therefore, the goal of this study is to investigate:

How practitioners perceive the potential benefits and drawbacks of RAs?

Industrial context. This study is part of an ongoing action-research initiative among everis and our research group, aimed to improve everis’ architectural practices. everis is a software consulting company that offers solutions for big businesses that provide a wide spectrum of services to their customers. The solution that everis provides them is based on the deployment of an RA in their company, from which concrete software architectures are derived and used in a wide spectrum of applications. In this context, everis commissioned our research group to systematically gather empirical evidence about the benefits and drawbacks of the adoption of RAs for their clients, in order to avoid just relying on anecdotal evidences.
2 Benefits and drawbacks of RAs from the Literature

We identified the following benefits (B) and drawbacks (D) from the literature:

- **B1** Standardization of concrete architectures of systems [1][4][5][7][8].
- **B2** Facilitation of the design of concrete architectures for system development and evolution [1][5], improving the productivity of system developers [3][4][8].
- **B3** Systematic reuse of common functionalities and configurations in systems generation [2][4][5], implying shorter time-to-market and reduced cost [3][4].
- **B4** Risk reduction through the use of proven and partly prequalified architectural elements [2][4].
- **B5** Better quality by facilitating the achievement of quality attributes [3][8].
- **B6** Interoperability of different systems [2][4][5].
- **B7** Creation of a knowledge repository that allows knowledge transfer [2][7].
- **B8** Flexibility and a lower risk in the choice of multiple suppliers [2].
- **B9** Elaboration of the organization mission, vision and strategy [2].
- **D1** The need for an initial investment [6].
- **D2** Inefficient instantiation for the organization’s systems [5].

3 Benefits and Drawbacks of RAs from our Study

9 RA projects executed in 9 different organizations that were clients of everis were analyzed. 28 stakeholders from these projects participated in the study. They covered 3 essential roles: 9 software architects and 9 architecture developers that designed and implemented RAs for the 9 client organizations; and 10 application builders who created RA-based applications. We report the benefits and drawbacks of RAs for the development of systems as seen by these practitioners. Fig. 1 includes a bar chart that shows the frequency in which stakeholders mentioned RA benefits and drawbacks. The reader is encouraged to see how this study was conducted in www.essi.upc.edu/~gessi/papers/ecsa13-annex.pdf.

**Main benefits of RA adoption.** We report benefits for RA acquisition organizations with the code “Ben” whereas we use the code “Ven” for the benefits to RA vendors.

- **(Ben-A)** Reduced development costs. Mainly due to software component reuse that facilitate functionality and speed up the process, leading to shorter time-to-market.
- **(Ben-B)** Reduced maintenance costs. Because of: better understandability of systems derived from the RA; the fact that RA common elements have fewer errors.
- **(Ben-C)** Easier development and increased productivity of application builders by architecturally-significant requirements already addressed and RA artifacts.
- **(Ben-D)** Incorporation of latest technologies, which among other things facilitates the recruitment of professionals with the required technological skills.
- **(Ben-E)** Applications more aligned with business needs.
- **(Ben-F)** Homogenization (or standardization) of the development and maintenance of a family of applications by defining procedures and methodologies.
• (Ben-G) Increased reliability of RAs software elements, which are common for applications, that have been tested and matured, with the reliability that it implies.
• (Ben-H) Others benefits.
• (Ven-A) The consulting company harvests experience for prospective RA projects. The main reason is that requirements are very similar between client organizations.
• (Ven-B) Reusing architectural knowledge can speed up prospective RA projects and reduce time-to-market (e.g., by reducing their planning and development time).
• (Ven-C) They gain reputation for prospective client organizations and gain organizational competence.
• (Ven-D) Previous experience reduces the risks in future projects because a “to-be” model exists. It can be used in projects without very specific requirements.
• (Ven-E) It provides a shared architectural mindset.
• (Ven-F) It turns tacit knowledge into explicit knowledge in the reference model. Some tool support (e.g., wiki technologies) helps in managing such knowledge.

Main drawbacks of using RAs.
• (Dra-A) Additional high or medium learning curve for using the RA features.
• (Dra-B) Limited innovation by giving prescriptive guidelines for applications.
• (Dra-C) Applications’ dependency over the RA. When applications have requirements that the RA does not offer yet, applications development is stopped.
• (Dra-D) Complexity. Participants who indicated that the use of the RA is complex.
• (Dra-E) None. Responders who indicated that RA adoption presents no drawbacks.
• (Dra-F) Wrong decisions about the technologies to be used in all the applications.
• (Dra-G) Other drawbacks.

4 Discussion of Main Findings and Conclusions

Table 1 summarizes the benefits and drawbacks of RAs respectively. Its columns respectively indicate: 1) benefits or drawbacks from the literature and uncovered ones by our survey that we could not match to the former ones; 2) the extent to which the results from our survey confirm (✓), partially support or help to understand (±), do not explicitly mention (°), refuse theoretical evidences (×) or uncover new results (new); and 3) survey findings related to such benefits or drawbacks.

In conclusion, a survey was conducted to analyze benefits and drawbacks of RAs in the industrial practice. It provides evidence to corroborate or refuse existing research. The main findings were: 1) the support of already known RAs benefits, mainly cost savings in the development and evolution of software systems, and the facilitation of the design of concrete software architectures; 2) new risks of adopting RAs emerged, such as additional learning curve, less room for innovation and complexity. As future work, we plan to perform further analysis of this survey.

Acknowledgements. This work has been supported by “Cátedra everis” and the Spanish project TIN2010-19130-C02-00. We thank all participants of the survey.
Fig. 1. Benefits and Drawbacks of RA adoption in organizations as seen by practitioners.

Table 1. Summary of benefits and drawbacks of RAs.

<table>
<thead>
<tr>
<th>Benefits</th>
<th>D1</th>
<th>Findings</th>
<th>Drawbacks</th>
<th>D1</th>
<th>Findings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standardization (B1)</td>
<td>√</td>
<td>Ben-F</td>
<td>Investment (D1)</td>
<td>±</td>
<td>Dra-G</td>
</tr>
<tr>
<td>Facilitation (B2)</td>
<td>√</td>
<td>Ben-C</td>
<td>Inefficient instantiation (D2)</td>
<td>±</td>
<td>Imp-C</td>
</tr>
<tr>
<td>Reuse (B3)</td>
<td>√</td>
<td>Ben-A, Ben-B,</td>
<td>Learning curve</td>
<td>new</td>
<td>Dra-A</td>
</tr>
<tr>
<td>Risk reduction (B4)</td>
<td>√</td>
<td>Ben-G, Ven-D</td>
<td>Limited innovation</td>
<td>new</td>
<td>Dra-B</td>
</tr>
<tr>
<td>Enhanced quality (B5)</td>
<td>±</td>
<td>Ben-E</td>
<td>RA dependency</td>
<td>new</td>
<td>Dra-C</td>
</tr>
<tr>
<td>Interoperability (B6)</td>
<td>°</td>
<td></td>
<td>Complexity</td>
<td>new</td>
<td>Dra-D</td>
</tr>
<tr>
<td>Knowledge repository (B7)</td>
<td>√</td>
<td>Ven-A, Ven-F</td>
<td>Wrong decisions</td>
<td>new</td>
<td>Dra-F</td>
</tr>
<tr>
<td>Flexibility for suppliers (B8)</td>
<td>±</td>
<td>Ven-E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mission, vision, strategy (B9)</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Latest technologies used</td>
<td>new</td>
<td>Ben-D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reputation</td>
<td>new</td>
<td>Ven-C</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a. Notes: 1) diagnostic; 2) not mentioned as a benefit; 3) mentioned as enterprise architecture benefit.

References