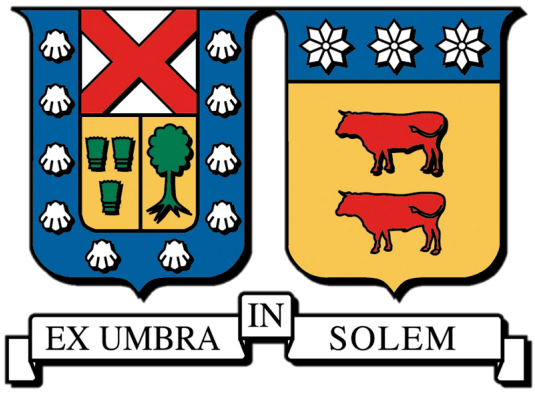
Universidad Técnica Federico Santa María
Departamento de Informática
Valparaíso, Chile



The Essentials of the Software Architecture Evaluation Practice: A SEMAT-based Representation

Pablo Cruz Navea

a thesis submitted in partial fulfillment
of the requirements for the degree of

doctor en ingeniería informática

Main Advisor:
Dr. Mauricio Solar F.
Co-Advisor:
Dr. Hernán Astudillo R.

September 16, 2026

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☐ Memoria o trabajo de título; ☒ Tesis de Postgrado;

Título del trabajo: The Essentials of the Software Architecture Evaluation Practice: A SEMAT-based Representation

Nombre del candidato(a): Pablo Cruz Navea

Carrera / Grado: Doctorado en Ingeniería Informática

Campus: Casa Central Valparaiso **;** **Departamento**: Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Mauricio Solar Fuentes, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: ; **Firma:** _____

Estudiante o Candidato(a):

Fecha: 28/08/2026 ; **Firma:** _____

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

EVALUATION COMMITTEE

Dr. Mauricio Solar F. — *Main Advisor*
Universidad Técnica Federico Santa María, Chile.

Dr. Hernán Astudillo R. — *Co-advisor*
Universidad Andrés Bello, Chile.

Dra. Elizabeth Montero U. — *Internal Member, President of the Committee*
Universidad Técnica Federico Santa María, Chile.

Dr. Samuel Sepúlveda C. — *External, National Member*
Universidad de La Frontera, Chile.

Dr. J. Andrés Díaz-Pace. — *External, International Member*
ISISTAN-CONICET / UNICEN, Argentina.

"Recién entonces nos percatamos de una gran diversidad que se nos presenta como plétora desconcertante y nos obliga a separar, clasificar y reagrupar. Así surge finalmente un orden que podemos abarcar más o menos satisfactoriamente."

— Johhan Wolfgang von Goethe, *Teoría de los Colores* (1810)

A mi familia.

Resumen

La evaluación de arquitecturas de software es una práctica clave en el aseguramiento de la calidad de un sistema de software. Desafortunadamente, la experiencia y la literatura muestran que la ejecución de esta práctica no siempre es tan directa y fluida como se desearía. Tanto la literatura como las opiniones profesionales coinciden en que se necesita más ayuda concreta para ejecutar una evaluación de este tipo.

Este trabajo propone un conjunto de cinco elementos esenciales de la práctica de evaluar arquitecturas de software: atributos de calidad, descripción de la arquitectura, decisisones de arquitectura, objetivos del negocio, y adopción de la evaluación. Estos elementos esenciales han sido modelados y representados como Alphas, usando el Kernel de SEMAT como *framework* de modelamiento, haciendo que cada uno de estos se caracterice en términos de una composición de etapas discernibles que progresan desde niveles iniciales hasta la compleción de la ejecución de las evaluaciones de arquitectura en contextos reales. Se argumenta en esta tesis que la atención de estos Alphas propuestos, en las evaluaciones de arquitectura, facilita que se ejecuten de forma guiada y controlada.

El uso de la propuesta en evaluaciones de arquitectura de software en contextos reales, con sistemas de software de tipo legado en instituciones de gran tamaño y cuyos enfoques de desarrollo son de tipo tradicional con uso de prácticas ágiles de desarrollo, ha permitido levantar evidencia para argumentar que los Alphas son apropiados para ayudar a los profesionales en guiar y evaluar/auditar el progreso de las evaluaciones de arquitectura de software. Además, un conjunto distinto de profesionales del área de la arquitectura de software fueron contactados para presentarles la propuesta y levantar retroalimentación por medio de una encuesta cuyos resultados permiten observar que los profesionales reconocen que aspectos como ordenar el proceso de la evaluación, mejorar la planificación, la definición de tareas, roles y productos de trabajo concretos y la objetividad son algunas de las características

beneficiadas.

Se espera que, en el futuro, un uso más amplio permita a tanto practicantes profesionales como investigadores del área definir con mayor concreción sus evaluaciones de arquitectura de software, así como guiarlas y evaluar la progresión de estas. Así también, la utilización de la propuesta podría permitir eventualmente la transmisión de conocimientos de la práctica en el entrenamiento profesional y en el aula universitaria, permitiendo que los organizadores e instructores de cursos orienten el material de sus clases de evaluación de arquitecturas de software en torno a los cinco elementos esenciales, así como la progresión de estos hacia una evaluación terminada.

Palabras clave: Arquitectura de software, Kernel de SEMAT, Estándar Essence, Ingeniería de software.

Abstract

Software architecture evaluation is a key practice in assuring the quality of a software system. Unfortunately, experience and literature show that executing this practice is not always as straightforward and seamless as desired. Both literature and professional experience agree that more concrete support is needed to enable actionable execution of the evaluations.

This work proposes a set of five essential elements of the software architecture evaluation practice: quality attributes, architecture description, architectural decisions, business goals, and evaluation adoption. These essential elements have been modeled and represented as Alphas, using the SEMAT Kernel as a modeling framework, with each element characterized in terms of a progression of discernible stages, from initial levels to the completion of the execution of architecture evaluations in real-world contexts. This thesis argues that addressing the proposed Alphas, when evaluating architectures, facilitates these efforts to be conducted in a guided and controlled manner.

The application of this proposal in real-world software architecture evaluations—specifically on large-scale legacy systems within major institutions that utilize traditional development approaches alongside agile practices—has provided evidence to argue that these Alphas are appropriate for helping professionals guide and audit the progress of software architecture evaluations. Furthermore, a separate group of software architecture professionals was contacted to review the proposal and provide feedback through a survey. The results indicate that professionals recognize benefits such as maintaining order in the evaluation process, improving planning, defining specific tasks, roles, and work products, and increasing objectivity.

It is expected that, in the future, broader adoption will allow both professional practitioners and researchers in the field to define software architecture evaluations with greater concreteness, as well as to guide them and assess their progression. Additionally, utilizing this proposal could eventually facilitate knowledge transfer in professional training and univer-

sity classrooms, enabling course organizers and instructors to align their software architecture evaluation teaching resources around the five essential elements and their progression toward completing an evaluation.
Keywords: Software architecture, SEMAT Kernel, Essence Standard, Software engineering.

List of Abbreviations

ADL	Architecture Description Language.
ADR	Architecture Decision Record.
Alpha	Abstract-Level Progress Health Attribute.
ATAM	Architecture Tradeoff Analysis Method.
BPMN	Business Process Model and Notation.
CMMI	Capability Maturity Model Integration.
DCAR	Decision-Centric Architecture Reviews.
IEC	International Electrotechnical Commission.
IEEE	Institute of Electrical and Electronics Engineers.
ISO	International Standards Organization.
NDA	Non-disclosure Agreement.
OMG	Object Management Group.
PBAR	Pattern-based Architecture Reviews.
SAAM	Software Architecture Analysis Method.
SARA	Software Architecture Review and Assessment.
SBAR	Scenario-based Re-engineering.
SEMAT	Software Engineering Method and Theory.
SysML	Systems Modeling Language.
UML	Unified Modeling Language.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A software architecture is decisive for achieving software quality [1, 2, 3]. Quality goals for a software system are the key concerns that shape both the architecture design process and the architecture itself [4]. At the same time, when a software architecture is designed, the constraints of many quality attributes are determined by the chosen architecture [5]. Although software architecture is not the sole determinant of software quality, its criticality comes from the fact that any attempt to improve quality after a poorly designed architecture is often futile [6]. An inadequate software architecture design will also hinder the adoption of modern software engineering approaches such as continuous software engineering due to the lack of support for related practices, notably test and integration automation [7], rapid delivery [8], and continuous software evolution [9].

The evaluation of a software architecture plays a vital role in software quality [10] at design time [11]. An architecture evaluation is used to identify architectural risks [6] and to determine if quality requirements are addressed in the architecture design [12]. Experience shows that evaluating software architectures is an integral part of software architecture design [13, 14], although it is commonly carried out in an informal way [15]. The importance of evaluating a software architecture has been stressed even in agile development [16], where the rapid delivery of software features often competes with long-term quality goals [17].

1.1 Problem Statement

The importance of evaluating a software architecture does not always go hand in hand with an actionable practice. Challenges appear in the evaluation itself because of the gap between understanding an evaluation method and bringing it into practice. For example, the well-known Architecture Tradeoff Analysis Method (ATAM) is mentioned as having a steep learning curve for practical use [18]. In addition, a practitioner-oriented book on software architecture remarks on the lack of real-world focus of some of the methods for evaluating architectures [19]. These issues make the practice of systematically evaluating a software architecture too dependent on personal experience [20].

The software architecture research community has acknowledged this and has made efforts to ease the application of architecture evaluations in the real world. For example, the Decision-Centric Architecture Reviews (DCAR) method [21] emerged as a proposal to overcome the "heaviness" of methods such as ATAM.

1.2 Thesis Proposal

I have been working in the software architecture evaluation domain for several years, performing architecture evaluations for many types of software systems and in various organizational contexts; some of these experiences have been published in peer-reviewed venues [22, 23, 24, 25]. By adopting a research-oriented reflection-in-action approach [26] in such endeavors, I identified the key elements that constitute the software architecture evaluation practice. In this thesis work I propose a representation of this practice using the SEMAT Kernel as a modeling framework. The adoption of this framework commits to the representation of a software practice as a set of "essential elements", each progressing through a set of discernible statuses.

This thesis aims to answer the following research questions:

- What are the essential elements that need to be considered when evaluating a software architecture?
- What are the discernible progression states of these essential elements that can be used to assess the progression of the evaluation of software architectures?

The essential elements of the software architecture evaluation practice that I propose are:

- Quality Attributes.
- Architecture Description.
- Architecture Decisions.
- Business Goals.
- Evaluation Adoption.

The SEMAT Kernel [27, 28] provides both a framework for describing software practices in terms of the "essential elements" that practitioners must work with, and a notation for representing the practices [29]. In the SEMAT Kernel, these "essential elements" are represented as Alphas and the progression of each Alpha (i.e., the discernible statuses) is represented as a set of States. The identification of the Alphas and their States is a process that is colloquially known as "essentializing a practice" [29][1].

A key salient characteristic of the SEMAT Kernel is that it is defined as an actionable, extensible, and practical thinking framework for the description of software engineering practices [27]. The extensibility of the Kernel means that in addition to the standard set of Alphas of software engineering, more Alphas can be added to the Kernel to define more concrete software practices.

I have explored the suitability of the SEMAT Kernel notation to express the architecture evaluation practice, where a conceptual validation was provided [30]. Its use in several actual architecture evaluations allowed me to find several key issues, mostly related to the progression of the Alphas; for example, the State Working well in the Alpha Evaluation Adoption [30] was not consistent with how activities are done in an architecture review, thus a better articulation for progression from State 2 (Method integrated and in use) to State

[1]The use of the word "essential" emerges from two facts. First, the SEMAT Kernel assumes that an Alpha or "essential" is an element that must be attended when using the practice that is described in terms of these Alphas. For example, in my proposal I say that any architecture evaluation must attend the five Alphas or "essentials" that were identified. Second, the SEMAT Kernel eventually became an OMG (Object Management Group) standard and its name was chosen to reflect this: Essence Standard [28].

3 (Working well) had to be reworked. Eventually, the research focus shifted from testing a notation to modeling the software architecture evaluation practice.

Two of the various real-world experiences in which this proposal has been used are reported in this thesis document with a case study research approach (see Chapter 6). In addition, the proposal was also presented to software architecture and software engineering practitioners who were asked to rate its suitability along several aspects (see Chapter 7).

1.2.1 Thesis Hypothesis

The main hypothesis of this work is *"the progression through the proposed Alphas and States leads to completing, and assessing the progression health of a software architecture evaluation endeavor."*

The specific hypotheses derived from the main hypothesis are:

- The proposed model leads people with experience in software architecture (including evaluation) to better assess a software architecture evaluation endeavor.
- The proposed model leads people with no prior experience in software architecture (including evaluation) to better understand the progression growth path (i.e., what is required to do) in a software architecture evaluation endeavor.

This thesis document summarizes the identification of the essential elements of the software architecture evaluation practice, the progression path of each element, and evidence gathered from case studies and a survey to support the claim that this representation is appropriate for completing and assessing the progression health of a software architecture evaluation.

1.3 Contribution

This work contributes to both the software architecture practitioner and research communities with an actionable model for guiding and assessing the progression of software architecture evaluations, based on the identification of the essential elements (i.e., Alphas) of software architecture evaluations, each characterized by a set of progression steps (i.e., States). This

definition can also be used to guide and audit the progression of software architecture evaluations. This work also contributes with a graphical SEMAT-based representation of this mechanism as Alpha State Cards, which has been made available on GitHub (see [31]). These cards provide a concrete description for enacting an architecture evaluation. Other preliminary results have shown an expedient use of the proposal for organizing software architecture evaluation teaching material, as well as a valuable aid in learning the architecture evaluation practice.

1.4 Challenges and Threats to Validity

The main challenge for a thesis like this one is the lack of control of the many variables that surround the empirical experiences that are used to find some evidence supporting or contradicting the claims. Often, the empirical experience is "contaminated" by several factors that are not negotiable, for example, the use of a particular method in addition to the method being tried, the use of specific tools, the selection of participants, among others. While these factors can be set in light of the experience in a controlled research environment such as a case running in a university classroom, in a real-world case these factors are strongly influenced by the industrial context and the researcher has little to no influence on setting these variables.

The use of the case study approach is not always a matter of design. It is often a consequence of the lack of control that makes it difficult if not impossible to use other designs such as traditional experiments [32]. In light of the alternatives "run the experience" versus "do not run the experience", trade-offs must be assumed, and a case study emerges as the appropriate empirical enquiry to "run the experience" and to collect (mostly) observational evidence to support claims.

This "main challenge" can be articulated into many sub-challenges. To mention a few:

- Conducting a software architecture evaluation with a research-oriented goal in a real-world context means that the institution have to grant access to their artifacts, processes, and other elements to people from the outside (i.e., the researcher). This implies that some kind of non-disclosure agreement (NDA) must be signed. The non-disclosure agreement might eventually hinder the "publishability" of the results. One thing I have observed is that the research community often lacks awareness of the fact that software

engineering and software architecture processes, work products, results, and data are industrial assets and companies are reluctant to provide them for free[2].

- Organizations run their own methods and practices. These methods and practices define boundaries for the software architecture evaluation. An organization can accept conducting an architecture evaluation in a real-world case, but at the same time the organizations frequently require that such endeavor run bounded to their own methods and practices. For a research this means that some variables will be set outside the desire of the researcher.

- In some cases, the first time an organization runs an architecture evaluation is when this proposal is presented. In such cases, the risk of falling in the so-called "post-hoc" fallacy (i.e., when the claims attributed to the use of the proposal might have been observed just because the order of the events: from no experience, to a first experience) is a matter to consider. In this case, the only element I have as a researcher is to gather contextual evidence to support the claim.

- The selection of subjects, which is a key aspect of experimental designs is hardly achievable in real-world contexts. In an ideal case, one could classify subjects according to some criteria. For example, two groups: one with people experienced in architecture evaluation and one with people with no experience in evaluating architectures. This is far from being achievable in a real-world context in which software engineering teams are often relatively small and specialized in some software products. To make matters worse for research, there is often no effort available to run two experiences with different groups.

- While this kind of empirical experiences are by no means human-based experimentation, the sole inclusion of people in the empirical experience makes ethics and moral aspects come to the surface. In one case, I was rejected in conducting an architecture evaluation with a research-oriented mindset because I had no formal approval from an ethics committee. Although this was not a common issue, I recommend that people who work in this kind of research consider in advance ethics and moral aspects.

[2]However, in rare cases I have observed that companies are open to publish their assets in exchange of the eventual product and product's qualities publicity that a published paper could provide.

These challenges not only constrain the design of this work, but also affect the validity of the claims. Chapter 8 is especially focused on the analysis of such threats to the validity of the claims as well as explaining how such risks were accepted and/or mitigated.

1.5 Main Results

I consider important to highlight two categories of results in this thesis work:

- The results that are considered as products, that is, the concrete usable work products that are delivered for practitioners and researchers to use.
- The empirical results, that is, the observable effects when using the proposal of this thesis in real-world cases, and the results obtained from the answers of software architecture practitioners to whom this proposal was presented.

The main result of this thesis is the representation of the software architecture evaluation practice as a set of essential elements that progress towards the completion of the enactment of the practice. These elements are represented as Alphas and States because the SEMAT Kernel was chosen as the modeling framework.

In addition, several sub products are results of this thesis:

- A proposed extension for the standardized SEMAT Kernel (Essence) to recognize the software architecture evaluation practice, including Alphas, States, and Activity Spaces.
- A set of Alpha State Cards that practitioners and researchers can use.

In addition to these products, research results include the following:

- Real-world evidence gathered from two case studies in which the proposal was found to be appropriate to guide a software architecture evaluation in the context of legacy systems in classic development styles.
- Survey-based evidence from software architecture and software engineering practitioners that suggest that the proposal is helpful for aspects that influence the guiding and progression assessment and audit.

1.6 Thesis Outline

The remainder of this thesis document is structured as follows:

- Chapter 2 surveys the background and related work, including the SEMAT Kernel.
- Chapter 3 describes the research strategy.
- Chapter 4 introduces the proposed Alphas that extend the SEMAT Kernel to represent the software architecture evaluation practice, as well as a brief discussion on how to use this proposal for guiding and assessing the progression of a software architecture evaluation.
- Chapter 5 strengthens this proposed extension of the SEMAT Kernel by introducing new Activity Spaces.
- Chapter 6 presents two case studies of the proposal in real-world architecture evaluations.
- Chapter 7 presents the results of a survey about the utility of this proposal.
- Chapter 8 discusses the main threats to validity.
- Finally, Chapter 9 summarizes and concludes this thesis document.

Chapter 2

Background and Related Work

2.1 Software Architecture Evaluation

There is a widely agreed assumption that the quality of a software system is greatly determined by the high-level design embodied in its software architecture [1, 5, 4, 2]. Therefore, evaluating a software architecture is a key practice in achieving software quality.

In general, a software architecture evaluation aims to identify risks in the proposed architecture [6, 12, 33] and to determine whether quality concerns and quality requirements have been addressed at the architecture level [1, 4, 15].

Rather than answering "yes" or "no" to a general question about the suitability of the architecture for the purpose of the system, an architecture evaluation typically reports risks considering the purpose of the system [6] and provides concrete evidence on the degree to which the system meets its quality criteria [34].

Experience shows that architecture evaluation is a key practice when designing software architectures in the industry [13], although it is frequently carried out informally [15].

Two general approaches are used to evaluate design at the architecture level: *questioning*, and *measuring* [12]. When using a *questioning* approach, the evaluation is based on a set of qualitative questions that are applied to a software architecture to assess how well the design supports the question. Examples of the questioning approach are the use of scenarios, questionnaires, and checklists [12]. On the other hand, the *measuring* approach includes the use of metrics, simulations, prototypes, and experiences [12].

There is an evident trade-off between the two approaches: applicability. Questioning techniques are applicable to assess the architecture in light of any quality attribute [12]. Measuring techniques bring more specific answers and, therefore, they are applicable to a more specific set of qualities [12].

An architecture evaluation can be done in a systematic and repeatable way by following an evaluation method; depending on the method's intention, it adopts one or another of the aforementioned approaches and techniques. For example, the Architecture Tradeoff Analysis Method (ATAM) [6] uses the scenario technique to operationalize the quality attributes that will be *questioned* in the evaluation (the method goes further by not only using scenarios to *question* the architecture in light of quality criteria, but also analyzing how these quality goals interact, or "trade off", between them). Specific measure-based evaluations can be designed; for example, [35] evaluates the *resilience* of self-adaptive systems, using a simulation-based technique.

Although a software architecture can eventually be evaluated in an ad hoc way, the use of a method supports repeatable analysis [36]. As of this writing, there are several methods for evaluating an architecture, each embracing a specific approach. The Software Architecture Analysis Method (SAAM) [34] uses scenarios to characterize the circumstances in which the software system will be used to assess how well the proposed software architecture will satisfy the specifications expressed in the scenarios. The method works by *questioning* the architecture how well it supports elicited scenarios, distinguishing between direct scenarios (i.e., those that are directly supported by the architecture) and indirect scenarios (i.e., those that require some changes in the architecture to support them).

The Scenario-based Re-engineering Method (SBAR) [37] presents a reengineering approach using a scenario-based analysis to assess the current architecture and decide if a transformation is required; unlike the previous methods, SBAR highlights in its openness to embrace other approaches for architecture assessments [37].

The Decision-Centric Architecture Reviews Method (DCAR) [21] takes a different approach: it follows a decision-based architecture evaluation where the software architecture being evaluated is characterized as a set of design decisions that are assessed against potential cases in which the decision would be challenged [21].

The Pattern-Based Architecture Reviews Method (PBAR) [38] emerges as a lightweight pattern-based architecture assessment method; it proposes to examine the architecture to

identify the architecture patterns used and to determine if these patterns align with the quality goals of the software. Its authors argue that by focusing on identification of established patterns, the analysis is suitable for small and production-focused teams [38] where architecture documentation is expected to be sparse.

Running an architecture evaluation is deemed a challenging effort. Challenges appear in the evaluation itself because of the gap between understanding an evaluation method and bringing it into practice. Some authors [18] argue that ATAM has a too steep learning curve for practical use. Successful practitioner-oriented books on software architecture (e.g. [19]) highlight the lack of real-world focus on some of the methods for analyzing architectures, sometimes making the use of an evaluation method too dependent on personal experience [20].

Several efforts have been made to overcome these challenges. For example, experience reports (like [18, 39, 40, 41, 42]) are a well-known way to share experiences about architecture evaluations, the challenges the reviewers faced, and how they dealt with those challenges. Some authors [43] have researched the factors that influence the use and success of an architecture evaluation practice. Knodel and Naab's book [44] is an interesting development of the common phases for running software architecture evaluations; it condenses the results from many experiences that they were involved with. There are also "standard" sources, such as the "Software Architecture Review and Assessment (SARA) Report" by Obbink et al. [45] and the ISO/IEC/IEEE 42030:2019 "Software, systems and enterprise — Architecture evaluation framework" [46][47].

None of these efforts has presented a more actionable representation of architecture evaluations for practical use. An actionable representation should be concrete enough so that practitioners can understand the progression of architecture reviews and act upon the determined progression state.

2.1.1 Architecture Tradeoff Analysis Method

The Architecture Tradeoff Analysis Method is a well known architecture evaluation method. In addition, this method was used in the evaluation of Case Study 1 (see Chapter 6). Thus, the ATAM steps deserve more explanation.

The ATAM [6] prescribes the following steps:

1. Present the method: the method is presented to the evaluation team.

2. Present the business drivers: the business drivers that explain the opportunity that justifies a software system are presented.

3. Present the architecture: a first architecture description is presented, where the emphasis is on explaining how the current architecture follows business drivers.

4. Identify the architectural approaches: current architectural decisions are identified.

5. Generate the quality attribute tree: the quality attribute tree is an artifact that traces quality attributes to stakeholders' concerns and then to concrete scenarios. The quality attribute tree is prioritized.

6. Analyze architectural approaches: the identified architectural approaches are discussed in light of the quality attribute tree. In this step, architectural decisions are studied to determine if they favor or hinder achieving high priority scenarios.

7. Brainstorm and prioritize scenarios: more stakeholders join the architecture evaluation and scenario elicitation is open to a wider discussion.

8. Analyze architectural approaches: the architectural approaches are again analyzed, now considering the new scenarios elicited.

9. Present results: the execution of the method ends with a presentation of the results of the architecture evaluation.

Given that ATAM is strongly based on the use of scenarios, it is commonly regarded as an example of the scenario-based architecture evaluation methods [12].

2.2 The SEMAT Kernel and the Essence Standard

The SEMAT Kernel is one of the most prominent results derived from the "call for action" that Jacobson, Meyer and Soley initiated in 2009 [48], aiming at refounding the software engineering discipline based on a sound theoretical framework. The SEMAT Kernel aims at providing this theoretical framework by standardizing the essential elements that any software engineering endeavor should do and progress [49].

The SEMAT Kernel along its description language (both in graphical and textual forms) eventually became standardized in the OMG (Object Management Group) Essence Standard [29][28]. The description of a practice such as the evaluation of software architectures using the SEMAT Kernel is colloquially known as "essentializing" the practice [29] which it should be understood as using a common standardized way of expressing the practice rather that trying to change its nature.

The Kernel tries to achieve a good balance between being too specific, and therefore losing general applicability, and being too general, and therefore ending up in describing any kind of endeavor without considering software engineering ones [28]. To overcome this challenge, the Kernel is designed to be extensible [50], meaning that more specific practices can be described (or essentialized) using the elements that the Kernel provides.

In its base and standardized form, the SEMAT Kernel [28] defines seven "Alphas" that capture the key concepts in software engineering, serving as a common ground for expressing methods and practices. Each "Alpha" defines a series of States that allows practitioners to assess the progression and health of a software engineering endeavor. The seven "Alphas" are Opportunity (dealing with the justification for the software system to be developed or changed), Stakeholders (the people, groups or organizations that affect or are affected by the software system product and its engineering), Requirements (the expectations that stakeholders have with the system), Software System (the software system as a product considering software, hardware and data, to provide its primary value), Work (the mental or physical effort to achieve a result), Team (the people involved in the development, maintenance, delivery or support of the software system), and Way of Working (the set of practices and tools that are used by the team to guide and support their work).

In addition, the Kernel organizes the elements in three areas of concern: Customer (dealing with the use and exploitation of the software system, represented with color green, and encompassing the Alphas "Stakeholders" and "Opportunity"), Solution (containing everything related to the specification and development of the software system, represented with color yellow, and encompassing Alphas "Requirements" and "Software Ssystem"), and Endeavor (related to the teams and the way the teams approach their work, represented with color blue, and encompassing Alphas "Work", "Team", and "Way of Working").

The Kernel also defines the work that practitioners do in a software engineering endeavor. For the Kernel, this work is expressed as "Activities". The Kernel avoids the definition of

specific "Activities" [29]. Rather, it defines a set of "Activity Spaces" that act as placeholders to group specific "Activities" that practitioners are expected to fill. The current "Activity Spaces" are [28]: Customer Area of Concern: Explore Possibilities, Understand Stakeholders Needs, Ensure Stakeholders Satisfaction, Use the System; Solution Area of Concern: Understand the Requirements, Shape the System, Implement the System, Test the System, Deploy the System, Operate the System; Endeavor Area of Concern: Prepare to do the Work, Coordinate Activity, Support the Team, Track Progress, and Stop the Work.

As noted before, the Kernel is extensible and has been previously extended to describe the design of micro services architectures [51], to represent the test-driven development practice [52], and to guide and assess software architecture evaluations [53], to mention a few cases.

Chapter 3

Research Method

The research method used in this work consists of four main strategies (see Figure 3.1):

- The use of the reflection-in-action approach in real-world architecture evaluations.
- Literature research to find the state of the art in the software architecture evaluation domain.
- A case study approach for empirical experiences.
- A survey to collect primary, first-hand data from a set of practitioners different from the participants in the case studies.

This chapter is organized into two sections:

- Section 3.1 presents the method followed for the identification of the "essentials" (i.e., Alphas to be added in the SEMAT Kernel) and their use in two case studies.
- Section 3.2 presents the survey design considerations as well as the execution of the survey.

3.1 The identification of the "essentials" of a software architecture evaluation

In the first strategy, the architecture evaluation leader used the reflection-in-action [26] approach to get reflectively involved in the practical work of several architecture evaluations.

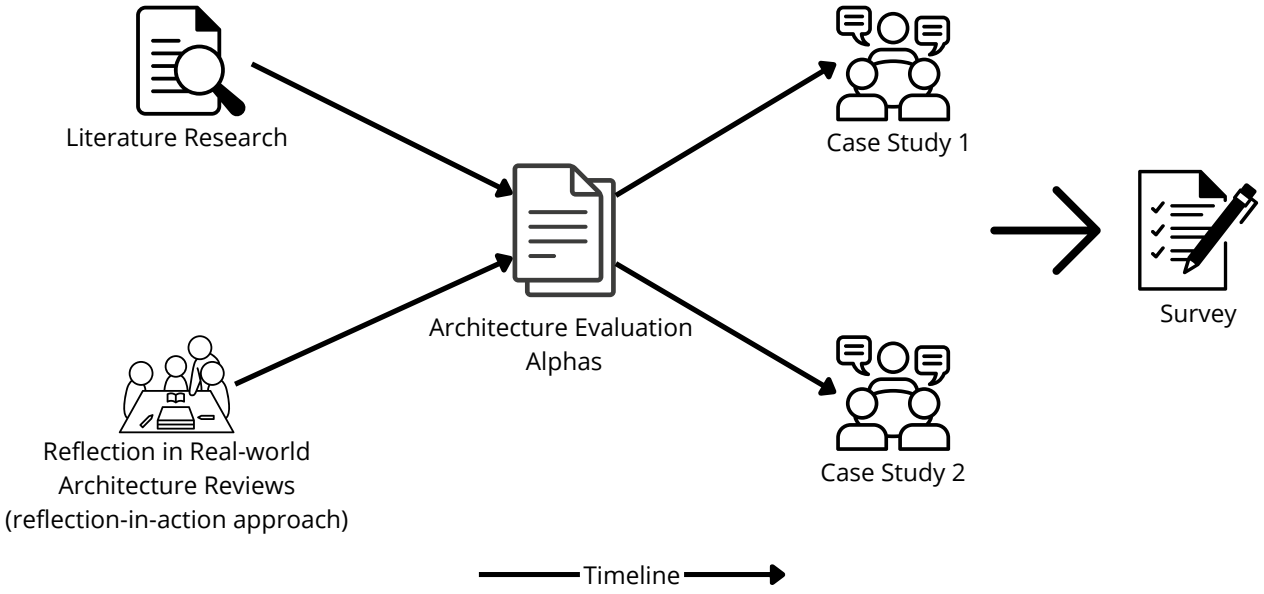


Figure 3.1: The research strategies used in the research method.

The reflection-in-action approach is frequently used in software engineering research [49][54]. In practical terms, a reflection-in-action approach means a continuous research-oriented review of the architecture evaluation practice-related activities. Literature research is used to support a solid ground for the domain of such evaluations. In a reflection-in-action approach the interest is in both etic and emic issues [55], that is, those key insights that serve as a conceptual structure for the study of the architecture evaluation phenomenon and those key insights that appear and emerge when running an architecture evaluation and reflecting in, and on the action, respectively. Etic and emic issues are deeper aspects that are of interest to researchers [55]. Retrospectives were used for this purpose, serving as more formal instances where the team reflected in, and on the architecture evaluation activities and their results. This first strategy was used to deeply study the architecture evaluation phenomenon in its real-world context and to define the essentials of such evaluations and their progression paths towards the completion of the evaluation effort.

The second strategy is the use of case studies. Case studies are empirical enquiries [32] that are commonly used in software engineering [32] when the phenomenon of interest is

strongly affected (although not necessarily determined) by the context [56], making other empirical methods hard to apply due to lack of experimental control [32]. Caution must be exerted for unnecessary research intrusion to avoid the case study from degrading into an artificial artifact [57]. Case studies typically make use of an observational approach which aims at collecting insights as the case goes through its execution [58]. The study of cases focuses on the interpretative understanding [56] of a real-world phenomenon by engaging in a "dialogue" with those who are part of the real-world context [56]. In this work, each case corresponds to an evaluation of a software architecture in a real-world context with a real software product.

Case studies can take an analytical generalization (rather than statistical) approach [59] as well as a naturalistic approach [56][60]. In the first approach, the case is predefined with an analytical generalization focus [59], trying to identify general and universal patterns [56]. In contrast, the naturalistic approach [56] is oriented toward bringing out the so-called emic issues, that is, the key insights that emerge from a case in a real-world context [55]. Therefore, a key aspect of a case study that takes the naturalistic approach is that the context in which the case is observed and studied is real world, that is, the ordinary setting and natural habitat of the case [56]. Case studies often take a narrative approach [61] to report the case. A narrative approach is also common in software engineering research, especially in practitioner-oriented research reporting [62].

A naturalistic approach was adopted because I had the opportunity to run two software architecture evaluations in two varied real-world contexts. The first case was the evaluation of a health record management system in a public cancer-treatment oriented hospital. The second case was the evaluation of a human resources software system in a military institution. For reporting the cases in this thesis, a narrative form was adopted with an emphasis on how the cases proceeded. The key insights that appeared in these cases are also discussed. The case study designs followed logistic aspects such as the planning of evaluation meetings, the use of meeting rooms with specific support tools such as modeler software and whiteboards, as well as meeting several "entrance requirements" because both institutions where the cases took place had privacy-sensitive data. In both cases, short retrospectives were used to iteratively evaluate the use of this proposal in the architecture evaluation. The many emic issues that appeared outside the retrospectives were also written down. The reporting of the two cases in this paper is followed by a discussion where key insights are highlighted, both

observed and reported by the evaluation participants.

This proposal has been used in several architecture evaluations. However, two of such evaluation experiences were chosen to be reported as case studies in this work. The rationale behind the selection of these two experiences as case studies in this thesis is summarized as follows:

- Criticality of the system for the organization: I believe that if a system is critical for the organization, the experience of evaluating their architectures would be closer to real-world situations. In my experience, stakeholders take a more serious stance in architecture evaluation if the system is critical to the organization. Both systems evaluated and reported in this article are considered critical to their organizations and therefore were chosen to be presented in the article.

- Size of the software system: the size of a software system is correlated with the size and complexity of the architecture. A bigger architecture gives more space for evaluation activities, and therefore I expected more issues to be observed and learned from evaluating bigger architectures. Both architecture evaluations presented in the article are considered big in terms of subsystems (both systems exhibiting more than 15 subsystems; most of these subsystems are software systems in themselves).

- Diversity of stakeholders: evaluating a software architecture is a stakeholder-intensive endeavor. I believe that a greater diversity of stakeholders in an evaluation is closer to the context where the proposal is expected to be used (i.e., with many people not being experts in architecture evaluation). Both experiences presented in the article exhibit this characteristic: a very diverse set of stakeholders.

In all cases, all participants were informed that they will be working with a research-oriented approach, and thus some part of the experience will eventually be published. The stakeholders agreed in all cases to this, always asking not to disclose critical or structural information. Previously published experiences were presented to stakeholders. In this way, stakeholders understood that such papers only report the experience and not any related software assets (e.g., architecture descriptions), unless explicitly approved by the involved parties. In addition, the evaluation participants were always instructed to avoid disclosing

sensitive data (e.g., patient information), as concrete instances of such assets are typically of no much interest in the architecture evaluation.

3.2 Survey design and execution

A survey was designed to collect information from a different, independent group of software architecture practitioners. The survey was design using recommendations from [63, 64, 65].

The central idea for the survey in this work is to complement the evidence gathered in the case studies. By using a survey, a broader variety of practitioners can be reached, something not feasible in a case study, especially in a real-world case study in which the participants are dictated by the environment. It is hard to believe that a company will risk the effort used for an architecture evaluation with a mix of participants just to satisfy the constrain that less experienced practitioners could be separated from more experienced practitioners.

Most of the survey participants were contacted via the LinkedIn[1] social network. Others were contacted through direct (i.e., people that the I knew in person and that I knew were or are involved in software architecture at the time) or indirect communication (i.e., people that I contacted because people that I directly know referred me to them). The main target was practitioners related to software architecture. It was observed that many people self-identify as software architects, but following their practitioner experience profile it was found that many of them were far from the definition of a software architect used in this work (i.e., someone with necessary skills and responsible of making software systems architectural decisions).

Participants were told that the survey is anonymous and that even if the results eventually are published, anonymity is preserved. The survey consisted of four sections: the first section asked participants to indicate their experience in software architecture and software architecture evaluation; the second section asked participants to rate in a traditional five-level Likert scale the perceived helpfulness of the proposal in thirteen aspects; the third section asked participants to determine the relative importance of the Alphas for evaluating a software architecture; finally, the fourth section provided participants with an open text field to provide thoughts or comments. More details about the survey can be read in Chapter 7.

[1] `https://www.linkedin.com/`

3.3 Chapter Remarks

This thesis followed a multi paradigm approach. A reflection-in-action approach was used in architecture evaluations to determine the essentials of the architecture evaluation practice. Literature research was used to provide solid ground for the identified essentials and to be more consistent with the SEMAT Kernel, which was used as the modeling framework for the essentials. Case studies were used to gather more real-world evidence from the use of the proposal in real-world architecture evaluations. Finally, a survey was run to complement the results from case studies with a broader variety of practitioners.

Chapter 4

The Essentials of the Software Architecture Evaluation Practice

In this chapter, I propose the five essentials of the software architecture evaluation practice:

- Quality Attributes.
- Architecture Description.
- Business Goals.
- Architecture Decisions.
- Evaluation Adoption.

These five "essentials" are represented as Alphas, using the SEMAT Kernel as a modeling framework.

The essentials for describing endeavors at the practice level need to be general enough to be applicable to all cases of the practice (in this case, software architecture evaluations), while at the same time specific enough to be representative of the practice and not others. There is a trade-off in the decision to articulate some essentials as well as to leave others out.

In this thesis work, the reflection-in-action approach (see Chapter 3) was used to concretely understand and articulate the software architecture evaluation practice in terms of these essentials. Sometimes, I defined elements as essentials only to find out later in other

architecture evaluations that they were too specific and caused some conflict when used in other evaluations. For example, in Section 4.3, I describe the case in which I initially considered scenarios as essential. This essential was discarded because it was too specific for scenario-based architecture evaluation approaches.

In the following, Section 4.1 presents the definition of an Alpha as a modeling construct. Next, Section 4.3 describes each proposed Alphas.

4.1 SEMAT Kernel Definitions

An **Alpha** (acronym: Abstract-Level Progress Health Attribute) is an essential element that is considered relevant to the assessment of the progress and health of a software engineering effort [28]. An **Alpha** represents and holds the discernible **States** that can be observed in the progression of a software engineering effort. In other words, the Alphas express the subjects whose evolution in such efforts the practitioners want to understand, monitor, direct, and control [28].

A **State** in an **Alpha** is a discernible and uniquely identified characterization of the status of an **Alpha** in a specific moment in a software engineering endeavor [28]. An **Alpha** comprises a set of one or more **States** that represent its evolution from its creation to its termination. **States** are characterized in terms of a set of checklist items that must be achieved to indicate that the state has been reached. The graphical notation provided by the SEMAT Kernel represents an **Alpha** and its comprised **States** by means of an "Alpha Card" (see Figure 4.1) which contains the names of the **States,** the name of the **Alpha,** and a brief text that describes the essential that is represented by the **Alpha.** The use of a card to represent **Alphas** is a design decision that allows practitioners and researchers to "touch" the Kernel [27] and therefore use it in a practical, concrete form. The color expresses the Area of Concern: (blue for Endeavor; yellow for Solution; green for Customer).

There are two types of associations between **Alphas** in the SEMAT Kernel. The first one is the **containment** association which expresses that an **Alpha** is subordinated to a superordinate [28]. This association indicates only that the "sub-Alpha" contributes to the progression of the "super-Alpha" and there is no explicit relationship between both set of **States.** The second kind of association between **Alphas** is the general **Alpha Association.** Unlike the first one whose semantics are defined by the Kernel, this second kind of association

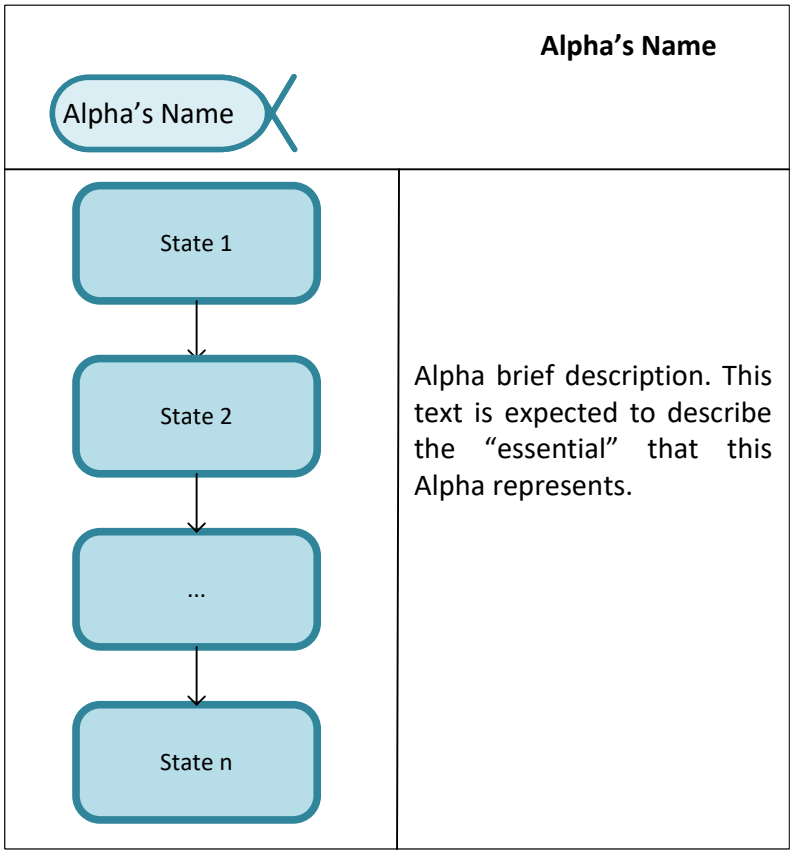


Figure 4.1: The SEMAT Kernel graphical notation for an Alpha and its States.

expresses a relationship between two or more **Alphas** with a semantic that is indicated by a software practice [28]. For example, in this work the Alpha Quality Attributes is considered as subordinate to the Alpha Requirements and at the same time the Alpha Quality Attributes is influenced by (relationship) Alpha Architecture Decisions. The semantics for this second association is defined by software architecture and software architecture evaluation practices.

A **practice** is defined as a repeatable approach to doing something with a specific objective in mind [28]. A **practice** is expressed in terms of **Alphas** and the relationships between the **Alphas** express the mechanisms by which these **Alphas** interact according to the practice. Modeling a **practice** using the SEMAT Kernel means identifying the essential elements of a practice and expressing them as **Alphas** and their relationships. This process

is colloquially known as "essentializing" a practice [29]. The five essentials of the software architecture evaluation practice identified in this work are expressed as the following **Alphas:** Business Goals, Architecture Description, Architecture Decisions, Quality Attributes, and Evaluation Adoption.

The SEMAT Kernel also provides a graphical notation for expressing the criteria that each **State** of an **Alpha** requires to mark the **State** as reached. The criteria are expressed as a checklist on an "Alpha State Card" [27][28]. Figure 4.2 shows a generic Alpha State Card with the checklist composed of checkpoints [28]. Each checkpoint must be marked as achieved to say that the **State** represented in the card has been reached. The same Figure shows that states can be numbered: in this case, **State** i from k **States.**

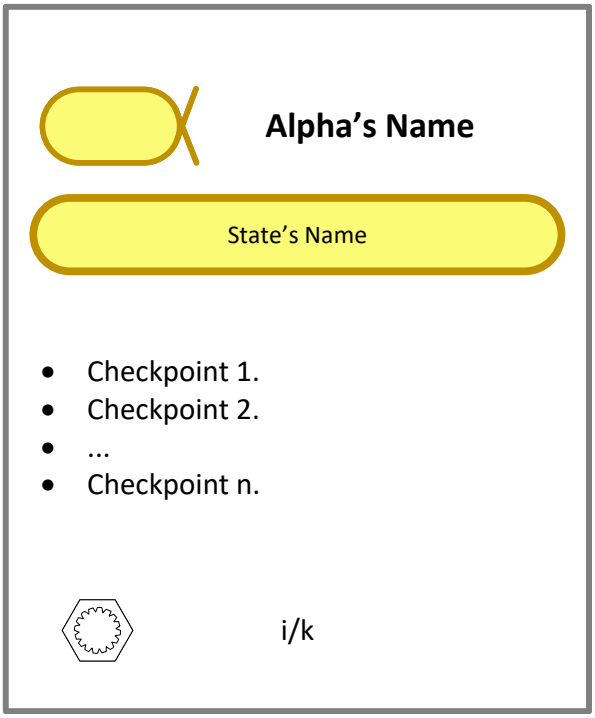


Figure 4.2: A generic Alpha State Card for a specific State.

4.2 An Argument in favor of the SEMAT Kernel

A natural question to answer here is why in this work the SEMAT approach was adopted. The argument in favor of the use of the SEMAT Kernel is supported by the following aspects:

- A software architecture evaluation is an endeavor, that is, a conscious and concerted effort devoted to the assessment of a software architecture fitness for the purpose the system was conceived for. This is the approach that was embraced by the SEMAT Kernel [28][29], that is, the representation of software engineering practices that run as endeavors.
- The SEMAT Kernel defines a common ground for expressing methods and practices in software engineering endeavors [28][29].
- The SEMAT Kernel defines both a graphical and a textual language to describe practices and methods in terms of the essential elements that can be identified after a careful study of a software practice [28][29].

The SEMAT Kernel focuses on software engineering endeavors. And a software architecture evaluation is an endeavor. Regardless of the moment the evaluation is run, it is a temporary effort. Therefore, the SEMAT approach is a natural match for the purpose of representing software architecture evaluations. Frameworks such as the successful Capability Maturity Model Integration (CMMI) [66][67] focus on evaluating and improving the organizational-level (rather than the endeavor-level) software process. Moreover, while the CMMI provides a collection of practices for this goal, it does not provide a *"kernel"* of essential elements from which new practices and methods could be expressed. To avoid confusion, it must be noted that the SEMAT Kernel and the CMMI are not incompatible; indeed, practices from the CMMI can be expressed in the SEMAT standardized language. The standard ISO/IEC/IEEE 42030:2019 "Software, systems and enterprise: Architecture evaluation framework" [68] provides a generic conceptual framework for the evaluation of software, systems, and enterprise architecture. The standard is seen as a complement for the standard ISO/IEC/IEEE 42020:2019 "Software, systems and enterprise: Architecture processes" [69] which define a generic software, systems and enterprise architecture evaluation process (among others) by providing generic recommendations for activities and tasks as well

as related work products. While both standards provide very valuable conceptualizations and recommendations for a process for architecture assessment, they do not provide a foundation of the mechanisms by which the concepts work, and they also do not provide a framework from which expected progression on the activities could be drawn upon.

The second aspect from the SEMAT approach is the availability of a "Kernel" of widely-agreed essential elements that software engineers need to progress in any software engineering endeavor. At the same time, this "Kernel" emerges as a strong common ground for expressing methods and practices [28][29]. As mentioned in Section 2, these essential elements are called Alphas and each of these elements has a progression path that is used to assess where software engineers are at in an architecture evaluation as well as to understand the next steps towards a healthy practice run. The SEMAT Kernel is extensible, which means that a new practice (or set of practices, i.e., a method [29]) can be represented using the Kernel basis as a foundation for the representation. This is also a key characteristic that makes the SEMAT Kernel very appropriate for the purpose of this work. In addition, the SEMAT Kernel became an OMG Standard [28] and as such it provides a standardized way to model software practices.

Finally, the third important aspect is that the SEMAT Kernel offers both a graphical and a textual notation. This allows us to focus on the efforts in the identification of the essential elements for the representation of a software architecture evaluation rather than in designing a new language for these purposes.

4.3 Identifying the Essential Elements of a Software Architecture Evaluation

This work argues that the software architecture evaluation practice characterized as an endeavor deals with the following "essentials" or key elements: Quality Attributes, Architecture Description, Architecture Decisions, Business Goals, and Evaluation Adoption. By adopting the SEMAT Kernel as a modeling framework, this work commits to the following aspects:

- The essentials are represented as Alphas, that is, key elements with which software practitioners work in a software architecture evaluation effort.
- These Alphas comprise a set of predefined States that represent the progression path

that a healthy software architecture evaluation should follow.

- The graphical notation for the Alphas and their States is adopted. Figure 4.3 shows the Alphas proposed in this work using the SEMAT's graphical notation. The black diamond indicates the subordination of the proposed Alphas to the existing Alphas in the SEMAT Kernel.

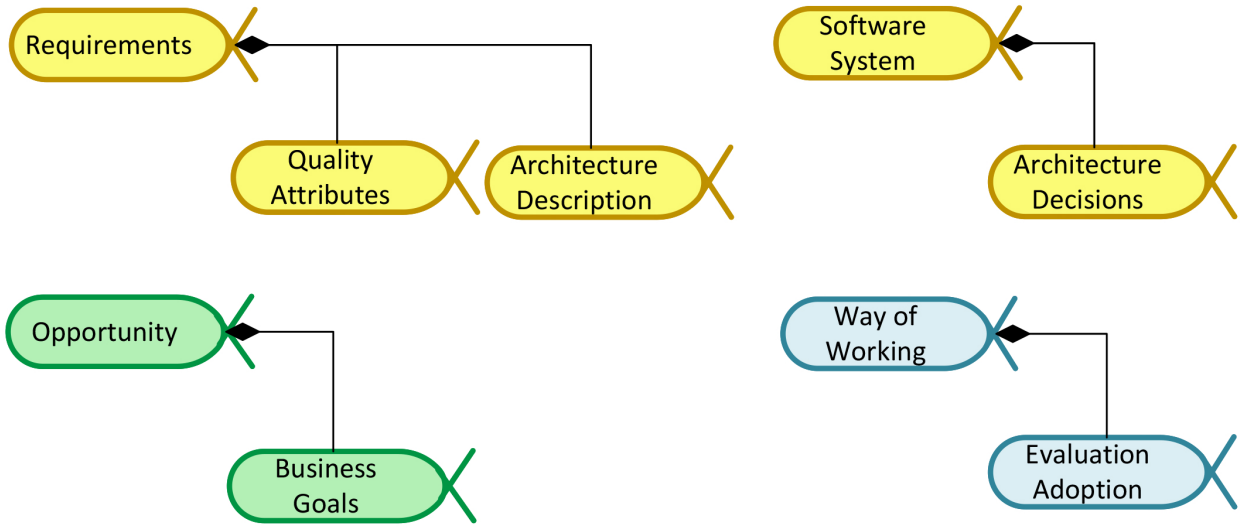


Figure 4.3: The essentials (Alphas) of a software architecture evaluation represented in the SEMAT Kernel graphical notation.

Identifying the Alphas of the software architecture evaluation practice is a key aspect of this research. The Alphas that are identified in this work constitute an extension to the original SEMAT Kernel. Required by the SEMAT Kernel, each Alpha comprises a set of discernible States that represent the progression from starting the work with the Alpha to finishing the work with the Alpha. According to the SEMAT Kernel, to achieve a State, the checklist of that State must be fulfilled [27]. I believe that when time and cost constraints arise, a more relaxed version of this rule could eventually be used: that is, allowing a State achievement by fulfilling a subset of the checklist (i.e., core plus desired items to be fulfilled), although more study is required to define that subset. I remark that this study considers the original SEMAT rule, that is, for a State to be achieved, the checklist must be completely

fulfilled. This is the rule that the I have been enforcing when using this proposal, including the real-world cases presented.

When modeling a practice, there are elements that are inevitably excluded from the set of *essential elements*. For example, scenarios are an important way to articulate quality attributes concerns. However, they are just one of the alternatives. For example, the goal-oriented software requirements engineering [70] provides another approach for dealing with quality attributes concerns. Therefore, Quality Attributes (rather than scenarios) is proposed as one of the essential elements (i.e., Quality Attributes is a new Alpha) that need to progress in evaluating a software architecture.

In the following, the definitions for each one of the new identified Alphas and their States are provided. In addition, Appendix F presents the checklists for each State in each Alpha. These checklists allow for more actionable guiding and progression assessment in an architecture evaluation effort. The checklists are also available as Alpha State Cards in [31].

Figure 4.4 presents in hierarchical form each Alpha with their own States.

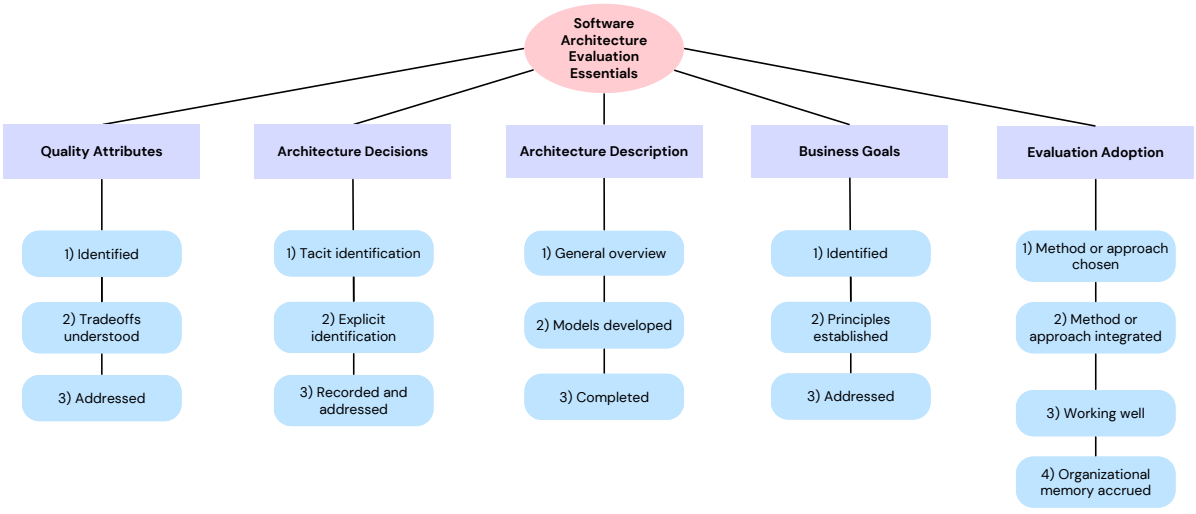


Figure 4.4: The Alphas of a software architecture evaluation with their States.

Hereafter, a "title case" is used when referring to Alphas, and a "sentence case" (or

simply lower case) is used when referring to a concept with the same name as the Alpha. For example, Business Goals refers to Alpha Business Goals and business goals refers to the goals that someone (or a company) has.

4.4 Alpha: Quality Attributes

The Alpha Quality Attributes represents the properties of a software system that are used to define quality in a specific context. Quality attributes are system-wide [71] and therefore system-level design (i.e., the software architecture) has a profound impact on them. In fact, a great part of the architecting of a system is committed to making decisions to satisfy quality attribute requirements [2][72]. Quality models such as the ISO/IEC 25010:2023 [73] comprise a balanced set of quality attributes. Because quality attributes are typically hard to characterize, such quality models often provide more than one level for quality characteristics (e.g., characteristics and sub-characteristics in [73]). In addition, quality models are often recommended to avoid over representing a few quality attributes [5] and balancing balancing stakeholders needs [74].

In an architecture evaluation, the people involved need to understand and characterize quality attributes because a software architecture, i.e., the set of architecture decisions, greatly affects these attributes. Some methods such as the Architecture Tradeoff Analysis Method (ATAM) [6] explicitly mention quality attributes. In my experience [23], quality attributes always come into play when evaluating a software architecture even if the method does not explicitly state working with them (for example, the Decision-Centric Architecture Reviews [21] method does not explicitly progress quality attributes understanding). This approach is also recognized in [75] where the authors argue that a reasoning framework requires working with quality attributes to determine unmet requirements and eventual changes in the architecture.

Characterizing software quality attributes in a way that they can be evaluated is a challenge [72]. Quality attributes tend to be very general in application, and thus a characterization is needed for practical use in architecting a software system as well as evaluating its architecture. Approaches such as utility trees alleviate this problem by making people to reason about quality attribute's concerns and concrete scenarios [17]. However, scenarios are just one more approach for characterizing quality attributes. For example, goal-oriented soft-

ware requirements engineering [70] provides another approach to deal with quality attributes concerns.

The proposed Alpha Quality Attributes comprises three States (see Figure 4.5): Identified, Tradeoffs understood, and Addressed.

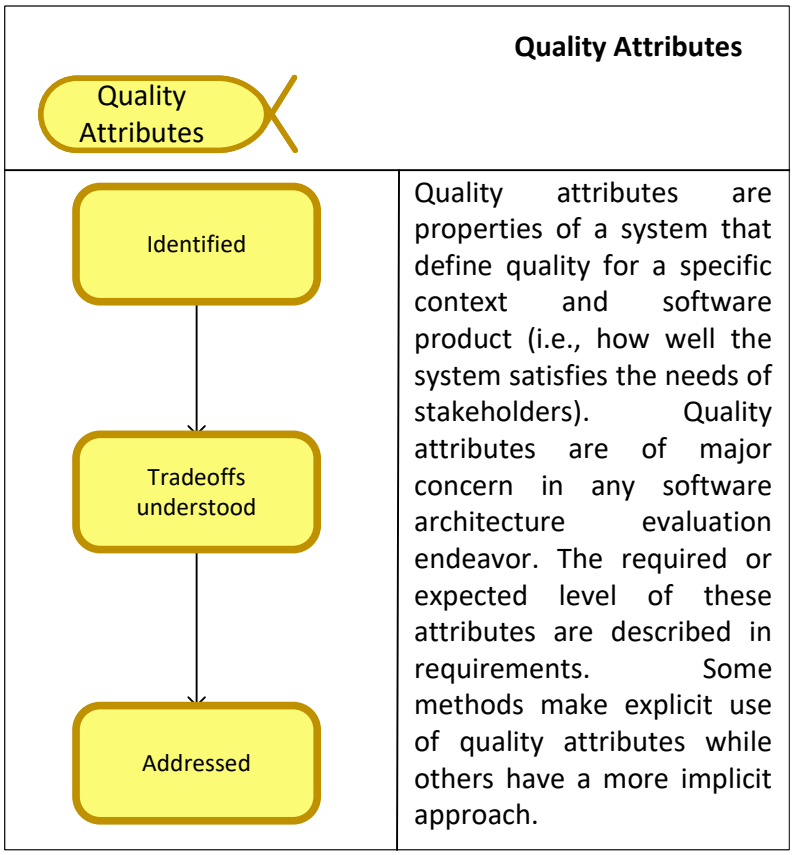


Figure 4.5: The States of Alpha Quality Attributes.

The States are described as follows:

1. **Identified:** Reaching this state means that quality attributes are explicitly identified and recorded. In practice, this might imply that the organization is signing off the acceptance and use of a specific quality model. Quality attributes are characterized as requirements, scenarios, goals, or other approaches.

2. **Tradeoffs understood:** Reaching this state means that quality attributes have been prioritized (i.e., their relative importance in regard to the software system whose architecture is being evaluated). This prioritization has been used to understand tradeoffs between quality attributes.

3. **Addressed:** At this state, prioritized quality attributes have been considered in the architecture evaluation. Tradeoffs have been explained and recorded.

4.5 Alpha: Architecture Decisions

The Alpha Architecture Decisions represents the design determinations that a software architect has made about the structure of a software system to enable software requirements to be met. When treated as first-class entities [76].

Software architecture has been conceptualized using many approaches. For example, Perry and Wolf [77] conceptualize software architecture as a set of architectural elements with a specific form (structure) and a rationale explaining the placement and connection of concrete architectural elements. Perhaps the most practical conceptualization for software architecture is the one that conceives a software architecture as a collection of design decisions [1][78]. This latter focus has motivated a wide variety of architecture activities such as knowledge management and sharing [79]. In this sense, a key aspect of architectural decisions is that they affect quality attributes (positive or negative), although this interaction is sometimes hidden by architectural patterns [80]. As the authors in [80] say, an analogy between question and answer can be made: quality attributes ask for a solution, and the decisions respond.

Architecture knowledge such as design decisions is created in the analysis, design, and even evaluation of a software architecture [81]. Moreover, the evolutive nature of a software architecture accepts that new decisions can be made, as well as the obsolete ones removed [82].

Methods such as the Architecture Tradeoff Analysis Method (ATAM) [6] and Decision-Centric Architecture Reviews [21] explicitly emphasize the evaluation of architecture design decisions, although the first takes a risk identification approach, while the latter takes a decision-challenging approach.

A great deal of architecture knowledge is found tacitly [83][84], including assumptions,

values, and internalized experiences [85]. For practical use in an architecture evaluation, practitioners must consider the progression of architecture decisions from a tacit identification to a concrete decision record. The problem with tacit knowledge is that it is difficult to communicate and analyze [85], and in the worst cases it can lead to a phenomenon called "knowledge vaporization" [84][86] (i.e., knowledge is lost forever). The progression from tacit knowledge to explicit knowledge is known as "knowledge conversion" [87]. Particularly important for the progression of Alpha Architecture Decisions is the "externalization" mode of knowledge conversion [87]. This mode uses codification [84][85][87] to express knowledge in an explicit form. Codification of tacit knowledge means embracing a dialogue or collective reflection [87] in which stakeholders articulate tacit knowledge to express architecture decisions in a structured way [84]. This structured form typically includes a name for the decision, a rationale explaining "the why" of the decision, the scope, the current state (e.g., decided, rejected, or obsolesced), the author, among other attributes [84]. Some authors also call for including the criteria for choosing one alternative [88].

For example, in an architecture evaluation, I observed a tacitly expressed decision about the organization of the modules of the system. After collective reflection in one of the evaluation meetings, this tacit decision progressed to a more articulated representation. This articulation was achieved by collective reflection with several stakeholders. A whiteboard was used to write down the important issues and to characterize the decision in a structured form. This articulated and explicitly defined decision allowed the evaluation to learn that the architecture was already in a desired state: the modules were already organized following a domain-driven approach. Part of this experience is reported in [25].

Once the tacit knowledge that is relevant for the architecture evaluation has been converted to explicit knowledge, the analysis can be approached in a more systematic way. Explicit knowledge is also easier to record for future reference (I have been using Architecture Decision Records (ADRs) for this purpose; see Appendix D for a more detailed description of the ADRs used). Decisions expressed as ADRs can be recorded in version control systems. Services such as GitHub have been used with semantic versioning [1] to store ADRs.

The Alpha Architecture Decisions comprises the following states (see Figure 4.6): Tacit identification, Explicit identification, Recorded and addressed.

[1] See recommendations for semantic versioning at `https://semver.org/`.

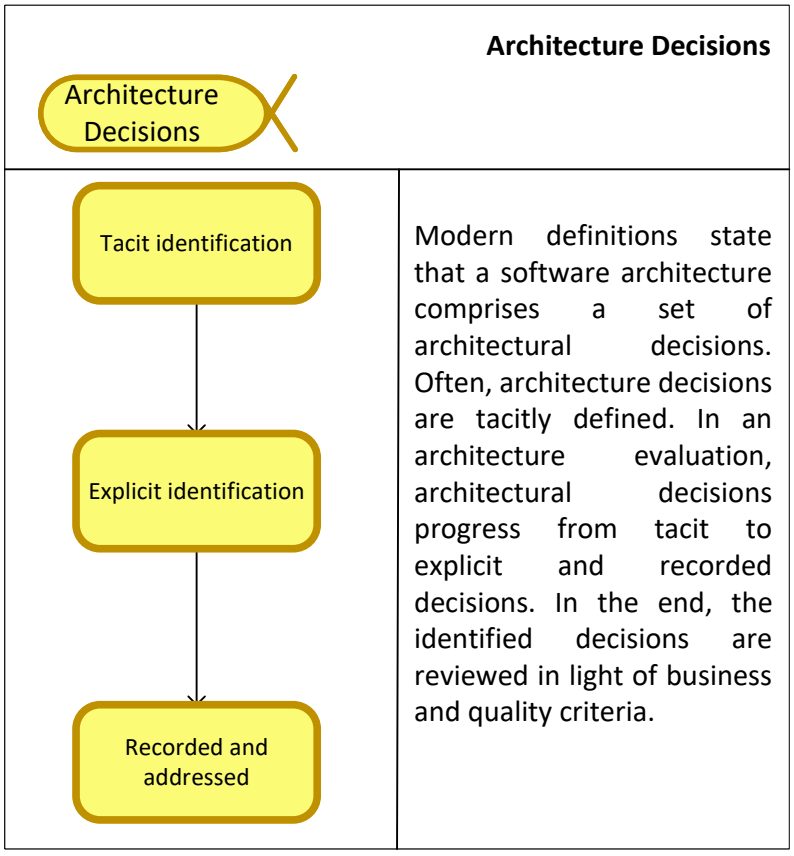


Figure 4.6: The States of Alpha Architecture Decisions.

The States are described as follows:

1. **Tacit identification:** Reaching this state means that architecture decisions are identified at least in a tacit form, that is, no explicit definition or recording is observed. In this state, some knowledge about the rationale behind those decisions is expected (some of them very old as in the case of legacy systems). Some knowledge of who made one or more of these decisions is also expected.

2. **Explicit identification:** At this state, architecture decisions are observed in explicit form. This means that decisions are characterized, although not necessarily recorded, to facilitate clear and concise communication. Some decisions are being (or have been)

evaluated according to the evaluation approach. There are observable efforts for recording decisions, although this is still done in an ad hoc manner. In this state, it is also clear who made the decisions and the rationale behind those decisions. Risks related to design decisions are also being observed as a result of the evaluation and the impact on quality attributes is understood. In this state, relationships between decisions are also expected to be identified.

3. **Recorded and addressed:** In this state, design decisions have been recorded in a specific format for this purpose (for example, the Architecture Decision Record or ADR[2]). Decisions have been reviewed according to the evaluation approach and the reporting of such reviews presents risks identification and the impact on quality attributes. Due to that at this state it is highly expected that some decisions are added, deprecated, or improved, it is also expected that the use of a specific version control system, ideally supported by a specific-purpose tool.

4.6 Alpha: Architecture Description

The Alpha Architecture Description represents the elements that are used for expressing, communicating, and analyzing a software architecture. The ISO/IEC/IEEE 42010:2022 "Software, systems and enterprise: Architecture description" [89] presents an architecture description as a work product that expresses an architecture of an entity of interest (e.g., a software product). As the standard goes generic when talking about architectures, the entity of interest is not explicitly specified [89]. Taylor and van der Hoek [14] agree that the availability of artifacts that allow the analysis of a software architecture is one of the intrinsic challenges of architecture design. Such artifacts are even more critical in globally distributed development [90].

Even if some architecture description exists at the time of the evaluation, this Alpha stresses that an architecture description must reflect actual and current architecture. If an architecture description does not express the current architecture, the evaluation team risks ignoring key architectural issues such as technical debt items [91] (the gap between

[2] "Documenting Architecture Decisions", Michael Nygard, 2011, `https://www.cognitect.com/blog/2011/11/15/documenting-architecture-decisions`.

architecture description and its implementation is a well-known problem in software architecture [92]).

Architecture descriptions are intended to express architecture abstractions, and therefore they are key for reasoning about the architecture [75][93].

For this work, the conceptualization of a software architecture as a set of architecture-level design decisions was adopted.

The Alpha Architecture Description comprises the following states (see Figure 4.7): General overview, Models developed, Completed.

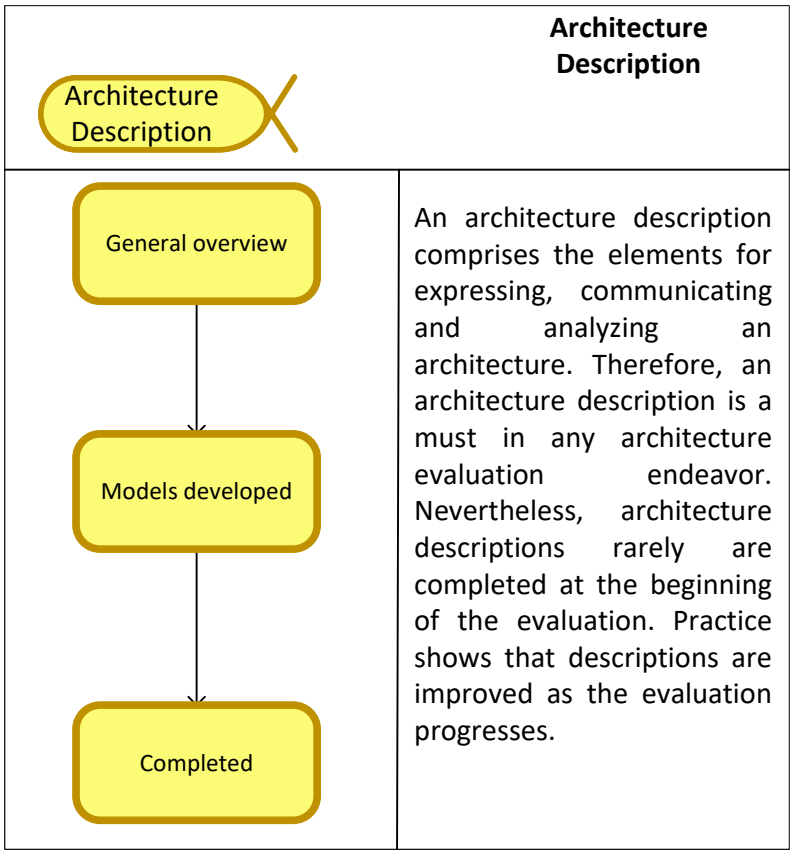


Figure 4.7: The States of Alpha Architecture Description.

The States are described as follows:

1. **General overview:** At this state, a general overview of the architecture exists. This

overview might be composed of one or more diagrams that adapt the abstraction to the stakeholders' concerns. Typically, this overview depicts a high-level view of the architecture components and their deployments. For this state to be reached, a specific configuration of components and connectors is not required.

2. **Models developed:** Reaching this state implies that the architecture description is now encompassing a set of several models presenting more specific system's configurations (components-connectors). The description includes views and viewpoints, and they are justified for evaluation purposes and the concerns of stakeholders. In this state, an architecture description is expected to be expressed informally (e.g., diagrams on a whiteboard complimented with natural language).

3. **Completed:** To reach this state, the architecture description is considered complete for the software system whose architecture is under evaluation. At this state, a description in a specific language such as ArchiMate, UML, SysML, and ADL, among other alternatives, is expected. The use of such languages allows the evaluation team to adopt a modeler tool which in turn enables the adoption of more systematic practices such as configuration management and version control.

4.7 Alpha: Business Goals

The Alpha Business Goals represents the intention that an organization has in relation to a software system. Business goals describe where an organization wants to end up [94] and they can express what the organization tries to achieve with the software system [95].

Not all business goals have an effect on a software system and its architecture [96][97][98]. However, if a business goal has a recognizable effect in the system, it provides the reason for some of the requirements, especially the quality attribute requirements [99] which in turn affect the architecture design [100][97][98].

Understanding business goals at an appropriate level is key for a software architecture evaluation because the software and its architecture should be aligned with these goals. In other words, business goals must be characterized to be suitable for the analysis [96].

The characterization of business goals follows an articulation from more abstract intentions towards more concrete definitions of requirements. I have extensively used two

approaches to articulate business goals. The first approach comes from the "Pedigreed Architecture eLicitation Method" (PALM) [97] where the main idea is to identify quality attributes that, if attended in the architecture, would help achieve the business goal. The second approach is the "NFR Framework" [101] in which the idea is to iteratively decompose the requirements that initially are specified as *softgoals* into more concrete *softgoals* until reaching the level of *operational softgoals*.

In practice, the first approach allows for the identification of a business goal that in some way influences the software system. Then the business goal can be treated as a high-level softgoal which is iteratively decomposed into a quality attribute and then to more concrete architectural alternatives that are said to operationalize the goal [101]. For example, in one architecture evaluation, I observed a business goal expressed as "increase market share" for the signature software product of the company (a payroll system). In the evaluation, this business goal turned out to be related to a quality attribute of performance. The performance attribute then turned into a principle that guides the architecture to support high performance in calculating payroll in companies with many employees and each employee with more than two liquid assets.

Strategic-level business goals definitions, which are organization-wide, tend to appear very abstract in their nature. The articulation of such goals is also important because it allows to understand how these goals shape the intention that the organization has with the software system. For example, I observed once that a cost reduction business goal was translated to a more concrete software-level goal: the migration from an expensive database engine to a free and open-source alternative.

Methods such as the Architecture Tradeoff Analysis Method (ATAM) [6] explicitly stress the identification and characterization of business goals, while other methods such as the Decision-Centric Architecture Review [21] rely on the implicit assumption that business goals have been characterized.

From the above discussion, it becomes clear that some quality attribute requirements are not directly affected by business goals. For example, the adoption of a domain-driven design can be related to the "modularity" quality attribute which is hardly traceable to a business goal. In practicing architecture evaluation in many systems, I have observed this issue several times, motivating the distinction between Alphas Quality Attributes and Business Goals in this proposal.

The Alpha Business Goals comprises the following states (see Figure 4.8): Identified, Principles established, Addressed.

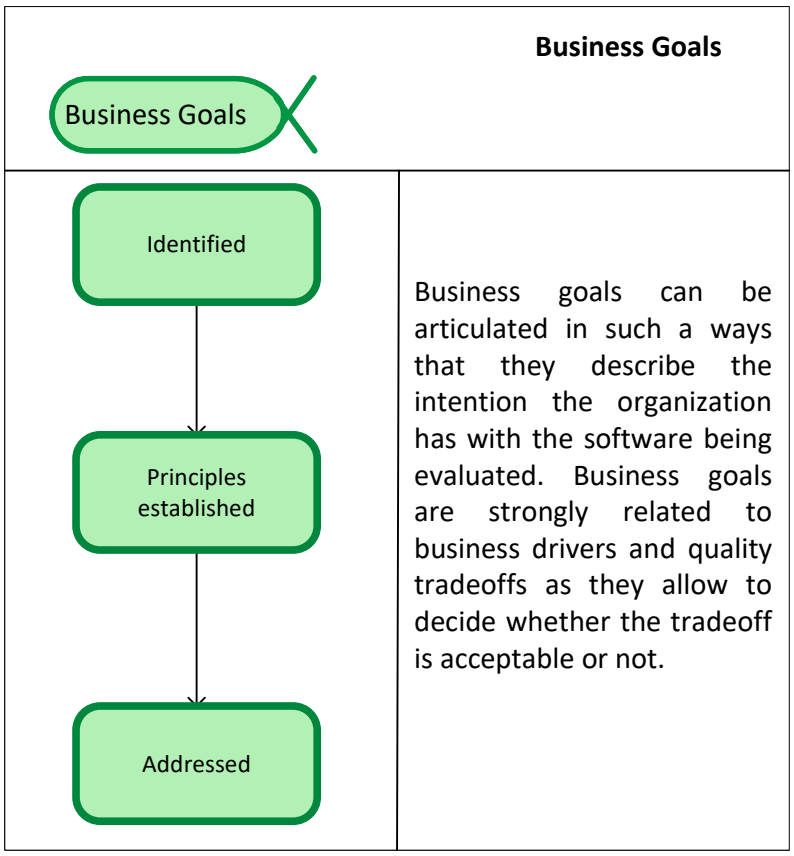


Figure 4.8: The States of Alpha Business Goals.

The states are described as follows:

1. **Identified:** At this state the organization's business goals (or mission goals) are explicitly identified. This means that the business goals are characterized in an appropriate form so that they can be used for the architecture analysis. The business drivers for the architecture are also identified.

2. **Principles established:** At this point, statements about the definition of the software architecture are clear and explicitly identified. Again, this means that these definitions

are expressed and characterized appropriately for architecture analysis purposes. The principles that guide the architecture being evaluated are also identified, and they are being used to evaluate the adequacy for the business (or mission) goals.

3. **Addressed:** Reaching this state means that the architecture evaluation has answered how the evaluated software architecture aids in, and/or risks meeting the business (or mission) goals that justify the software existence. In addition, a discussion of the tradeoffs between the business (or mission) goals in relation to the reviewed architecture is expected.

4.8 Alpha: Evaluation Adoption

The Alpha Evaluation Adoption represents the expected path for a healthy practice adoption. It consists of the progression path for the use of the software architecture evaluation practice. The claimed benefits of a software practice, such as architecture evaluation, will be observed if practitioners can integrate the practice into the development effort [102].

This Alpha progresses from the selection of an appropriate method or approach for the architecture evaluation, to the organizational memory accrued. I have used reflective experience in architecture evaluation to propose this path for a more natural adoption of architecture evaluation.

The Alpha Architecture Evaluation comprises the following states (see Figure 4.9): Method or approach chosen, Method or approach integrated, Working well, Organizational memory accrued.

The States are described as follows:

1. **Method or approach chosen:** Reaching this state means that the evaluation methods or approaches have been reviewed, the criteria for the selection of one method or approach have been agreed, and as a consequence the method or approach has been chosen. I have observed that not all stakeholders understand the benefits of running an architecture review, and thus in this state it is also expected that stakeholders have agreed that the architecture evaluation is justified.

2. **Method or approach integrated:** At this state, it is expected that the method or

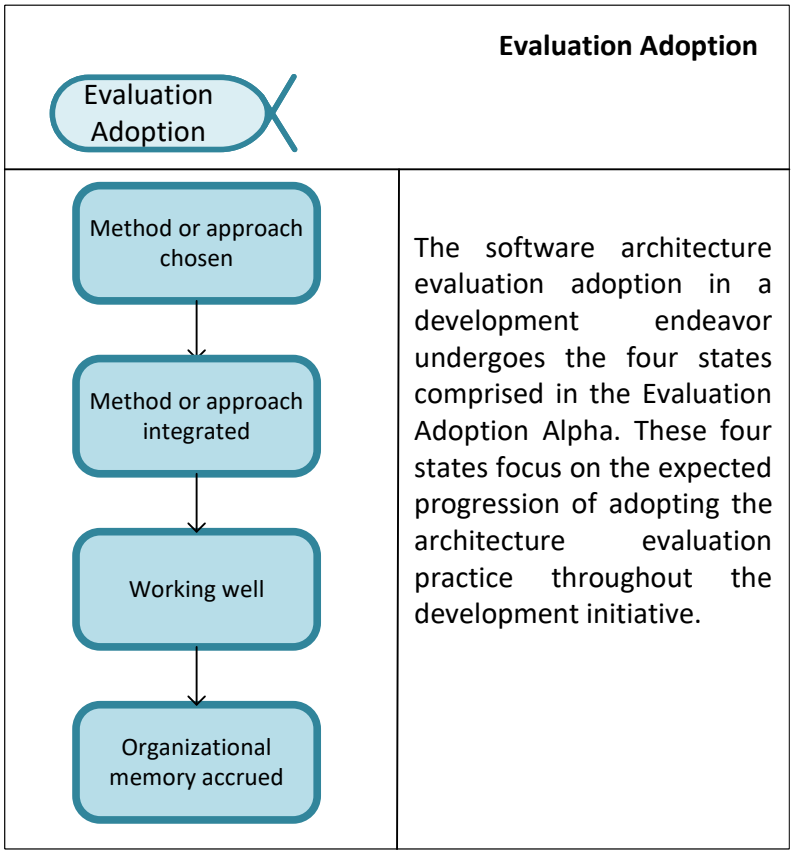


Figure 4.9: The States of Alpha Evaluation Adoption.

approach has been explained to the evaluation stakeholders. Steps, activities, roles, work products, and expected responsibilities and effort are understood and agreed. If for any reason one or more stakeholders find that the method or approach should be re-assessed in terms of its suitability for the case, then it is expected at this point that a discussion about this has already been done.

3. **Working well:** Reaching this state means that deviations between planned an actual activities have been corrected and the evaluation team and stakeholders are working in a seamless way. The results being delivered (partial or final) are expected to be consistent. The use of sound tools (e.g., software modelers, configuration management software) is expected, and stakeholders agree that the evaluation goals are being met.

4. **Organizational memory accrued:** At this point, the architecture evaluation inherent tasks are finished. Reaching this state means that the organization has accrued in some form the significant knowledge learned from the evaluation experience. The organization might have started accruing evaluation experience knowledge long before reaching this state (for example, in Scrum it is reasonably that this "knowledge accruement" started as soon as the team starts reflecting about the process in the retrospectives). For this state to be reached, the focus is on architecture evaluation process-related knowledge, rather than architecture-related knowledge (e.g., architecture decisions, a matter of the Alpha Architecture Decisions). Knowledge might be non-articulated (i.e., tacit knowledge) or articulated and recorded. The Alpha in this state does not prescribe any concrete mechanism for knowledge management as this is organization-dependent. What is important in this state is to understand that process-related memory is an important element to improve organization performance [103].

4.9 Alphas and Relationships

In accordance with the previous definitions for each Alpha, the following relationships are recognized:

- Business Goals drive Quality Attributes.
- Architecture Description express Architecture Decisions.
- Architecture Decisions influence Quality Attributes.
- Evaluation Adoption reviews Architecture Decisions.
- Evaluation Adoption improves Architecture Description.
- Evaluation Adoption examines Business Goals, and Quality Attributes.

See Figure 4.10 for Alphas and their relationships using the SEMAT Kernel standard graphical notation grouped by area of concern.

These relationships express the connection between these "essential" elements that practitioners work with in an architecture evaluation, and therefore they do not imply any ordering in the use of the Alphas of this proposal.

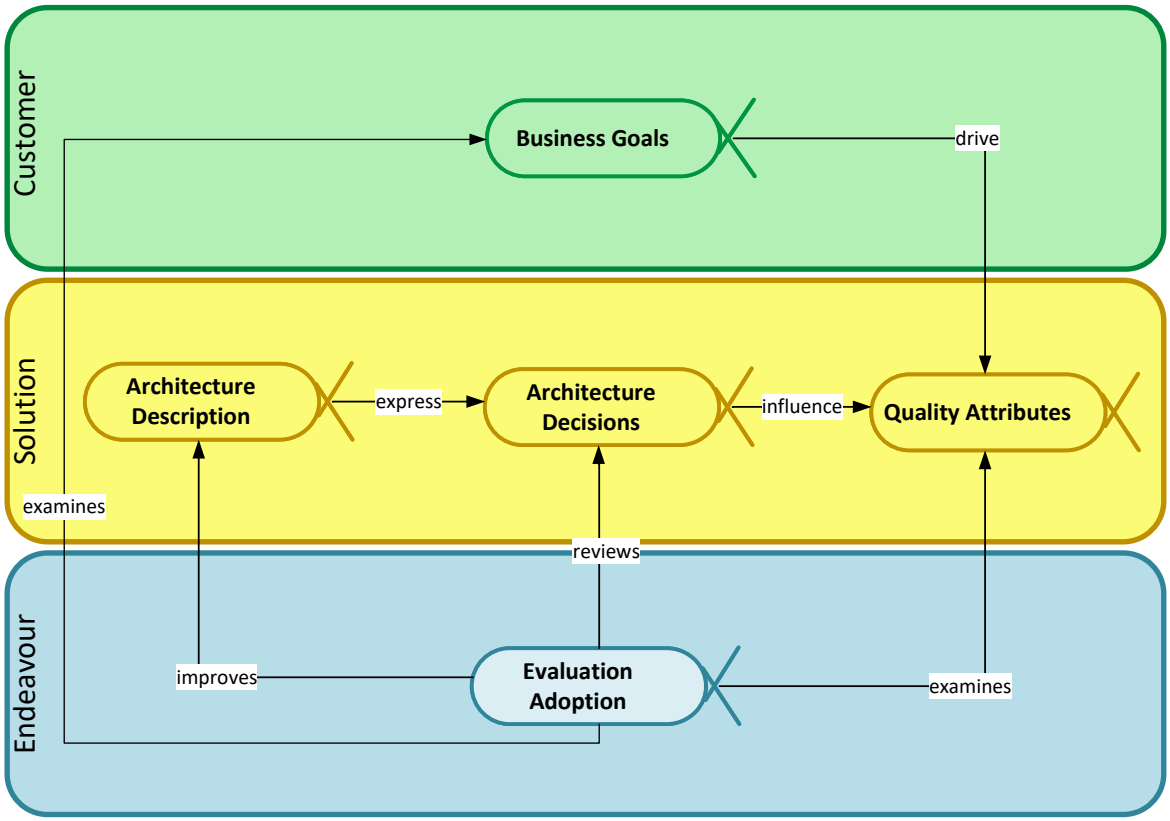


Figure 4.10: Relationships between the Software Architecture Evaluation Alphas represented using SEMAT's graphical notation.

4.10 A How-to Discussion

In practical use of this proposal, I have observed two difficulties that practitioners face when making first-time use of the proposal:

- How Alpha State Cards are used.
- What about Stakeholders (and other "essentials" in software engineering).

These difficulties are discussed in this section to help mitigate a potential "cold start" in the use of the Alphas proposed in this work.

4.10.1 How to use the Alpha State Cards to Guide and Assess a Software Architecture Evaluation

The reader should recall that each Alpha progresses through a predefined number of States (see 4.1). The progression is a consequence of meeting the criteria for each State that is expressed in an Alpha State Card (see Section 4.1).

For example, Figure 4.11 shows two Alpha State Cards from the proposal. These two Alpha State Cards present the States 1 (Tacit identification) and 2 (Explicit identification) for Alpha Architecture Decisions. In the example, for an engineering team to say that State 1 is achieved, they must find evidence that supports that the checklist items are met. In addition, an engineering team will seek to meet the criteria of State 2 to progress in this Alpha.

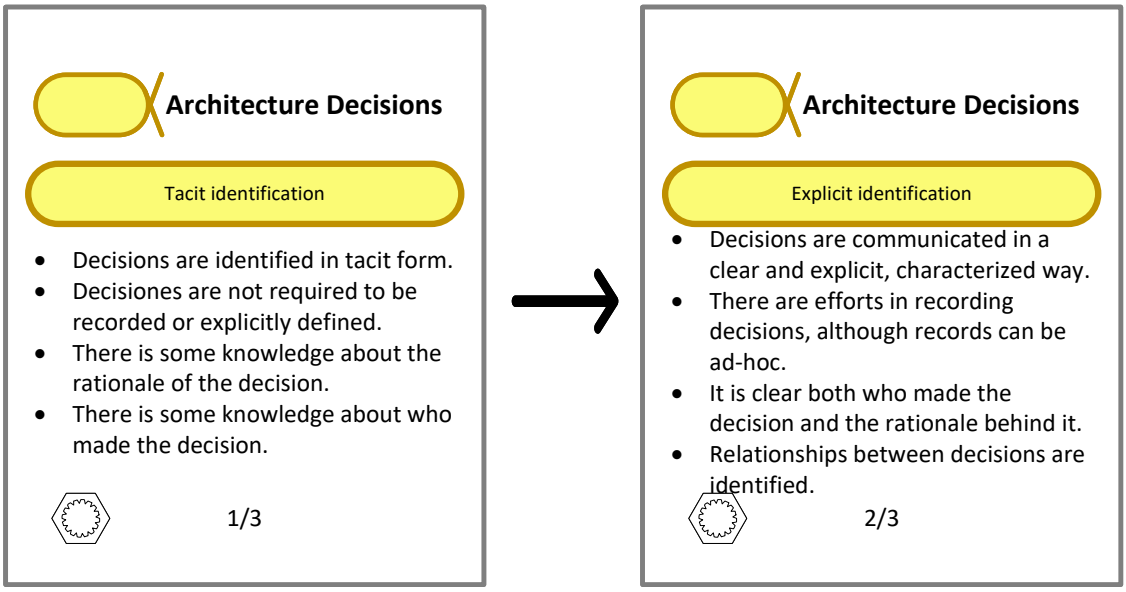


Figure 4.11: States 1 and 2 from Alpha Architecture Decisions represented using SEMAT's graphical notation for state cards.

In practice, an architecture evaluation is guided and its progression assessed by doing the necessary work to meet the criteria of the current and following States in an incremental way. If after making an assessment with the Alphas the team finds that no State is still reached,

then the evaluation team is required to perform the necessary work to satisfy the checklist items for the first State.

I do not prescribe a particular order for the use of Alphas. Indeed, I expect these Alphas to be worked with different efforts, that is, following an iterative and incremental manner. The ordering in which the Alphas will be attended depends on the particular product, project, or even organizational context. For example, if an organization has already been working with architecture evaluations and has decided to prescribe a particular method, then the Alpha Evaluation Adoption will not be a first priority (especially for the first State).

4.10.2 What about stakeholders?

Some practitioners have asked "what about stakeholders?" in the proposed Alphas. In fact, the proposal does not include an Alpha for stakeholders and they are critical to a software architecture [4] and its evaluation.

To answer this question, recall that this software architecture evaluation practice representation is also an extension of the SEMAT Kernel [28]. As stated in Section 2, the SEMAT Kernel already includes the Alpha Stakeholders [28]. Thus, when using this representation to guide and assess the progression of an architecture evaluation, the original SEMAT Kernel Alphas are expected to be used as well. The extent of use of the original Alphas will depend on the context. For example, in the second case presented in this paper, a very well-developed identification and characterization of stakeholders was observed. Thus, when the evaluation started, the Alpha Stakeholders was already in an appropriate state for architecture evaluation purposes.

The answer to this question has been exemplified with the Alpha Stakeholders because this has been explicitly noticed by some practitioners when using the proposed Alphas. Nevertheless, this answer applies to any other existing Alpha in the SEMAT Kernel (i.e., Opportunity, Software System, Team, Way of Working, and Requirements) [28].

4.11 Chapter Remarks

In this chapter, I propose five essentials to articulate the software architecture evaluation practice: Quality Attributes, Architecture Description, Business Goals, Architecture Deci-

sions, Evaluation Adoption. The SEMAT Kernel was adopted as the modeling framework. Thus, the five essentials are represented as Alphas, each with their own set of States that represent the progression from initial stages to stages in which the Alpha is considered fully addressed in an architecture evaluation.

In this chapter, the relationship between the Alphas is also described. The SEMAT Kernel graphical notation is also used to visually represent the Alphas with Alpha and Alpha State Cards.

Chapter 5

Software Architecture Evaluation Activity Spaces

5.1 Introduction to Activity Spaces and Activities

The SEMAT Kernel defines an Activity Space [28] as a placeholder for something to be done in a software engineering endeavor. An Activity Space acts as a group to classify zero or more Activities [28]. Each Activity is an element that defines one or more Work Items [28]. In turn, a Work Item is defined as a concrete and discrete piece of work (i.e., a mental and physical effort made by the team to produce a software system) that may lead to a State change or to the confirmation of a State [28].

A key design decision in the SEMAT Kernel is to avoid an explicit definition of Activities [29]. Instead, the Kernel provides a set of Activity Spaces that are general enough to accommodate a general software engineering effort, while at the same time avoiding restricting the applicability of the SEMAT Kernel to concrete engineering endeavors where these Activities would be appropriate. More specific practices such as evaluating a software architecture require more specific Activity Spaces which can be added due to the extensibility of the SEMAT Kernel [28][29].

As a consequence of this design decision, the SEMAT Kernel is not very specific when describing an Activity. The Kernel put emphasis on "Actions", that is, operations that can be performed and that are related to one or more Alphas [28], rather than in the characterization

of the "Action".

In this chapter, I propose nine new software architecture evaluation Activity Spaces. These Activity Spaces complement the software architecture evaluation Alphas and have been defined after a reflection-in-action analysis of the architecture evaluation experiences, including the two ones from Chapter 6. I propose these new Activity Spaces in the three SEMAT Kernel Areas of Concern: Customer, Solution, and Endeavor.

I also present fourteen concrete Activities, described with an emphasis on what has to be done to achieve the Activity goal. As mentioned in the SEMAT Kernel, the achievement of the goal of an Activity can contribute to the progression of some Alphas, although this is not guaranteed because the concrete instance of the practice (including the development context) will also have an effect on the Alphas progression [28].

In the definition of the Activity Spaces and Activities, I must stress the following:

- In proposing the Activities, I tried to avoid the definition of Activities for actions that are explicitly stated by architecture methods. In my opinion, restating actions in terms of Activities is not as useful as having Activities for actions that are not explicitly stated nor obvious when evaluating software architectures.

- Activities should be understood as modular units, much like Alphas as defined in the SEMAT Kernel [27]. This means that the proposed Activities are included at the discretion of the practitioners/researchers in a particular architecture evaluation. Practitioners/researchers are not required to include all Activities presented here. In addition, practitioners/researchers in their own development contexts can feel that new, more context-specific Activities can be added.

5.2 Activity Spaces in Area of Concern "Customer"

5.2.1 Understand the Goal

A software architecture evaluation is a serious endeavor that should be planed, and this includes defining what the goal (or goals) is for the endeavor. Is it an evaluation for a software system subject to acquisition? Is it an evaluation for a software migration? Is it a software evaluation for a software system that is currently being developed? The goal might

even be more technical, such as evaluating an architecture aiming at finding decisions that are prone to increase architecture technical debt.

There will be a myriad of possible goals. The important thing to do is to carry one or more activities to explicitly discover the goal.

An Activiy example is to Elicit What the Company is Trying to Achieve with running an architecture evaluation.

5.2.2 Understand the Organization

The evaluation of a software architecture is done in the context of an organization that is, most of the time, trying to bring concrete evidence for supporting (or rejecting) architecture decisions. Understanding the static and dynamics aspects of an organization plays a vital role in running an architecture evaluation. For example, when I have evaluated software architectures in highly structured and hierarchical organizations such as in the military, I have found critical to understand the communication approaches to avoid miscommunication issues that might eventually erode people interaction.

Identify Organization's Communication Approaches and Identify Support and Sponsorship in the Organization are two examples of Activities in this Activity Space.

5.2.3 Manage Expectations

Evaluating a software architecture can be a very inspiring activity, even more if it is the first time the evaluation is done. I believe that expectations — that is, what stakeholders should expect (and when) — should be managed in some ways. The specific ways are a matter of specific Activities. For example, in two evaluations, I have been asked about recommendations to abandon a database provider. In these evaluations, I have noticed that some stakeholders tried to use the evaluation to gather evidence to validate a previously existing thought on abandoning the database provider.

How expectations are specifically managed is a matter of specific Activities (see section 5.6). I have found that showing generic and depersonalized results adapted from previous architecture reviews (provided that there is permission to do that and there is no violation of a non-disclosure agreement) is a good vehicle for stakeholders to understand what the evaluation will bring to the discussion.

State the Architecture Evaluation Goal is one example of Activity in this Activity Space.

5.3 Activity Spaces in Area of Concern "Solution"

5.3.1 Explore Architecture Alternatives

One of the critical things to do in a software architecture evaluation is to explore alternatives to the architecture (or part of the architecture) being reviewed.

There are various ways architecture alternatives can be explored. For example, one can take a pattern-based approach in which using a set of quality attributes important for the system are used to guide a pattern selection.

Challenge Decisions to see if the decision appropriately supports some scenarios and to discover alternative architecture decisions is one Activity example for this Activity Space.

5.3.2 Capture the Architecture

Capturing the architecture is expressing the software architecture subject to evaluation by some means. As with other Activity Spaces, the concrete ways an architecture is captured is a matter of specific Activities. For example, if taking the "architecture as a set of design decisions" approach, the architecture would end up as a set of architecture decision records (perhaps complemented with other artifacts).

Identify System's Roles/Actors, Globally Overview the Current System and Identify Architecture Patterns are examples of Activities that belong to this Activity Space.

5.4 Activity Spaces in Area of Concern "Endeavor"

5.4.1 Do the Offline Work

The evaluation of a software architecture requires the preparation of work products that we expect to use in the review. This is especially important for a first-time evaluation in which one can reasonably expect that the software architecture exists in a tacit form.

Therefore, by "Offline Work" I mean all the things that we need to do in the context of an architecture evaluation but are not explicitly part of the evaluation method itself. For

example, printing the architecture evaluations work products and organizing them in a folder for each participant. If required, Offline Work can also include organizing some amenities such as a coffee-break.

An example of Activity in this Activity Space is to Record Organizational Learning (typically done in a retrospective after running the evaluation method) and to Print Architecture Documentation (typically done before starting the evaluation).

5.4.2 Meet Entrance Requirements

Evaluating the software architecture of a software system needs gaining access to key information from the system and the company or organization. This is especially true if the review is conducted by a review team that is external to the company. In my experience, I have found that the military has the most stringent entrance requirements. Nevertheless, most companies will have more or less specific entrance requirements, such as signing a non-disclosure agreement.

Entrance requirements can be very diverse, ranging from just organizing and presenting software engineering certificates to more stringent requirements, such as the request for a criminal record certificate from all review team members.

An example of Activity in this Activity Space is to Sign a Non-Disclosure Agreement.

5.4.3 Understand the Practice

Stakeholders (including the review team) should be aware of the practical aspects of running an architecture evaluation. Stakeholders not only need to comprehend a particular evaluation method to be used, but they also need to increase their awareness of the required work products, the roles that will be required, and the expected effort in all the planned evaluation tasks.

This Activity Space comprises all the Activities that aim at making stakeholders aware of the practice of evaluating a software architecture as well as understanding how this will be done. An Activity example in this Activity Space is to Adapt Scenarios Examples.

5.4.4 Present Results

An architecture evaluation must show results from the work. Whether more formally, such as in the ATAM [6], or more informally, such as in PBAR [38], an architecture evaluation always has to present some kind of results.

This Activity Space encompasses all the things to do when presenting the results in the context of an evaluation effort.

The Present Partial Results Activity is an example of something to do that belongs to this Activity Space.

5.5 Activity Spaces in SEMAT Graphical Notation

Figure 5.1 shows the proposed Activity Spaces in the graphical notation provided by the Essence Standard. The diagram also shows the type of Area of Concern that each Activity Space belongs to.

5.6 Activities in Activity Spaces

In this section, I propose and present fourteen software architecture evaluation Activities, each of which is classified in one of the proposed Activity Spaces. I recall that the SEMAT Kernel avoids the definition of explicit Activities to ensure that the Kernel is applicable to a wide spectrum of software engineering endeavors. In this thesis, I decided to propose a set of Activities that are general enough to be applicable in any software architecture evaluation endeavor.

I also recall the following:

- Activities are also seen as modular units in the SEMAT Kernel. Therefore, the Activities that I propose should be seen with the same approach, meaning that practitioners and researchers can decide to include one or more according to the circumstances.
- The Activity Spaces allow for more Activities. In other words, the Activities that I propose are by no means an exhaustive list of actions to be done in evaluating an architecture. If other Activities are necessary, the new ones can be added to the Activity

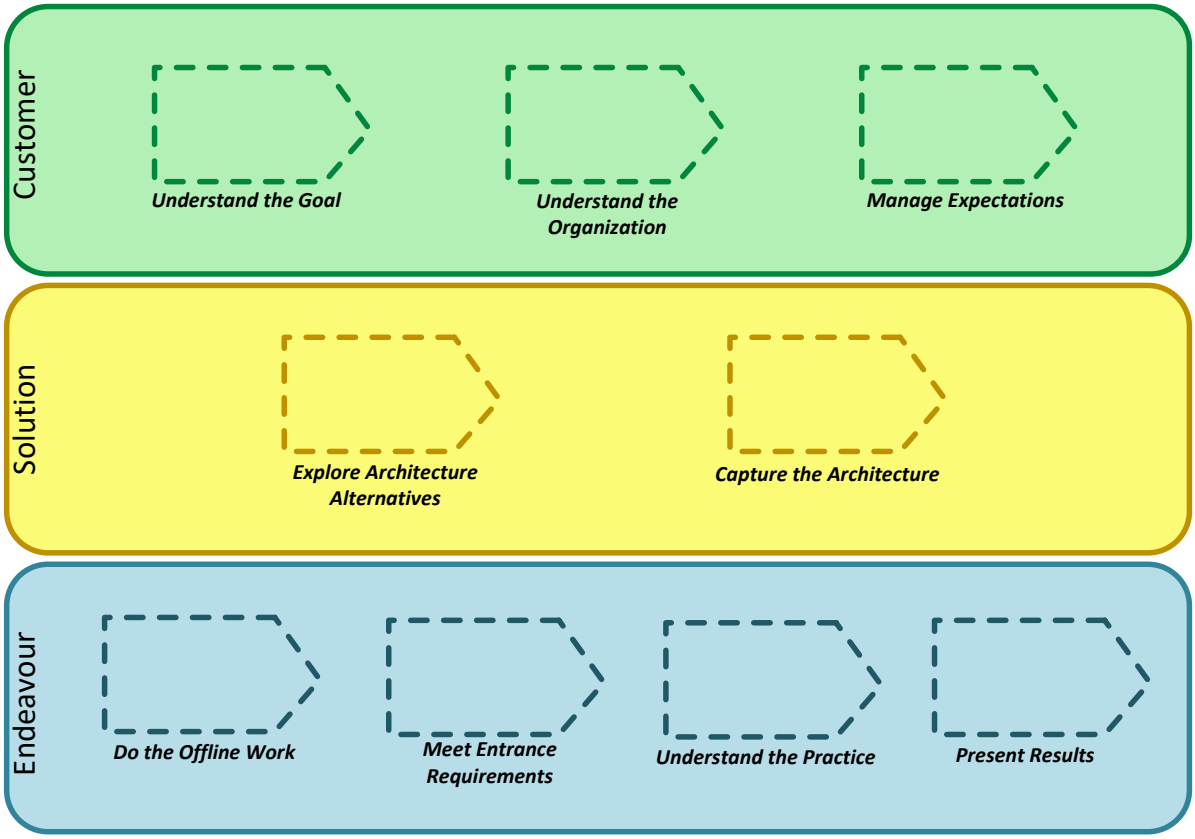


Figure 5.1: The new proposed Activity Spaces, grouped in three Areas of Concern.

Spaces (in other words, this proposal also inherits the extensibility property of the SEMAT Kernel).

5.7 Activities in Activity Space "Capture the Architecture"

5.7.1 Activity: Identify System's Roles/Actors

Context

The vague definition of roles is a recurring issue in many organizations. This is not to say that the skills and responsibilities of a role are not displayed on the job. Rather, what happens is that the characteristics of the roles are assumed by different people. For software architecture evaluations, the participation of people who take specific roles is important. For example, financial users, administrators, data analysis users, among others, especially when eliciting quality and functional requirements that the evaluation team have to face with current architecture (see Alphas Quality Attributes and Architecture Decisions). Proper identification of roles will aid in correct planning of the work sessions due to the participation of the right people in terms of the responsibility they have with respect to the elements evaluated.

Goal

Be more efficient in eliciting quality and functional requirements for the architecture evaluation by focusing on the concrete people that play a role in the system.

Action to do

The assessment leader must be careful to distinguish between role and title. Many times, job positions do not directly correlate with the roles. For example, an engineer hired as an analyst may also be the one who has the most participation in the decisions of the architecture, making this analyst someone who also takes the role of architect (and, therefore,

important for the evaluation). The evaluation leader must also pay attention to the specific roles indicated by the architecture evaluation method that will be used.

5.7.2 Activity: Globally Overview the Current System

Context The evaluation methods of software architecture are dependent on the participation of stakeholders in the broad sense of the concept. For example, a lawyer in the organization is a potential stakeholder who may not be familiar with the architecture of the software system to be evaluated (see Alpha Architecture Description). In practice, I have observed that many stakeholders are taken by surprise when told of the software system to be evaluated. By the time the various stakeholders begin to participate in the evaluation meetings, they will be clear about the system, its main components and relationships, and its context.

Goal Familiarize (increase awareness) the evaluation stakeholders with the software system whose architecture will be evaluated.

Action to do Before starting the architecture evaluation, the evaluation team must worry about generating precise and easy-to-understand documentation to present the general view of the system. The idea is that when stakeholders begin to work in the evaluation, they understand what functionality they commonly use is provided by the software system (or part of it). In my experience, the lack of a proper name for the system is one of the most recurring problems when dealing with the understanding of what functionality is provided by the system (often stakeholders just use the system, and they do not explicitly know what the system is). In summary, the team must prepare general overviews of the systems, ideally identifying roles and their relationships to the system.

5.7.3 Activity: Identify Architecture Patterns

Context A software architecture may exhibit architectural patterns, especially if the initial efforts in designing the architecture has considered well-known approaches. The word pattern in this Activity is used in the broad sense (i.e., style, pattern, or tactic), much like

the ATAM [6] when it states to identify architectural approaches (see Alphas Architecture Description and Architecture Decisions).

Not all architectures are expected to exhibit patterns. Theoretically, a software architecture can be the result of years of inorganic growth. In those theoretical cases, architecture patterns will be scarce or simply absent. This Activity does not call for identifying patterns when no patterns are expected. Rather, this Activity reminds practitioners to search for patterns.

Goal Facilitate architecture risk analysis by providing concrete analyzable architecture's design artifacts.

Action to do Work closely with Alphas Architecture Description and Architecture Decisions to clearly identify and record the current architecture's design patterns. Record the design patterns according to these Alphas in order to facilitate the analysis of risks posed by the current architecture in light of stakeholders' concerns.

5.8 Activities in Activity Space "Explore Architecture Alternatives"

In this Activity Space, I propose one Activity: Challenge Decisions.

5.8.1 Activity: Challenge Decisions

Context

As discussed in previous chapters, a widely agreed definition of software architecture states that it is a collection of architecture design decisions [78, 82]. Understanding the reasoning behind a design decision is an important step in any architecture evaluation because this allows evaluators to understand the "why" of the decisions. The DCAR method [21] explicitly encourages challenging decisions. The idea of this Activity is to increase awareness of the reasoning behind the decisions, something I have observed is well valued by evaluation

participants because often the people who made the decision are no longer present to explain it.

Goal

Increase understanding of the arguments behind architectural decisions and determine whether there is solid ground supporting them.

Action to do

After articulating and explicitly recording architecture decisions (see Alpha Architecture Decisions), present the decision to evaluators and ask them to present their concerns regarding the decision and how it supports (or hinders) the stated requirements (this might imply working with Alpha Quality Attributes to elicit and define concrete requirements).

Use the characterizations of the design decisions (Alpha Architecture Decisions) to challenge each decision and, if possible, allow the architects involved in each decision making to present arguments supporting each decision. If the arguments do not convince the evaluators, mark these decisions as risky or with another specific agreed tag. It is important to remember that this Activity encourages challenging the decision as an entity, avoiding blaming people involved in the decision making (if present at the time of the evaluation).

5.9 Activities in Activity Space "Understand the Organization"

The Activities that I propose for this Activity Space are two: Identify the Organization Communication Approaches and Identify Support and Sponsorship in the Organization.

5.9.1 Activity: Identify the Organization Communication Approaches

Context Knowing and respecting the formally or tacitly defined communication approaches, the channels and their manners, is key to avoid undermining interpersonal relationships. I have observed that well-managed interpersonal relationships are an important factor in the eventual success of an architecture evaluation (see Alpha Evaluation Adoption). Architecture

evaluations are invasive in terms of communication with and among people in the organization. For this reason, it is important to respect their approaches to ensure communication efficiency. In my experience, in contexts such as military organizations, respecting these communication approaches is even more critical where it is important for the evaluation team to know how to reach certain stakeholders without affecting these channels.

Goal Ensure that the architecture evaluation stakeholders are aware of the communication approaches so that they do not hinder the fluency of the process.

Action to do Especially applicable to environments in which the channels and forms of communication are formally defined and used, this Activity motivates the evaluation team to discover these channels and the form of communication in order to respect the culture of communication in the organization. I have used ad-hoc "communication trees" to elicit the typical communication channels and approaches. Although it is hard to characterize, I have also observed that watching and following attention to the day-to-day communication, that is, the regular exchange of information, thoughts, and even emotions is a good strategy to identify the communication approaches.

5.9.2 Activity: Identify Support and Sponsorship in the Organization

Context Every organization has some organizational structure. Whether it is formally defined or tacitly defined, software architecture evaluations are especially sensitive to the support and sponsorship of heads, managers, and equivalent positions in other types of institutions (e.g., educational, armed forces). A good sponsorship increases the awareness and adoption of the architecture evaluation (see Alpha Evaluation Adoption).

Goal Increase support in the architecture evaluation running by identifying and involving key stakeholders in the organization that can positively influence the enactment and adoption of the evaluation.

Action to do The evaluation team must discover and characterize the organizational structure in order to identify positions that are especially important for the support and sponsorship of the evaluation. In my experience, most of the time, the difference between the participation and the absence of some stakeholders is due to a lack of support and sponsorship.

5.10 Activities in Activity Space "Manage Expectations"

The Activity that I propose in this Activity Space is State the Architecture Evaluation Goal.

5.10.1 Activity: State the Architecture Evaluation Goal

Context

In general, software architecture evaluations are performed with some goal in mind. While an architecture evaluation can be part of iterative processes, an architecture evaluation is rarely seen without a goal. Determining this goal is important to guide the evaluation efforts towards where the organization wants to be after finishing the evaluation (see Alpha Evaluation Adoption). For example, I have seen organizations that want to enact an architecture evaluation to support strategic decisions such as migrating a platform or buying a new software system, to bring evidence to stricter clients, as well as the typical quality assessment motives. In practice, this Activity is not about discovering the goal; rather, most of the time, this Activity is about establishing the goal, so involved people get their expectations well grounded.

Goal Increase architecture evaluation awareness and support by establishing where the organization wants to be after finishing the evaluation, while at the same time motivating better direction for efforts.

Action to do After using some goal-searching techniques to discover the goal of evaluating the architecture of a software system, explicitly define and write down the evaluation effort

goal.

The goal of the evaluation effort should be easily accessible to all people involved in the review, not just the reviewer, especially the stakeholders who have occasional participation. This is to avoid them struggling to understand what the organization and the review team are doing.

5.11 Activity Space: "Understand the Goal"

5.11.1 Activity: Elicit What the Company is Trying to Achieve

Context

Establishing the goal of the evaluation is one thing. Another is defining the more concrete achievements such as risk reports, quality issues reports, among others. Templates, if available, can be helpful for defining the expected achievements. I propose this Activity to increase awareness in the evaluation tasks by enforcing the idea that the progression of the evaluation adoption (see Alpha Evaluation Adoption) is influenced by concrete statements that outline the achievements towards the goal.

Goal Increase awareness and participation of stakeholders by concretely stating the outlines of the achievements that will eventually be observed along the evaluation.

Action to do At the beginning, broadly discuss the evaluation tasks. For each general task, discuss with stakeholders why they need to be engaged in the work and what are the expected achievements towards the evaluation goal. If possible, present examples from previous evaluations, confirming that the evaluation team has the rights to do this.

5.12 Activities in Activity Space "Present Results"

In this Activity Space I propose the Activity Present Partial Results. Practitioners and researchers should expect the final result to be composed of one or more of these partial results.

5.12.1 Activity: Present Partial Results

Context

In architecture evaluation practice, I have found that people tend to be more engaged in the evaluation if they see early, partial results from the effort.

Many times, these partial results provide concrete evidence that confirms previous conjectures from the stakeholders, which is appreciated. I have also observed that this is especially true for newer or younger stakeholders that have been struggling to understand "the why" of previous architecture decisions that have been made by previous architects that commonly do not work in the company anymore.

Presenting partial results helps to progress the Alpha Evaluation Adoption.

Goal Increase and/or maintain the participation of stakeholders in the evaluation effort.

Action to do Start showing results from the evaluation effort as soon as possible. Take advantage of evaluation methods by following the prescribed steps to define partial results to be shown and discussed before running the next steps (this sometimes requires a more iterative-incremental approach than is typically perceived from the architecture evaluation methods).

5.13 Activities in Activity Space "Understand the Practice"

In this Activity Space, I propose two Activities: Adapt Scenarios Examples and Present the Method. This Activity Space revolves around a proper understanding and engagement of stakeholders in the practice of evaluating a software architecture.

5.13.1 Activity: Adapt Scenarios Examples

Context Scenarios have gained attention in software engineering, especially as a quality assurance technique in the context of architecture evaluations [104]. Scenarios are commonly used in software architecture evaluation methods [6][34][37]. In the ideal case, scenarios

should be described in stimulus-response form, possibly including other features such as the measure used to verify the response [2].

In my experience, although scenarios are behind common tools such as use cases and user stories, practitioners often struggle to write them. I have found that adapting some general scenarios descriptions to the context in which the evaluation is taking place is an effective way to familiarize stakeholders with scenarios.

Goal Familiarize the evaluation *stakeholders* with the use and writing of scenarios.

Action to do Adapting examples of scenarios means taking the stimulus-response scheme and looking for real cases of the organization in which the architecture will be evaluated to create examples with real stimuli and responses. This Activity can be implemented progressively, or even prior to the presentation of the method that will be used for the evaluation. It is the responsibility of the evaluation leader to pay attention to the way of working and the work described by the organization in order to find cases that can be expressed as scenarios. These examples will be presented to the evaluation team to alleviate the scenarios writing "cold-start" issues. When dealing with a massive number of people involved in the writing of scenarios, this Activity can be presented to the people responsible for others so they can spread the examples to their teams.

5.13.2 Activity: Present the Method

Context

I have observed that, in many cases, running an architecture evaluation is the first formal quality-related activity. In this way, the practice becomes much more cloudy than the participants would like. One way to alleviate this problem is to increase awareness of what participants should expect. The architecture evaluation is facilitated (see Alpha Evaluation Adoption) because of the stakeholders awareness of the practice enactment.

Goal Increase stakeholders awareness of the method or approach that will be used.

Action to do Prepare a brief presentation of the method and deliver it to participants. The presentation should focus on expected outcomes, expected effort, and the responsibilities of the people in the organization. In other words, this is not a "teaching class" on "how to evaluate an architecture with a method". Rather, it is a presentation of what stakeholders should expect when conducting the evaluation with the method or approach.

5.14 Activities in Activity Space "Meet Entrance Requirements"

In this Activity Space, I propose the Activity Sign a Non-Disclosure Agreement.

5.14.1 Activity: Sign a Non-Disclosure Agreement

Context

Software architecture evaluations imply that the evaluation team has access to sensitive information of the organization. Sometimes, this forces organizations to demand compliance with a series of requirements before starting the evaluation process to facilitate organizational assets sharing, which is critical to ensure a fluid evaluation (see Alpha Evaluation Adoption). An example of these requirements is a non-disclosure agreement by which the parties involved in the evaluation agree the duty not to share information. This is particularly important when the evaluation team is external to the organization.

Often, I have observed that these agreements are reused from time to time. Thus, it is important to observe the agreement statements to avoid being compromised in too stringent specifications because the agreement document is being copied from past experiences, many of which do not relate to architecture evaluation and may include more stringent aspects than needed (this is especially important for researchers that would eventually want to publish an experience report).

Goal Alleviate a potential "cold start" issue in sharing and disclosing architectural assets.

Action to do Identify the information assets that would eventually be subject of the evaluation effort. Carefully review the non-disclosure agreement statement, especially observing for too stringent specifications. Sign the agreement and do not forget to comply with it. Recall that access to architectural assets implies responsibility, and not always is "full access" good because of this.

5.15 Activities in Activity Space "Do the Offline Work"

In this Activity Space, two activities are proposed: Record Organizational Learning and Print Architecture Documentation.

5.15.1 Activity: Record Organizational Learning

Context

In addition to the architectural knowledge created, recorded, and delivered in architecture evaluations, there is also organizational knowledge that contributes to organizational learning and is worth recording, especially for the next architecture evaluations if the organization plans to run the practice again. Alpha Evaluation Adoption also recommends recording organizational learning (organizational memory).

To a great extent, all of these Activities proposed are the result of careful knowledge articulation and recording.

Goal Preserve institutional learning and fuel continuous improvement.

Action to do The first action to take is to define specific checkpoints where organizational knowledge will be analyzed and recorded. This can be in a per-meeting fashion (i.e., after each meeting, a brief retrospective is held), although this frequency can be somewhat cumbersome for some contexts. What is recorded in a retrospective meeting is outside the scope of this work; however, a recommendation from the case studies is to focus on concrete and important events, and on what observable effects these events have, as well as evidence that supports the claims.

5.15.2 Activity: Print Architecture Documentation

Context

According to the Alpha Architecture Description, at some point, the architecture evaluation must have some kind of architecture representation. In my experience, some documentation is better understood if it is printed on paper rather than shown on screens. In addition, stakeholders can easily carry the printed documentation to other places for further analysis.

Goal Increase stakeholders awareness of the current software architecture.

Action to do Create printable images from the architecture descriptions (Alpha Architecture Description) and deliver them to the stakeholders involved in the architecture evaluation. Evaluate the suitability of the printed descriptions as not all descriptions will be appropriate (i.e., some will be better appreciated in screens). The idea is to provide concrete and observable architecture documentation to the stakeholders involved.

5.16 Activities in SEMAT Graphical Notation

Figure 5.2 shows the use of the Essence Standard graphical notation to represent the Activity Spaces of this proposal, along with the Activities presented in this chapter. The majority of the Activities are from the examples presented in this section.

5.17 Chapter Remarks

In this chapter, I present nine new proposed SEMAT Kernel Activity Spaces to cover the Activities that take part in evaluating software architectures. I also included fourteen Activities, each one being classified in the proposed Activity Spaces.

Both the Activity Spaces and Activities that I present were derived after careful elicitation from reflection-in-action-approached architecture evaluations in real-world contexts, including the two that are presented as case studies in this thesis.

Practitioners and researchers can take the Activities as modular units; that is, they can include them in their own architecture evaluation experiences as needed. However,

Figure 5.2: Example showing how Activities can be encompassed in Activity Spaces.

recommending and prioritizing Activities is outside the scope of this thesis, and it is a very interesting field of work that is left for future research.

Chapter 6

Case Studies

Several architecture evaluations have been run with the proposal of this thesis. However, as noted in Chapter 3, two of these experiences were chosen as case studies to be included in this work.

In this chapter, I present two cases in which an architecture evaluation has been conducted in real-world settings for real-world products. Each case follows a narrative form to present how this proposal is used in each architecture evaluation. This chapter also provides a discussion about the key aspects observed to provide contextual evidence that supports the suitability of this proposal to guide software architecture reviews.

In both cases, the evaluation leader took the role of the researcher.

6.1 Case Study Design Aspects

This work embraced the following guidelines for the design of case studies [32][105]:

- **Define the case using a real-world problem:** A key recommendation for the design of case studies is to avoid the use of "toy programs" or "toy examples [105][32][57]. In both case studies, the presented case is defined as the use of the proposed Alphas to guide a real-world software product architecture evaluation.
- **Use more than one data collection approach:** Both case studies relied on:
  - **Observation:** this is a first degree data collection strategy [105] than implies a

careful watching of the progression of the architecture evaluation. In both cases, the emphasis was put in "events". For this work, an "event" is a fact that occurs in the context of the evaluation endeavor that effects a change in an invariant condition of the evaluation. An example of an observable event is a "disagreement between participants in the way the evaluation tasks are run". This event makes a change in an invariant condition, for example, the "scenario brainstorming" changing to "scenario brainstorming stopped". Avoiding loosing focus on the research approach is critical, and thus only events that have a potential effect in the evaluation effort are considered. For example, given the previous definition, a "phone call" is also an event. However, unless the "phone call" has an effect of research interest, it will not be considered.

- **Retrospectives:** Brief retrospectives (15-20 minutes) were used as needed as the formal instance to discuss the progression of the evaluation, as well as impediments that can hinder the evaluation objectives. The retrospectives followed an unstructured approach [105] to allow the evaluation leader to develop the conversation based on the research interest. Nevertheless, the following issues (or issue questions) were always in the mind of the evaluation leader when running the conversation:
  - Are the evaluation participants understanding the current state in the progression of the architecture evaluation?
  - Are the evaluation participants clear on what has already been done in the architecture evaluation?
  - Are the evaluation participants clear on what the next expected activities are going to be done in the evaluation effort?
  - Does the evaluation team have an overview of the expected work products in the evaluation effort?

  In some retrospectives, interesting questions appeared that arose from the stakeholders. For example, in one of the retrospectives in Case Study 2 (i.e., in the military institution), an interesting question was asked by one of the military leaders: How much of the discovered architecture issues and risks were already known by the software and infrastructure teams? Although such questions are more re-

lated to the evaluation itself rather than to the use of the proposed Alphas, the discussion is always illuminating to understand the case.

The retrospectives were also used to discuss the classic "retrospective questions" such as what are the identified impediments to doing the work and what the team plans to do in the next days.

- **Archival data:** A third data collection approach is the use of archival data [105]. Archival data are coincidentally necessary in an architecture evaluation because it allows to define a starting point for eventual analysis in light of potential new architecture decisions. As noted in [105], the main problem with archival data is that despite being critical for the case, the data is already created and thus there is no control over how the data are generated.

- **Use of triangulation to improve the precision of case study research:** the use of triangulation, that is, broadening the viewpoints for approaching the studied object [105][106] is suggested to increase the precision of empirical case study research. In this work, two kinds of triangulation were used:

  - **Source triangulation:** this kind of triangulation promotes the use of more than one data source [105], where the same insights and issues are collected in different contexts. In this paper, two data sources are presented as Case Study 1 and Case Study 2.
  - **Methodological triangulation:** According to [105], this triangulation focuses on diversifying the data collection methods. In this work, observation, retrospectives, and archival data contributed to diversifying data collection methods.

The **goal** in both cases is to explore the suitability of the proposed Alphas to guide and assess an architecture evaluation endeavor. In both cases, **the case** is defined as the guidance of a software architecture evaluation using the proposed Alphas. In the first case, the context is a military institution where the architecture of a human resources system is the object of the evaluation. In the second case, the context is a public cancer-treatment oriented hospital where a health record system's architecture is the object of the evaluation. Both systems for which their architectures were evaluated exhibit the characteristics of legacy systems: they

are key for the organization's business or mission success [107], they are considered aged and complex and they encompass key domain-specific knowledge [108] which is still of significant value [109], they typically run on old platforms and maintenance is expensive [110], with code written following old techniques and in old languages [111], and pose significant challenges for software evolution to meet new functional and non-functional requirements [112].

In the first case, the Architecture Tradeoff Analysis Method (ATAM) was chosen as the evaluation method. This decision relied on the intention of the military institution to perform a formal architecture evaluation, for which a published and well-known method served the purpose. In addition, the ATAM has been reportedly used in other military institutions. This is an interesting example where the lack of experimental control is observed: the decision to use the ATAM was mostly beyond my control.

To strengthen the evidence from the use of this proposal, the second case was analyzed. In the second case, no specific evaluation method was used: the case was guided and assessed using this proposal alone.

I recall that in both cases an evaluation leader was present, taking the role of the researcher, and it was external to both institutions.

Table 6.1 summarizes the design aspects discussed in this section.

6.2 Case Study 1: Human Resources Software System in a Military Institution

This case describes the guiding and progression assessment of an architecture evaluation of a human resources software system in a military institution. The case involved a permanent evaluation team with 7 people and an approximate number of 50 stakeholders. Most of the stakeholders were day users of the system. Unlike Case 2, in this Case the evaluation team explicitly committed to the use of the Architecture Tradeoff Analysis Method (ATAM) [6].

Most evaluation methods prescribe actions to be done in a very general way. Often, examples are provided in books (examples for the ATAM method are provided in [6]). The proposal of this paper takes another approach: it provides concrete expected status for all of the essential elements (Alphas) that practitioners need to consider when evaluating an architecture. Each discernible progression status is known as a State and to achieve it the

Table 6.1: Summary of case study design aspects used in this work.

Design Aspect	Explanation
Goal	• Exploring the suitability of the proposed Alphas to guide and assess a software architecture evaluation endeavor.
Cases	• Case Study 1: using the proposed Alphas to guide the software architecture evaluation of a human-resources system in a military institution. • Case Study 2: using the proposed Alphas to guide the software architecture evaluation of an electronic health record system in a public, cancer-treatment oriented hospital.
Data Gathering Strategies	• Active observation: emphasis in events that change the active condition of the architecture evaluation. • Retrospectives: using a conversation-oriented, unstructured approach. • Archival data: review of archival data that is necessary for running the architecture evaluation.
Kinds of Triangulation	• Source triangulation: two different context in which an architecture evaluation takes place (Case Study 1 and Case Study 2). • Methodological triangulation: use of active observation, retrospectives, and archival data review.

evaluators must fulfill the concrete results indicated as checklist items. Given that States express an expected status rather than a general prescription of what to do, they are used to audit and check the progression of an architecture evaluation.

An interesting aspect of this case is that because the team followed the ATAM method, the order in which the Alphas were worked by the team members was to a great extent predefined. For example, Step 1 in the ATAM method indicates to present the method to all stakeholders. When completed, this step helps the team to reach State 2 of the Alpha "Evaluation Adoption". Still, the proposal is observed valuable in the Case: in the same example, the Alphas provide a good mechanism for periodically checking the status and progression of the architecture evaluation. The same happens with steps 2, 3, 4, 5, 6, 7, and 8, which are closely related to the Alphas "Architecture Description", "Quality Attributes", "Business Goals", and "Architecture Decisions", for which the order was mostly dictated by the sequence of the ATAM steps.

The following aspects can be attributed to the proposal of this work in this case:

- Evaluation method/approach selection: ATAM is an evaluation method that assumes that the method is already chosen. In turn, Alpha "Evaluation Adoption" begins with asking stakeholders to analyze evaluation methods/approaches before choosing one (State 1 in Alpha "Evaluation Adoption"). In this Case, this item of the checklist motivated to better understand the rationale behind the selection of ATAM.

- Concrete check points for determining the current status of an architecture evaluation endeavor: while ATAM prescribes steps that can be used to globally understand the progression, the Alphas provide, by means of their States, concrete items to check for progression and auditing purposes. The team uses these checkpoints for auditing progression.

- Concrete expected results for each State: the checklists are defined in a way that specifies concrete results that are needed before advancing to the next State in the progression of the evaluation. For example, the Alpha "Architecture Description" explicitly asks for specific models when advancing to State 2. The ATAM only requires one to present the architecture as a step to be conducted, leaving the evaluators to their experience to decide on the kind of architecture description to elaborate.

The first Alpha used in this case was "Evaluation Adoption". The main criteria the team used for choosing the ATAM method was that the method has been previously reported in military environments (see, for example, [113]). This decision was agreed and signed off, and the team advanced the Alpha to State "Method or approach chosen".

Before starting with the method, a half-day training was conducted with the aim of explaining the goals of an architecture evaluation and explaining the method to key stakeholders. All members of the evaluation team agreed on the expected effort and the expected tasks and work products. The team then advanced the Alpha "Evaluation Adoption" to the second State: "Method or approach integrated".

The evaluation was "officially" started by articulating and eliciting business drivers (step two in the ATAM). The major concern here was to translate the strategic goals into more product-level-related goals. For example, the goal of "cost reduction", which was explicitly stated by the institution, was translated into more specific goals such as "increasing the automation of many of the use cases that the system could support" (i.e., avoiding the costly effort of human intervention to satisfy use cases that were repetitive in nature and thus subject to a high degree of automation). Other more concrete goals were related to maintainability aspects (i.e., reducing the time and effort required for future feature additions) and to the dependency of costly old components that still have great importance in the system. The team then moved to State "Identified" of Alpha "Business Goals".

At this point, the ATAM prescribes moving to step three, that is, the architecture presentation. Here, the team started working with Alpha "Architecture Description" where it was found that the institution already had several general overviews for the architecture, although they were diagrams created in general-purpose diagramming software. Thus, in this Alpha the team started working towards state two: "Models developed". At the same time, the team started working with Alpha "Architecture Decisions" by identifying and understanding the main architecture decisions. The evaluation team also discussed the rationale behind many of these decisions, especially to understand why and how some of the old decisions were made. After meeting the criteria to reach the first State in Alpha "Architecture Decisions", the team sought to meet the criteria for the second State: "Explicit identification".

The next step in the ATAM (i.e., the identification of architectural approaches) moved the evaluation effort to State "Explicit identification" in Alpha "Architecture Decisions". The work carried out in this ATAM step was very technical in nature. The evaluation team

worked with software engineers and personnel from the technology infrastructure department. The same work with these stakeholders allowed the team to advance the Alpha "Architecture Description" to the State "Models developed", making strong use of a modeler software.

Following the identification of architectural approaches, the team continued with the following ATAM step: the generation of the utility tree. In this case, the utility tree was generated with the collaboration of the "first line" of stakeholders. In this step, the evaluation team started working with Alpha "Quality Attributes". At first, the team started identifying the quality characteristics. The use of the quality models from ISO 25010:2011 [73] helped the team in identifying quality attributes. Before committing to the use of both quality models (product and quality in use), the evaluation leader presented the quality models, and the team discussed its fitness for the software being evaluated. The quality standard was accepted, and then it was used as a guide. Initially, the team screened the quality characteristics and selected those that appeared to be critical to the system. Then, stakeholders started thinking about concerns and scenarios. At this point, Alpha "Quality Attributes" advanced to State "Identified". The team also advanced to State "Principles established" the Alpha "Business Goals" because the same discussion allowed the team to understand the rules and fundamental assumptions that govern the architecture. The identification of quality attributes, concerns, and scenarios in the generation of the utility tree brought about a better understanding of the business goals (in this case, the team preferred to say "mission goals").

For the next ATAM step, the analysis of architectural approaches, the officer in command of the department responsible for this software system development and maintenance was invited, and its participation was very important for the first part of this ATAM step. The discussion took about one full working day distributed in three calendar days. Much of the discussion concentrated on the prioritization of scenarios and a better articulation of many of them. The second part of this step was more technical, and the team worked toward better descriptions for the architecture and a better articulation for architecture decisions. The team then started mapping scenarios onto the architecture. At this point, the evaluation effort had met the criteria to advance Alpha "Architecture Description" to State "Completed", Alpha "Quality Attributes" to State "Tradeoffs understood", and Alpha "Evaluation Adoption" to State "Working well". The team also started to share the first partial results. The evaluation leader recommended this to maintain stakeholders engaged in the review.

The scenario brainstorming, which is the next step in the ATAM, was a bit tricky in this case. The evaluation team started to discuss the direction that the evaluation was taking. One of the stakeholders claimed "I don't understand what we are doing, what is the point of doing this". The evaluation leader presented the Alphas and marked the items achieved (in all Alphas) and briefly explained the current position and the next activities to expect with emphasis on the Alpha "Quality Attributes". This helped the team to regain attention to the current progression and the next steps expected. The evaluation team also recalled Alpha "Stakeholders" (one of the original in the SEMAT Kernel) to identify the users hierarchy in the institution (and express it in a diagram). Several hierarchies were found, and some of them very deep. The team quickly found that it would be nearly impossible to directly work with all stakeholders involved. The approach used in this evaluation to deal with this stakeholder-massive brainstorming was to invite the people responsible for these hierarchies to explain the scenario brainstorming step and, most importantly, to explain to them how a scenario is articulated as well as several hints to elicit them. The evaluation team also helped them prioritize the scenarios when they returned with the elicited ones. At this point, the evaluation has elicited almost 40 scenarios. An estimated 30% of scenarios were added in the brainstorming scenario step (other scenarios were just confirmation for the first ones elicited in the utility tree generation step). Roughly a 20% of the elicited scenarios were deemed as low priority, while quality attributes Interoperability, Modifiability, Performance, and Security comprised the key priority scenarios. Some scenarios were considered to be part of the quality characteristic "Functional correctness" while they might have been classified as "Integrity". Given the experience of the evaluation leader in previous ATAM-based evaluations, the team was instructed not to put much effort into this discussion, as the priority was to maintain the focus of the evaluation effort in the review rather in discussions about which quality characteristic has a better name for classifying some scenarios.

The ATAM calls for a new analysis of architectural approaches with all scenarios elicited up to this point. The new analysis improved the articulation of architecture decisions and scenarios. This allowed the team to address the quality attributes that are operationalized by specific scenarios. The team then moved Alpha "Architecture Decisions" to State "Recorded and Addressed" and Alpha "Quality Attributes" to State "Addressed". The team also advanced Alpha "Business Goals" to State "Addressed" because business (or mission) goals were addressed in the articulation of quality attributes and the mapping of scenarios onto

the architecture.

Finally, the evaluation team presented the results. A meeting with the officers in command of the departments involved in the use, development, and maintenance of this system was scheduled, including the ones that joined the evaluation endeavor at the scenario brainstorming step. The team then advanced Alpha "Evaluation Adoption" to State "Organizational memory accrued". The stakeholders involved expressed explicit interest in learning from this experience for future consideration.

Before starting the architecture evaluation, the stakeholders indicated that they were doing ad-hoc architecture evaluations. They reported the lack of a systematic approach to organize both architecture evaluation activities and results, and that the software architect role was distributed among two people. The introduction of the Alphas and their States established a set of actionable progression status checking mechanism, which acted in compliment to ATAM. The evaluation team continuously checked the current status and marked States as achieved (and therefore checklists were fulfilled), and then determined the progression and health of the evaluation by analyzing the States already "burnt" and the deviation from the expected items in the checklists. They also reported that results from the ad-hoc evaluations were not systematically approached. Introducing the Alphas permitted the participants to structure both the presentation and the evaluation report along the Alphas with the following sections: introducing the approach (related Alpha: "Evaluation Adoption"), describing the architecture (related Alpha: "Architecture Description"), describing the business and quality goals (related Alphas: "Business Goals" and "Quality Attributes"), and the approved/revised/rejected architecture decisions and the rationale behind their status (related Alpha: "Architecture Decisions"). In addition, other interesting changes observed are:

- The introduction of the Architecture Decisions Records (ADRs), making the team aware of this technique for recording decisions.
- The introduction of a modeler software.

While both ADRs and the use of modeler software are changes that can be discovered by other means, in this case the introduction of the Alphas and the explicit statement for progressing towards systematic decisions recording and architecture descriptions as models made the team aware of these aspects.

In the following, a summary of the main value added by the proposed mechanism to this ATAM-based architecture evaluation is provided. For each step of ATAM, the most related Alpha/State is presented along with the main observed value added to the architecture evaluation. This summary is provided to aid readers in understanding what aspects can be attributed to the use of the proposed mechanism (i.e., the Alphas) in this ATAM-based evaluation. Readers should also note that, in practice, it is not possible to define strict boundaries for the Alphas in terms of the ATAM steps. For example, Alpha "Evaluation Adoption" starts before Step 1 "Present the method" in ATAM; however, in the following discussion, the Alpha "Evaluation Adoption" with State 1 progressing towards State 2 appear related to Step 1 in ATAM.

1. **Present the method:**

   - **Related Alpha/State:** Evaluation Adoption (achieving State 1).
   - **Main Value Added:** Concrete guidance: find arguments for justifying the use of ATAM.
   - **Evidence:** Recorded criteria for the use of ATAM: the main recorded criteria is the availability of reported uses of ATAM in military institutions. The availability of examples of the use of ATAM was also noted as an important aspect to motivate the selection of ATAM.

2. **Present the business drivers:**

   - **Related Alpha/State:** Business Goals (State 1 to State 2).
   - **Main Value Added:** No specific value added: State 1 was achieved by preparing a business goals and architecture drivers presentation, which is already called by ATAM. Although a recommendation was made by the evaluation leader, this recommendation is not attributable to the use of the Alpha.
   - **Evidence:** N/A.

3. **Present the architecture:**

   - **Related Alpha/State:** Architecture Description (State 1).

- **Main Value Added:** Concrete guidance: leverage (already prepared) deployment diagrams (at service-level), avoiding specific configurations.
- **Evidence:** Two deployment diagrams (no specific modeling language used), presented in Power Point slides.

4. **Identify the architectural approaches:**

   - **Related Alpha/State:** Architecture Decisions (State 1 to State 2) and Architecture Description (State 1 to State 2).
   - **Main Value Added:** Concrete guidance: articulate/model specific configurations and considering appropriate views.
   - **Evidence:** First specific configuration modeled: staff career system modules and database with views for components/connectors and interoperability with other systems.

5. **Generate the quality attribute tree:**

   - **Related Alpha/State:** Quality Attributes (achieving State 1) and Business Goals(State 1 to State 2).
   - **Main Value Added:** Concrete guidance: adopt a quality model to avoid lack of, and over, representation of quality attributes. Business Goals advancement is considered an effect of the discussion given by the first State in Alpha Quality Attributes.
   - **Evidence:** ISO 25010:2011 quality model adopted.

6. **Analyze architectural approaches:**

   - **Related Alpha/State:** Evaluation Adoption (State 2 to State 3).
   - **Main Value Added:** Concrete guidance: enforce the delivery of partial results; enforce the use of tools for tracking progress.
   - **Evidence:** Articulation of Quality Attributes into concrete requirements delivered to the officer in command; use of spreadsheets to track the progress of the evaluation.

7. **Brainstorm and prioritize scenarios:**

   - **Related Alpha/State:** Quality Attributes (State 2) and Evaluation Adoption (State 3).
   - **Main Value Added:** Concrete checklists to audit progression and to determine (1) what has been achieved and (2) what is left for the activity (related Alpha: Quality Attributes).
   - **Evidence:** Spreadsheet with four columns: Alpha, State, Checklist, and Done (per checklist item). When activities were carried out to achieve the checklist items, the achieved item was marked as Done.

8. **Analyze architectural approaches:**

   - **Related Alpha/State:** Architecture Decisions (State 2 to State 3).
   - **Main Value Added:** Concrete guidance: recording structured decisions subject to version control.
   - **Evidence:** Git-based repository containing ADRs (two folders: one for approved decisions and another for rejected/deprecated decisions).

9. **Present results:**

   - **Related Alpha/State:** All Alphas and primarily the final States of each Alpha.
   - **Main Value Added:** Reports structured following the argument given by the progression of each Alpha, with special emphasis in the last States of each Alpha (the last States represent the desired status of the results of each Alpha).
   - **Evidence:** Both presentation and written report with the following sections: introducing the approach, describing the architecture, describing the business and quality goals, and the approved, revised, or rejected architecture decisions and the rationale behind their status.

Table 6.3 presents a summary of the contrast before/after within this case. In the contrast summary, for each key attribute a before and after recounting is provided showing which important changes attributable to the proposal were observed.

Table 6.2: Key observed effects in Case 1.

Observed Effect	• Effect 1: Actionable guidance and progression auditing. • Effect 2: Common ground for reporting. • Effect 3: Regain focus in scenario brainstorming.
Related Alpha	• Effect 1: All Alphas. • Effect 2: All Alphas. • Effect 3: Alphas Quality Attributes and Architecture Decisions.
Evidence Observed	• Effect 1: calling stakeholders for analyze and justify the use of the chosen evaluation method. Stakeholders understanding concrete expected results for each State in each Alpha, although in this case the ordering of attention of Alphas was dictated by the method. Marking the States achieved as "done" and using the checklists from each State as check points for auditing the progression and health of the architecture evaluation. • Effect 2: use of the Alphas to define both the presentation and written report structures (i.e., the Alphas supported the argument of these communication artifacts). • Effect 3: during discussion about the relation between a planned activity (scenario brainstorming) and the evaluation goal, the Alphas and their States were reviewed to understand how this activity fits in the evaluation effort; stakeholders regained focus on the architecture evaluations by understanding the relation between this activity and the others to the evaluation goal.
Action	• Effect 1: define activities for fulfillment of checklists of Alphas. Communication to stakeholders of the status of the architecture evaluation with specific references to the States of the Alphas. • Effect 2: elaborate final presentation and report with sections containing the results that appeared in the checklists of each State. • Effect 3: stop the discussion and re-plan the activity for eliciting key requirements and scenarios (State 1 to 2 in Alpha "Quality Attributes"). The discussion occurred at the end of the session and thus the elicitation continued on next meeting.
Consequence	• Effect 1: a SEMAT-based mechanism (i.e., the Alphas) as a guiding and progression assessment/auditing tool was introduced. • Effect 2: architecture evaluation results presentation and reporting were structured using the Alphas to guide the sections' content. • Effect 3: a SEMAT-based mechanism (i.e., the Alphas) for clarifying the relation between the evaluation activities and the evaluation goal was introduced.

Table 6.3: Within-case contrast for Case 1.

Progress tracking	• Before: Informal, ad-hoc evaluation progress tracking. • After: A spreadsheet was created to track progression of Alphas in terms of the achievement of the items of the checklists of the States.
Reporting	• Before: Non-structured reports; all reporting made in verbal, tacit form to related officers in command. • After: Alphas provided a common ground for structuring both presentation and written report. Included in this aspect is the fact that a modeler software was adopted for the creation of the architecture models that were not only part of the analysis, but also of the reports.
Decision handling	• Before: Architecture decisions not completely identified and in tacit form. Most of the decisions knowledge was transferred by word of mouth (especially the historical decisions for which no original decision maker was present). • After: Architecture decisions written down in structured format (ADR), and stored and versioned in a Git-based environment.
Stakeholder participation	• Before: Ad-hoc stakeholder participation (i.e., stakeholders involvement as required). • After: Stakeholders alerted of their eventual participation because the Alphas were presented before the architecture evaluation starts. A stakeholder in particular took the role of checking the progression of the evaluation endeavor.
Organizational memory	• Before: Tacit knowledge about previous informal architecture evaluations; most of the knowledge was transferred by word of mouth. • After: A general-purpose repository was defined and used for storing meeting reports and especially templates generated in the evaluation.

6.3 Case Study 2: Health Record System in a Public Cancer-treatment Oriented Hospital

This case describes the guiding and assessment of the progression of a software architecture evaluation for an electronic health record system that was in production and operation in a public cancer-treatment oriented hospital. The number of people directly involved in the evaluation was approximately 15 people, with an evaluation leader (external and taking the role of the researcher), evaluation supporters (external), information technology infrastructure manager, software developers, innovation leaders, and domain experts as the most important stakeholders observed. The evaluation took approximately 4 full days (8 hours each), but the evaluation sessions were not held continuously. Rather, in general, half-days were used for meeting with stakeholders, along with offline work such as preparing architecture descriptions and eliciting architecture decisions.

Approximately half of the meetings were held online using a video-based collaboration platform. Electronic mail was also used to share information about the evaluation itself. When the evaluation sessions were held online, the Alpha State Cards, with their states and checklists, were shared to participants using the "sharing screen" option in the video-based collaboration software.

This architecture evaluation took place in the context of a consulting service. Therefore, the original SEMAT Kernel Alphas were also intensively used. An interesting case was observed when using the Alpha "Stakeholders". When using the Alpha "Stakeholders", two more stakeholders were identified. These two new stakeholders were not initially considered. The new identified stakeholders were considered critical for the system's architecture, and thus they were invited to join the architecture evaluation.

The evaluation effort started by using the Alpha "Evaluation Adoption". The participants were told in general terms what an architecture evaluation is and arguments for possible review approaches were provided. The evaluation team agreed that the approach to be used is a mix of decision-challenging and scenario-based perspectives. The mixing of these two approaches was the reason the team agreed to choose "an approach" rather than "a method". After formally choosing the approach, the team moved to the State "Method or approach chosen" from Alpha "Evaluation Adoption".

Due to the good availability of the stakeholders, the work with Alpha "Architecture Description" followed Alpha "Evaluation Adoption". The first State for this Alpha is "General overview", which is reached if a high-level architecture description overview shows the main components and their deployments. This description did not exist at the time of this evaluation; therefore, the team started working towards elaborating this artifact. Most of the knowledge about the architecture was available in a tacit way. Some knowledge was also captured in an earlier effort by hospital personnel to model hospital processes, especially those related to health care record management. The Alpha's next State asks for creating models. Thus, after creating more elaborated diagrams in a whiteboard, offline team work consisted in quickly translating these diagrams to models using UML in a modeler software.

At this point, it was found useful to review the Alpha "Stakeholders" which is an original Alpha in the SEMAT Kernel. Following the progression path of this Alpha to the third State ("Involved"), the team found that communication with two important stakeholders (both related to biomedical engineering) was overlooked. Involving these two new stakeholders opened up a lot of valuable scenario-related information (at this point, still expressed in an ad-hoc way).

Slightly after starting the progression of the Alpha "Architecture Description", and due to the lack of explicit and recorded knowledge, the team started working with Alpha "Architecture Decisions". This Alpha asks for "Tacit identification" of decisions in order to reach the first state. Most of the architectural knowledge was in this form. Assessing this knowledge was the most intensive part of the evaluation in terms of stakeholders' participation. As soon as the team noticed that the knowledge was in tacit form and at the same time widely agreed by stakeholders, the team decided to move towards working towards the State "Explicit identification" of the Alpha "Architecture Decisions".

The next Alpha in consideration was "Quality Attributes". With this Alpha the team started by explicitly recognizing the key quality attributes for the systems being evaluated. The quality models for product and system use from the ISO/IEC 25010:2011 "System and software quality models" [73] were used to help the team in the identification of quality attributes. The team mapped several "qualities" that appeared in the many health care related laws, especially those related to the ownership, privacy, integrity and confidentiality of electronic health records. The team then advanced this Alpha to the first State: "Identified".

The Alpha "Business Goals" was worked in parallel with the "Quality Attributes" one. In

this case, the team held several meetings with stakeholders dealing with the current strategic concerns of the Hospital. After analyzing all business-related information, the team found that the Hospital has a clear view and mission in relation to one critical aspect: the digital transformation which included improvement of current software-supported processes. The team found clear statements that were already documented and also found that people in charge of the view and mission were clear about their thoughts and that the information already documented was also appropriate for the evaluation of the current system's design. Thus, this Alpha was advanced to the second state "Principles established".

Because with the progression of previous Alphas the team also discussed the evaluation implications, the team had also implicitly discussed the evaluation approach and the expected results. The team at this point was also presenting preliminary results from the evaluation, as required by the next State of the Alpha "Evaluation Adoption". The team then moved this Alpha to the second state: "Method or approach integrated".

The work continued to improve the architecture descriptions and better understand current architecture decisions. The team held two meetings especially devoted to this topic, where stakeholders from development and infrastructure departments joined the evaluation. At this point, the team finally finished the architecture models that started being elaborated in previous meetings, and thus the Alpha "Architecture Decisions" was advanced to the State "Models developed". In addition, the team continued working towards State "Explicit identification" of Alpha "Architecture Decisions".

Due to recommendations from development and infrastructure stakeholders, the team continued to work with Alphas "Business Goals" and "Quality Attributes". The team held other meetings with management hospital's stakeholders' that were directly related to this software architecture, as well as people responsible for health-related technology innovation initiatives. The team continued working with Alpha "Quality Attributes" by identifying main tradeoffs between current identified attributes and in consideration of current architecture decisions. As a result, stakeholders agreed on the understanding of the impact of current and potential decisions on quality characteristics and therefore Alpha "Quality Attributes" advanced to State "Tradeoffs understood"

After continuing to work with Alphas "Quality Attributes" and "Business Goals" the team held two meetings to discuss the fitness of the current architecture decisions for the articulated business goals and quality attributes. The team also started working on recording

decisions in ADRs [1], an approach that one of the team members has used before. Using the ADR format, the team identified decisions that are good, decisions that require more work, and decisions that are deprecated and sometimes replaced by new decisions. In parallel, Alpha "Evaluation Adoption" advanced to State "Working well" as in the brief retrospectives the team members gained consensus in that the architecture evaluation objectives were being met. The team also started presenting the first preliminary results of the evaluation.

When the team decided that reviewing the architecture decisions was finished, the Alpha "Architecture Decisions" was advanced to State "Recorded and addressed". Given that decisions were reviewed in light of business goals and quality attributes, the Alphas "Business Goals" and "Quality Attributes" were advanced to their respective final States, that is, "Addressed".

At this point, the team continued to improve the current architecture descriptions in order to provide the final report, and thus Alpha "Architecture Description" advanced to its final State: "Completed". The team also started working towards finishing the final report which consisted of a document summarizing key results from the evaluation. As the team also took notes about the progression of the evaluation in brief retrospectives, the Alpha "Evaluation Adoption" moved to the State "Organizational memory accrued".

At the end of the evaluation, the team has challenged about 10 key architectural decisions related to almost 20 priority scenarios. The key quality attributes were Privacy, Integrity, Confidentiality, and Interoperability. The last was related to scenarios describing how the hospital integrates and interoperates with the central metropolitan health care service.

Before starting the evaluation, in this case the stakeholders indicated the need for an architecture evaluation to determine how far the software system was from several goals, including those of digital transformation. The team reported having no previous experience or intention in formally evaluating the architecture. The role of software architect was very diluted among developers, and I believe that this is the reason for the lack of understanding of how to conduct an architecture evaluation. The introduction and brief review of the Alphas made stakeholders aware of what an architecture evaluation is and what to expect as results. In the evaluation itself, and given that the team had no experience in architecture evaluations, the guidance was step by step following each Alpha and their States. As the team did

[1] "Documenting Architecture Decisions", Michael Nygard, 2011, `https://www.cognitect.com/blog/2011/11/15/documenting-architecture-decisions`.

not have experience in architecture evaluation, no other mechanism for checking progression was in place. The progression and health assessment for the architecture evaluation was introduced by using the Alphas, marking the States already "burnt" while leaving States of Alphas not achieved as unmarked. The Alphas permitted the participants to structure the evaluation report (which before the case was non-existent) along the Alphas with the same sections as in Case 1: introducing the approach (related Alpha: "Evaluation Adoption"), describing the architecture (related Alpha: "Architecture Description"), describing the business and quality goals (related Alphas: "Business Goals" and "Quality Attributes"), and the approved/revised/rejected architecture decisions and the rationale behind their status (related Alpha: "Architecture Decisions"). The introduction of the Alphas, especially the Alpha "Architecture Decisions", made the team aware of the Architecture Decisions Records (ADRs). Unlike Case 1, the team was already using a software for modeling software architectures.

Table 6.4 provides a summary of the key observed effects of using the Alphas in the Case 2.

Table 6.5 presents a summary of the contrast before/after within this case. For each key attribute, a before and after description is provided showing the important changes attributable to the proposal that were observed.

6.4 Case Studies Discussion and Key Insights

The two cases presented in this work exhibit some commonalities. One thing in common is that in both cases all Alphas reached their last State when the evaluation effort finished. The researchers observed this to be very natural, that is, no resistance was observed when enforcing the rule that, for a State to be achieved, the complete checklist must be fulfilled. In both cases, it was observed that the teams were eager to increase their knowledge about software engineering and architecture quality practices, and I believe that this contributed to following the complete progression path of each of the proposed Alphas. I believe that when cost and time constraints arise, a more relaxed rule for achieving a State could eventually be used: for example, is it strictly required to fulfill "organizational memory accruement" (an item of the checklist of State 4 in Alpha "Evaluation Adoption") to consider an architecture evaluation as properly conducted?. This is an interesting issue, but more study is required to provide a lightweight version of the proposal.

Table 6.4: Key observed effects in Case 2.

Observed Effect	• Effect 1: Actionable guidance and progression auditing. • Effect 2: Common ground for reporting.
Related Alpha	• Effect 1: All Alphas. • Effect 2: All Alphas.
Evidence Observed	• Effect 1: stakeholders understanding what is required to advance in each architecture evaluation Alpha. Marking the States achieved as "done" and using the checklists from each State as check points for auditing the progression and health of the architecture evaluation. This was especially valuable when the new stakeholder joined the architecture evaluation, who quickly understood the current position in the evaluation effort and her responsibility in the next activities. • Effect 2: use of the Alphas to define the written report structure (i.e., the Alphas supported the argument of these communication artifacts). In this Case, the team has no experience in reporting architectural aspects of the system.
Action	• Effect 1: define activities to fulfill States' checklists. Communication to stakeholders of the status of the architecture evaluation with specific references to the States of the Alphas. Communicate the new stakeholder what has been done in the evaluation and what is remaining. • Effect 2: elaborate written report with sections containing the results that appeared in the checklists of each State.
Consequence	• Effect 1: a SEMAT-based mechanism (i.e., the Alphas) as a guiding and progression assessment/auditing tool was introduced. • Effect 2: architecture evaluation results reporting was structured using the Alphas to guide the report sections' content.

Table 6.5: Within-case contrast for Case 2.

Progress tracking	• Before: No formal or informal architecture evaluations done before. Only simple technical architecture analysis previously done by one of the developers. The analysis and adaptation of the architecture was done in an as-needed way. The adaptation progression tracking was informal and mostly tacit. • After: A spreadsheet to track progression of Alphas in terms of the achievement of the items of the checklists of the State was created and adopted for the architecture evaluation.
Reporting	• Before: As the previous architecture analyses were done by one person and with technical emphasis, only tacit reports existed, and many of them with knowledge vaporized. • After: The Alphas provided a common ground for structuring the written report of the architecture evaluation. The last States of each Alpha were used to guide the explanation of results.
Decision handling	• Before: Most of the architectural knowledge in tacit form. History of the decisions is not well maintained because the team was composed of new developers. • After: Architecture decisions articulated in ADRs and stored under a Git-based version control system.
Stakeholder participation	• Before: Stakeholders considered only for providing requirements. The only previous architecture analyses were done for small requirements implementations and with technical emphasis only. • After: Alphas presented to the evaluation team and therefore the eventual stakeholders were alerted about their potential involvement. Nevertheless, the inclusion of a new stakeholder in the architecture evaluation is not attributable to the mechanism itself as part of this effect was, as explained in Section 6, observed due to the use of one of the original Alphas in the SEMAT Kernel.
Organizational memory	• Before: Organizational memory available as processes, although many of them were related to the hospital services and not to the development activities. • After: The general-purpose repository was used to record templates, meeting records (including retrospectives), and the draft written report (which was stored as a sample for eventual future architecture evaluations).

Another aspect that is shared by both cases is that the two systems are considered as legacy ones and as such they are still in use and actively maintained and supporting key processes in these organizations. In addition, the two cases were observed in development contexts that can be considered as more classic development styles, although not strict waterfall-like approaches. It was also found that both teams were eager to adopt software practices, even when those practices implied disrupting the development tasks that were mostly oriented to maintenance and support. The stakeholders of both institutions were eager to embrace more agile approaches.

Mostly due to the nature of the organization, in the first case the evaluation meetings were required to be held in person at the institution's facilities. In the second case, it was possible to carry some of the evaluation meetings online (i.e., video conference).

In the first case, this proposal was used as a complement to the well-known Architecture Tradeoff Analysis Method (ATAM), while in the second case, the team made use of the Alphas alone. Due to the "real-world" nature of both cases, decisions such as the selection of the ATAM method were outside the control of this research (as noted in Section 6, the Alpha "Evaluation Adoption" was still used in the beginning of the evaluation to understand the rationale behind the selection of ATAM).

An interesting insight in the first case is that even when a method is present, this proposal was valuable because it provides engineering teams with actionable guidance to determine the current position in the progression of the evaluation and to recognize the future steps the evaluation team needs to focus on. In practice, this means that some of the team members started consulting the Alpha cards and started working beforehand on the necessary elements for the next steps, that is, in a more proactive way. Although the methods also provide guidance expressed in a series of steps, they lack explicit and concrete and practical guidance on how to enact an architecture evaluation (this is one of the drawbacks reported in the literature; see Section 2 for a discussion on this). Typically, practitioners can find some books devoted to evaluation methods in which authors express their recommendations from the practice, but I believe that this is not an expeditious way in which an architecture evaluation can be guided (especially in teams interested in running their own architecture reviews). In addition, I believe that it is not realistic to suppose that people engaged in an evaluation team would read books about this topic while implementing an architecture review.

The Alphas were also key for gaining attention to the architecture evaluation. This was observed in both cases; however, in the first case the Alphas proved to be key to regain focus on the architecture evaluation tasks when the team lost the orientation because of agitated discussions regarding the way the stakeholder-massive scenario brainstorming was being held.

The evaluation teams used Alphas (although not with graphical notation) to present an overview of the evaluation practice and what stakeholders could expect. This allowed not only to prepare a better presentation of the evaluation approach, but also stakeholders to use the concrete guidance that Alphas provide to begin planning their own schedules and to think about the future efforts expected.

Unlike other experiences in which I have used the SEMAT Kernel with the original Alphas to assess software development efforts (two of these experiences were published [114, 115]), in this case I was not asked how this proposal aligns with other frameworks such as the CMMI. I have been told in the past that if the teams were going to adopt the SEMAT Kernel, a key concern is how much the Kernel is aligned with such frameworks to avoid work duplication (in other words, the main concern from practitioners is that if all the work needed to progress in the Alphas from the SEMAT Kernel is aligned with the work needed for progressing through maturity levels in maturity frameworks). I believe that in the case of the architecture evaluation Alphas such concerns were not observed because there are no widely used frameworks or standards such as the CMMI oriented towards guiding and assessing an architecture evaluation endeavor.

When running these evaluations, I also observed that the teams were interested in learning more about architecture evaluation by following the indications of the proposed Alphas. Although this is not specifically designed to guide an architecture evaluation training, I observed that it actionably allowed team members to guide their own learning about specific topics in architecture reviews. In the second case, the evaluation leader was surprised to see one of the stakeholders joining the following evaluation meetings with new knowledge about the architecture evaluation effort, even considering that this participant was not coming from the software engineering domain.

A final remark is that this proposal facilitated better organizing of architecture evaluation reports. In both cases, the reporting consisted of a presentation and a document. The Alphas provide a concise and concrete architecture evaluation practice structure, and this was used to define a common structure and argument for both reports. At the same time, this common

structure makes the reports comparable.

In summary, for both cases the observations that were made to consider a State as achieved in each Alpha are described:

- Alpha: Evaluation Adoption
  - State 1: Discussion about the evaluation approach done and agreement on the approach to be used reached.
  - State 2: The approach has already been presented to all participants and partial results from the evaluation execution are being delivered (especially to high management stakeholders).
  - State 3: Stakeholders agreed that the evaluation met their expectations. Tools for writing down partial results and archiving artifacts is shown to be consistent. When a critical deviation from the planned activities appeared, actions were taken to correct the course of the evaluation.
  - State 4: Retrospectives carried out after each in-person evaluation meeting. Tips for the next architecture evaluation written down.
- Alpha: Quality Attributes
  - State 1: First set of quality attributes elicited. Quality model in use is agreed.
  - State 2: Quality attributes prioritized and articulated into explicit requirements and tradeoffs determined.
  - State 3: Priority quality attributes and their respective requirements analyzed considering their support by the architecture.
- Alpha: Architecture Description
  - State 1: First architecture overview presented. No details of the components are shown.
  - State 2: Relevant architecture configuration views created and stored in version control system.

Table 6.6: Summary of Case Studies

Characteristic	Case Study 1	Case Study 2
Type of institution	Military.	Public (i.e., state-run) cancer-treatment oriented hospital.
Main function	Human Resources Management System.	Electronic Health Record System.
Type of system	Legacy, still in use and critical.	Legacy, still in use and critical.
Method or approach	Use of the Alphas and the ATAM (Architecture Tradeoff Analysis Method) for guiding and assessing progression.	Use of the Alphas for guiding and assessing progression with a decision-challenging focus.
Number of stakeholders involved	$\sim$ 60 people, including a permanent evaluation team of 7 people, with one evaluation leader (external to the institution).	$\sim$ 15 people, including an evaluation leader and evaluation supporters (both external to the Hospital).
Main observations (specific to one case)	• As a compliment for the method, Alphas provide actionable guidance to determine the progression in the evaluation as well as future expected work to complete the review. • In complex scenario brainstorming, Alphas allowed to regain focus in the architecture evaluation. • The order in which Alphas were worked by the team appeared to be predefined because of the use of a specific evaluation method.	• Alphas alone provide enough actionable guidance for running the architecture evaluation, as well as an appropriate tool to determine current progression of the architecture review effort. • Use of the Alphas, especially Alpha "Evaluation Adoption" influenced the early results delivering.
Main observations (in both cases)	• Alphas provide a common ground for organizing final reports (presentation and document). • Guided by the Alphas, stakeholders exhibit interest in autonomous learning about the architecture evaluation topic. • Alphas provide general awareness of the evaluation endeavor, especially in the understanding of the expected outcomes, before the evaluation starts.	

  - State 3: Complete description (for evaluation purposes) created. Models created using UML and stored in the version control system. In addition, the models created are being used to document the system being evaluated.

- Alpha: Architecture Decisions
  - State 1: Architecture decisions identified, some rationale identified. Written on the blackboard.
  - State 2: Architecture decisions explicitly identified and written down in version control system.
  - State 3: Architecture Decision Records created for each decision. Also maintained in the version control system.

- Alpha: Business Goals
  - State 1: Business goals and architecture drivers determined and written on the whiteboard.
  - State 2: Business goals articulated into concrete quality goals and used in the evaluation.
  - State 3: Stakeholders agreed that the architecture is supporting the business goals to a high degree.

The reader should note that the observations presented correspond to the cases presented in this article. These observations are not mandatory for other evaluations; they can be used as examples.

In the following, for each of the reported observed effects (see Tables 6.2 and 6.4) a discussion of the evidence observed is presented. The discussion is articulated around each observed effect, and then the most involved Alpha(s) and its implication are detailed. Items that were more important for the observation are referenced to the identification given in the Appendix F.

1. **Observed effect: Actionable guidance and progression auditing**
   - Involved Alpha(s): Evaluation Adoption, State 3, especially items EA.S3.1 and EA.S3.2.

- Evidence Type: Creation and adoption of a simple spreadsheet with four columns (Alpha, State, Checklist Item, and Done).
- Timing: In both cases, State 3 was achieved in session 4 of the architecture evaluation (this is dependent on the particular architecture evaluation).

2. **Observed effect: Common ground for reporting**

   - Involved Alpha(s): Quality Attributes (especially State 3, item QA.S3.2), Architecture Decisions (especially State 3, item AD.S3.1), Architecture Description (especially State 3, item ADS.S3.2), Business Goals (State 3, item BG.S3.2), and Evaluation Adoption (especially State 1, items EA.S1.1 and EA.S1.2, and State 3, item EA.S3.1).
   - Evidence Type: Written reports and presentation (in Case 1) structured in the following form: introducing the approach (explaining activities done and justification for the evaluation method/approach), describing the architecture (a selection of the models developed and related decisions), describing the business and quality goals (prioritized quality and business goals with tradeoff analysis result and how the architecture supports them), and the approved, revised, or rejected architecture decisions and the rationale behind their status. Each section revolved around the specific items of the involved Alphas.
   - Timing: The written reports and the presentation in Case 1 were iterated from the early stages of the architecture evaluation, as the proposed mechanism instructs to deliver partial results to maintain stakeholders engaged. However, the finalization of the reports was made in the last days of the architecture evaluations. In Case 1, this work can be considered as part of the evaluation (Step 9 in ATAM); in Case 2, this work can be considered outside the evaluation as the Alphas do not explicitly mandate a "final report".

3. **Observed effect: Regain focus in scenario brainstorming (only observed in Case 1)**

   - Involved Alpha(s): Quality Attributes (especially States 2 and 3, items QA.S2.3 and QA.S3.1) and Evaluation Adoption (especially State 2, item EA.S2.2).

- Evidence Type: The most important artifact is the spreadsheet that was used to check the progression regarding the Alpha "Quality Attributes".
- Timing: In Case 1, this happened in the scenario brainstorming ATAM step, between sessions 4 and 5 of the architecture evaluation.

6.5 Chapter Remarks

In this chapter, I present evidence gathered from two case studies in two real-world contexts. The two software systems evaluated are legacy systems and the two organizational contexts are regarded as large with more classic development approaches, although agile practices were being incorporated.

Evidence shows that the proposal is a good vehicle for guiding an architecture evaluation in these contexts, as well as appropriate for assessing and auditing the progression of the evaluation efforts.

Chapter 7

Survey Analysis

In addition to the case studies presented in chapter 6, a survey was carried out to ask practitioners about the perceived utility of the proposal for certain aspects in evaluating software architectures.

The purpose of this survey is to complement the evidence gathered from the case studies with a wider variety of practitioners.

The presentation of the proposal and the survey itself was made in an online fashion (i.e., through the Internet). In total, 250 practitioners were contacted and 33 valid responses were received (13% response rate). The figures show a relatively low response rate, although it has been reported that online surveys tend to suffer from lower response rates when compared with other types of surveys [116]. In addition, I believe another key factor for this relatively low response rate is what I decided to call as "proposal understanding overhead". By this term, I mean the overhead effort required before answering the survey: practitioners need to understand the proposal, which is an intellectual task that requires a non negligible effort, and then start answering the survey.

In the following lines, I describe the survey and analyze the results.

7.1 Survey Design and Operationalization

The survey was designed with four sections:

- The first section asked practitioners to rate their self-perceived experience in software

architecture and in software architecture evaluation (both in formal and informal ways).

- The second section asked participants to rate, on a traditional five-level Likert scale, the perceived helpfulness of the proposal in several aspects.
- The third section asked participants to determine the relative importance of the Alphas for evaluating a software architecture.
- Finally, the fourth section allowed participants to express their thoughts or comments in an open text field.

7.1.1 Survey Design

The first section was included to gain insight regarding the self-perceived experience in software architecture and software architecture evaluation. I asked respondents to rate their experience on a five-level Likert scale: no experience, low experience, average experience, moderate experience, and high experience.

In the second section, I asked participants about the helpfulness of the proposal regarding the following aspects:

- Set in order and maintain ordering in the software architecture evaluation process.
- Avoid work duplication.
- Maintain expectancy and scope controlled.
- Maintain costs bounded.
- Assess when a software architecture evaluation is in problems.
- Be more objective.
- Improve planning.
- Solve in time problems that emerge in complex architecture evaluation.
- Define concrete tasks.

- Define concrete roles.
- Define concrete work products.
- Share knowledge about how a software architecture evaluation is run.
- Define and select tools to facilitate evaluation.

The question was "Considering your experience, this proposal for guiding a software architecture evaluation is helpful for: {aspect}." Respondents were asked to rate the helpfulness in a five-level Likert scale: strongly disagree, disagree, neutral, agree, strongly agree.

The third section asked participants to rate the relative importance of the Alphas (Architecture Description, Business Goals , Architecture Decisions, Architecture Evaluation Adoption, Quality Attributes) presented in the proposal for the sake of running an architecture evaluation.

Finally, the fourth section provided a text field for respondents to express their thoughts and comments in an open manner, beyond what the survey structure allowed.

7.1.2 Survey Operationalization

The survey and the proposal description were presented in Microsoft Forms. The form presented the proposal in the first page with a summary of the Alphas and their states. The description included links to further details.

The form link was shared to practitioners in two main ways:

- By using personal contacts: known practitioners working in the software industry were contacted.
- By sharing the link in personal messages through two LinkedIn Premium Accounts.

All respondents answered the survey using the same form link, even those who were contacted as personal contacts. Most of the participants were reached through LinkedIn.

In total, almost 250 practitioners were reached. A total of 33 valid responses were received, giving an approximate 13% response rate.

Practitioners were informed that the results could be published and that the survey was anonymous. It was also indicated that anonymity will be preserved even if data will be published.

7.2 Survey Results and Analysis

7.2.1 Software Architecture and Software Architecture Evaluation Experience

For the analysis, I created two categories for the self-perceived experience. The categories are:

- Total low or no experience responses: including no experience, and low experience level responses.
- Total average to high experience responses: including average, moderate, and high experience level responses.

Both categories were used to describe data from self-perceived experience in software architecture and software architecture evaluation.

In relation to the self-perceived experience in software architecture, a total of 7 respondents answered low or no architecture experience (see Figure 7.1) and a total of 26 respondents said they self-perceived between average and high experience.

When asked to rate their experience in software architecture evaluation, a total of 10 respondents answered low or no architecture evaluation experience (see Figure 7.2) and a total of 23 respondents said they self-perceived as having average to high experience in architecture evaluation.

I believe these results account for two expected issues because I have observed it in the industry: architecture evaluation is seen as part of a more mature software architecture practice, that is, practitioners recognize architecture evaluation as a key part of architecture design, but in general the evaluation of the designed architecture is often seen as a practice that requires more experience in architecture design.

7.2.2 Relative Importance of Alphas

When asked for rating the relative importance of the proposed Alphas and their progression paths, respondents agree that Alpha Business Goals deserves the first place in 85% of the responses (see Figure 7.3). Following "Business Goals", an almost 60% of the respondents

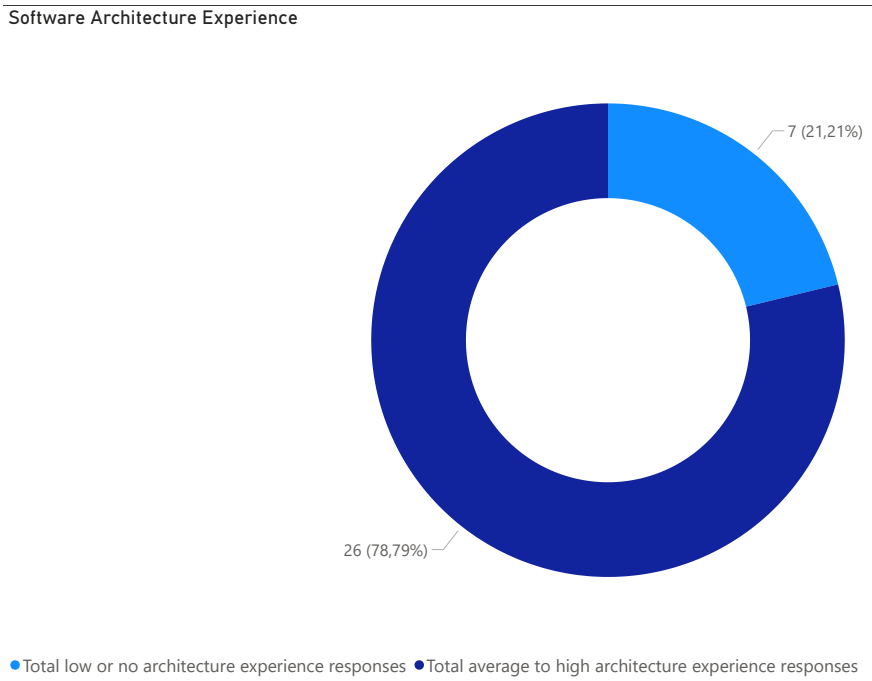


Figure 7.1: Software architecture self-perceived experience.

agree that Alpha Quality Attributes deserves a first or second place in the relative importance. Given that many quality attributes are driven by the business goals, this result is consistent with the current software architecture body of knowledge.

Alphas Architecture Description and Architecture Decisions follow next in relative importance. Alpha Architecture Description Alpha is rated as first or second place in relative importance in a 36% of the responses, while Alpha Architecture Decisions is rated as first or second place in relative importance in a 13% of the responses and first or second or third place in an almost 40% of the responses.

Alpha Architecture Evaluation Adoption is rated as the last place of relative importance in 60% of the responses and slightly over 80% of the times in the fourth and fifth place of relative importance. This Alpha relates to the progression of a methodologically correct architecture evaluation. I believe this result shows that practitioners do not see much importance to evaluation methods or approaches when compared to the other "essentials" of a software architecture evaluation. This is understandable: if one needs to run an architecture evaluation, it might sound obvious that finishing the architecture evaluation is the primary

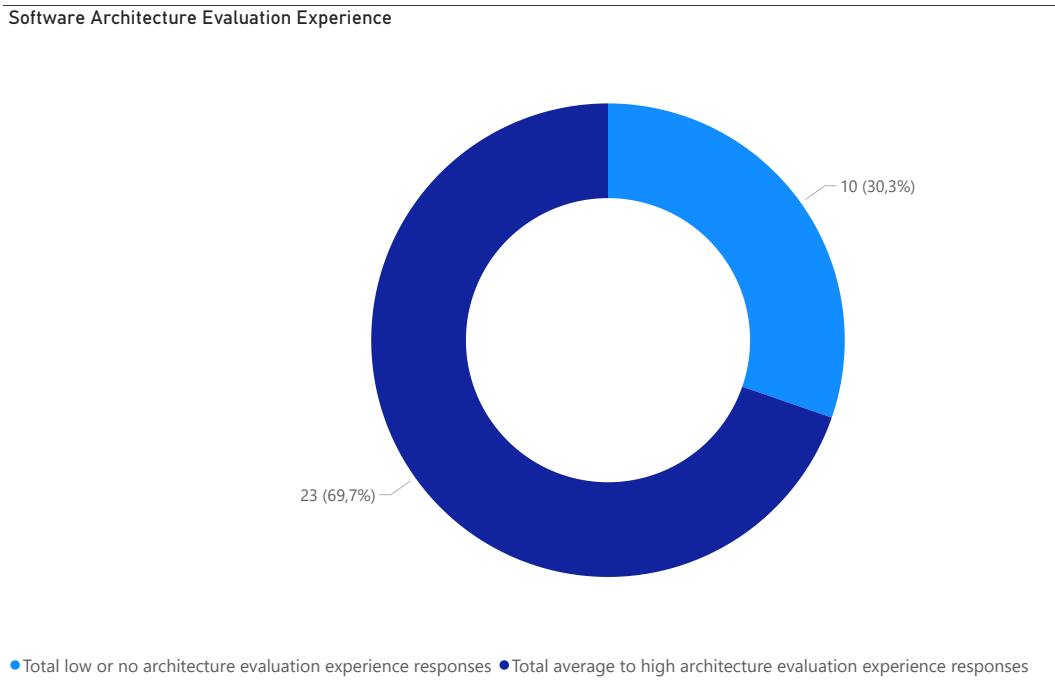


Figure 7.2: Software architecture evaluation self-perceived experience.

goal, and any attempt to improve the adoption of the architecture evaluation process is a secondary goal.

When disaggregated by experience, a 100% of the responses from respondents that self-perceived with low or no experience in architecture evaluation rate in fourth and fifth place of relative importance the Alpha Architecture Evaluation Adoption (see Figure 7.4). While still having lower relative importance, respondents that self-perceived as average to highly experienced in architecture evaluation gave more relative importance to the Alpha Architecture Evaluation Adoption with an approximate of 25% (see Figure 7.5) of the responses indicating a second or third place of importance to this Alpha. This result comes as a surprise because I initially believed that less experienced people would see more benefits in having concrete guidance in how to adopt the architecture evaluation process. At the same time, this result could show that more experienced architects are the ones that know the real problems that arise when trying to adopt the practice, and thus they see the Alpha Evaluation Adoption slightly more important.

The Alpha Business Goals consistently gets the first place in relative importance in both

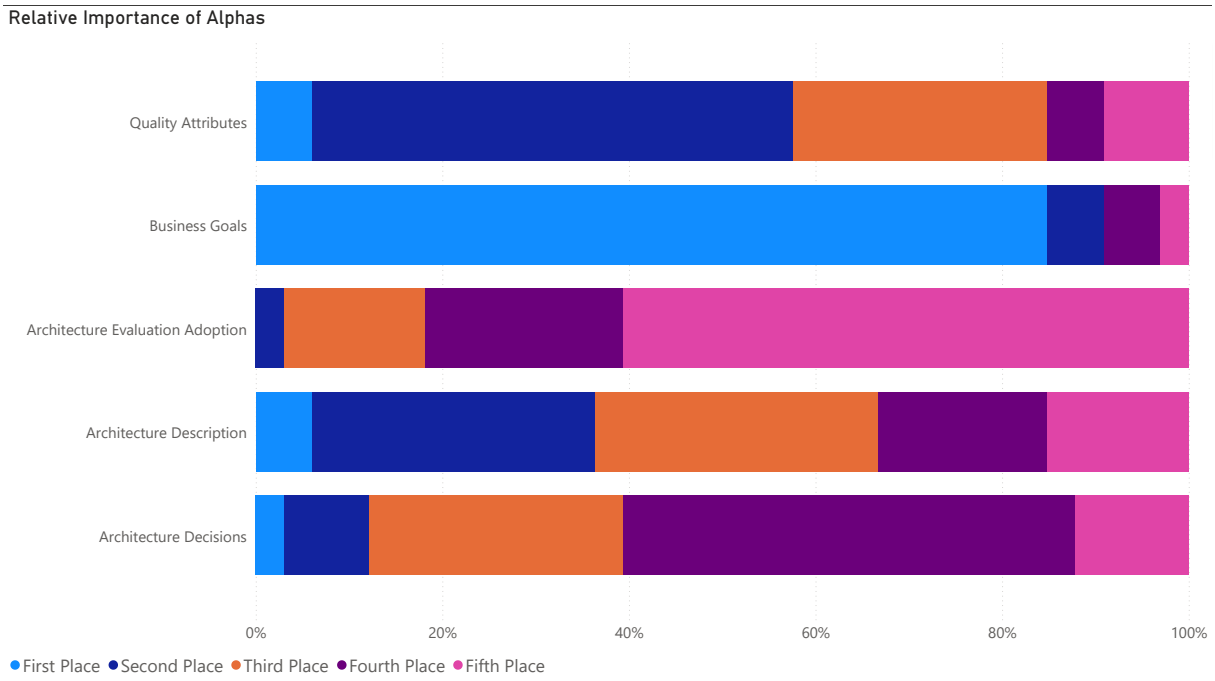


Figure 7.3: Relative importance of proposed Alphas (all responses).

categories of experiences (see Figures 7.4 and 7.5). Nevertheless, respondents that self-perceived as average to highly experienced in architecture evaluation rate this Alpha as first place of relative importance in the 90% of the responses, while in the case of respondents that self-perceive with low or no experience in architecture evaluation rate the Alpha between second and fourth place of relative importance in a 30% of the responses, thus lowering, although not quite substantially, the relative importance to Alpha Business Goals. My interpretation is that more experienced architects are closer to strategic levels and thus they are more aware of the importance of the business goals in architecture design. In addition, one respondent in the survey wrote in the open text field: "if the architecture is not aligned with business goals, we simply discard it". Although this is probably a very specific situation, the importance of business goals is confirmed.

In relation to Alpha Quality Attributes, respondents that self-perceive as average to highly experienced in architecture evaluation believe that this Alpha deserves a first and second place of relative importance in a slightly over 60% of the responses. In turn, respondents with low or no experience in architecture evaluation rate the same Alpha in 80% of the times

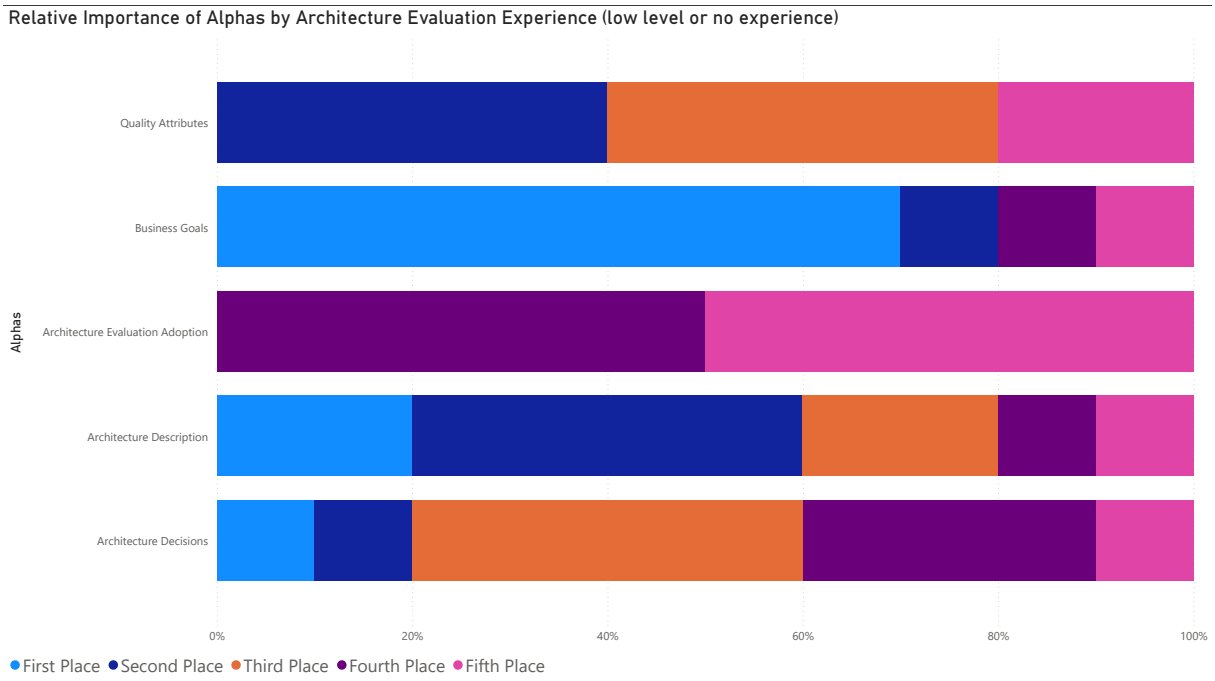


Figure 7.4: Relative importance of proposed Alphas (only low or no experience in software architecture evaluation).

between second and third place of importance. Quality attributes are typically analyzed when articulating business goals. Therefore, I believe this result is consistent with the relative importance that respondents gave in each experience level class to the Alpha Business Goals.

For the Alpha Architecture Description, respondents self-perceived as average to highly experienced in architecture evaluation rate this element between the second and third place of relative importance with a slightly over 60% of the responses. Respondents with low or no experience in architecture evaluation rate the Alpha between first and second place of relative importance in a 60% of the responses. I believe that less experienced practitioners tend to rely more in architecture descriptions than more experienced practitioners. This does not mean that experienced developers make less use of architecture descriptions. Given that Alphas not only represent essential elements, but also represent their progression to a "good" state for a healthy architecture evaluation, I interpret this result as that less experienced practitioners see more importance in following a path to good architecture descriptions while experienced practitioners see less importance in this progression path (e.g., perhaps they just make use

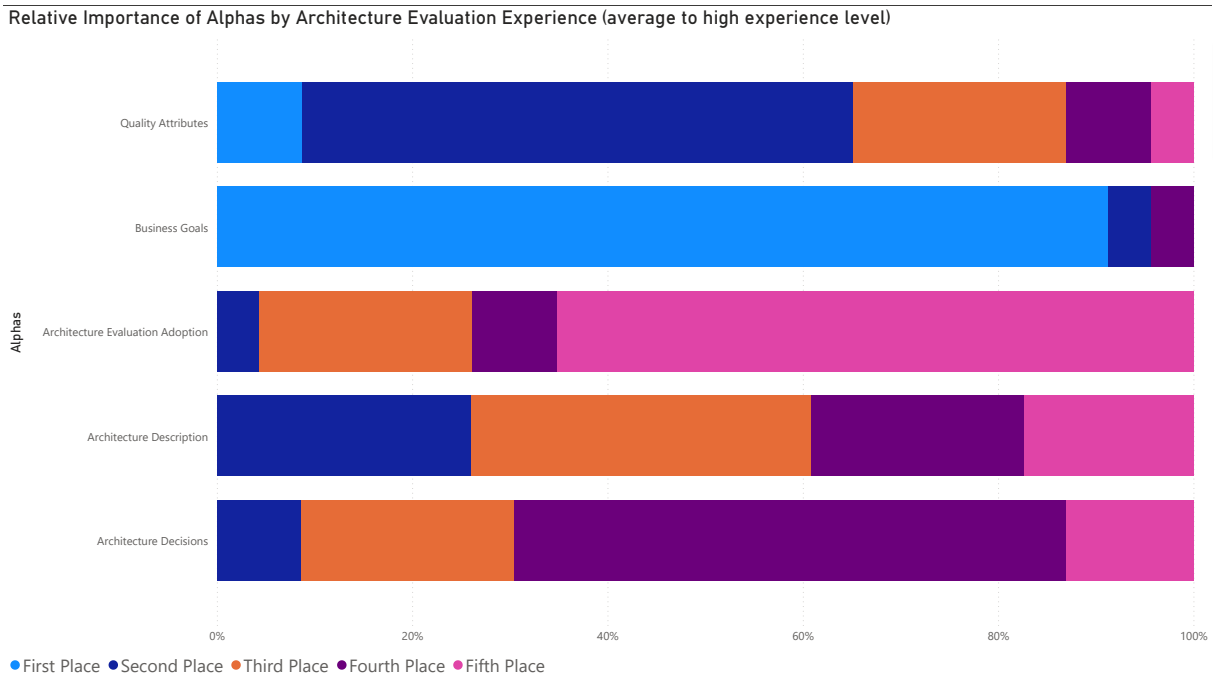


Figure 7.5: Relative importance of proposed Alphas (average to high experience in software architecture evaluation).

of whiteboard diagrams and that is just what they need for the purpose).

The Alpha Architecture Decisions is generally rated as less important than Alpha Architecture Description by both experience groups. When disaggregated by experience groups, respondents with low or no experience in architecture evaluation rate between first and third place this Alpha in 60% of the times. Average to highly experienced in architecture evaluation respondents rate this Alpha between second and third place in 30% of the times. No average to highly experienced respondents rated this Alpha in first place. As this Alpha mainly recognizes the progression from tacit architecture knowledge to explicit and recorded knowledge, I believe that less experienced practitioners see a benefit in following a path towards more formalized architecture knowledge in a less known practice enactment, while more experienced practitioners could eventually still evaluate an architecture without the need of relying in more formalized knowledge (which at the same time could risk the correctness of the evaluation results).

This relative importance by no means should be interpreted as a potential ordering of the

Alphas when guiding a software architecture evaluation. Indeed, in one of the cases presented in Chapter 6, the evaluation started with Alpha Evaluation Adoption, which is put in last place of relative importance in 60% of the responses by both experience level groups.

7.2.3 Helpfulness of the Proposal in Architecture Evaluations

In the second section of the survey, practitioners were asked to rate in a traditional five-level Likert scale the perceived helpfulness for the proposal in thirteen aspects (see Section 7.1.1). The five levels are: strongly disagree, disagree, neutral, agree, strongly agree.

In general, it can be observed (see Figure 7.6) that respondents agree and strongly agree in almost 90% of the times that the proposal is helpful for setting in order, and maintaining ordering in the process of a software architecture evaluation. At the same time, around 50% of the respondents answered that there is no clear aid from this proposal to maintaining costs bounded. In addition, if I add to this number the disagreement votes (disagree and strongly disagree), over 60% of the respondents do not find any clear aspect of this proposal to maintain costs bounded.

Slightly over 70% of the respondents agree and strongly agree that the proposal is helpful for sharing knowledge about how a software architecture evaluation is run. This result is very consistent with what I observed in all cases in which I have used this proposal (see Chapter 6).

Other interesting figures are observed in avoid work duplication and assessing when a software architecture evaluation is in problems. Almost 90% of the respondents agree and strongly agree that the proposal is helpful for avoiding work duplication, and around 70% of the respondents agree and strongly agree that the proposal aids in assessing when an architecture evaluation is in problems.

A similar figure is observed in being more objective and in solving in time problems that emerge in a complex architecture evaluation.

If I analyze by experience level, interesting issues can be observed in the helpfulness of the proposal in relation to solving problems in time that emerge in complex architecture evaluation. For respondents with low or no experience (see Figure 7.7) in architecture evaluation, around 50% of the respondents agree or strongly agree that the proposal aids in this aspect. In turn, respondents that self-perceive between average and highly experienced in

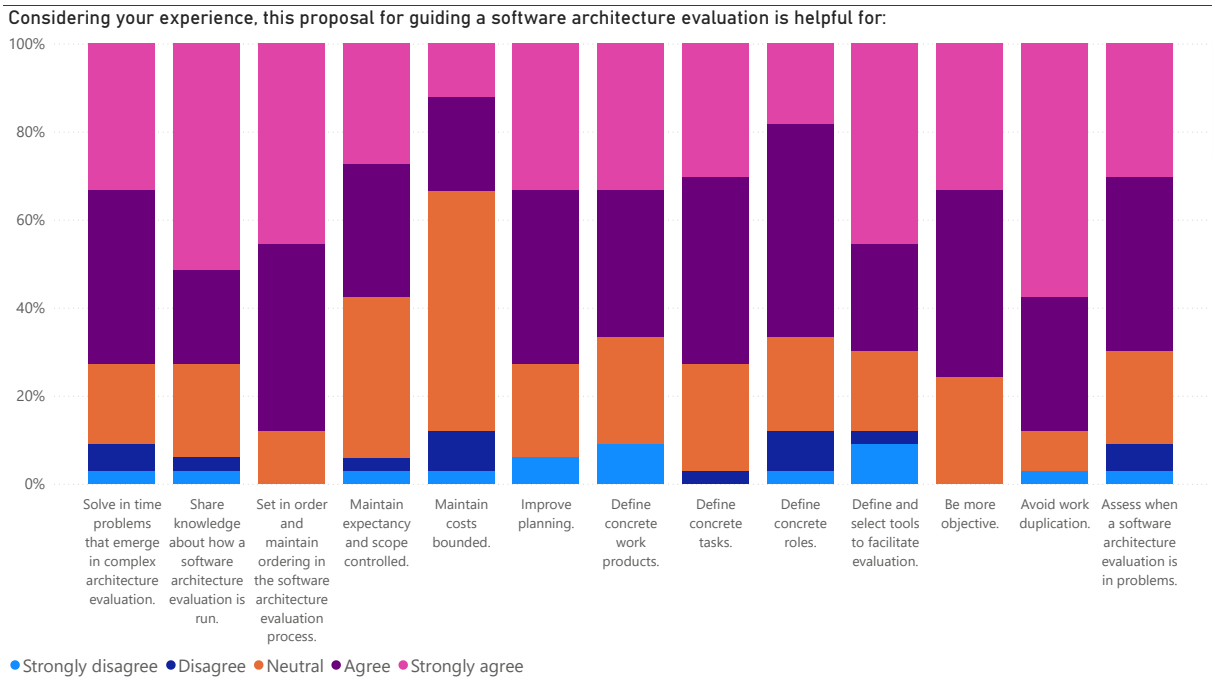


Figure 7.6: Proposal's helpfulness by aspect, including all practitioners.

architecture evaluation agree or strongly agree that the proposal aids in this aspect in an approximate 80% of the times (see Figure 7.8). My guess in this case is that less experienced architects struggle to distinguish if problems are due to the evaluation itself or are more contextual. In my opinion, more experienced architects typically have more tactics and tools to face contextual problems and the proposal becomes helpful for challenges that are specific to the evaluation.

The next interesting issue appears in the aspect of maintaining costs bounded. In the case of the less experienced architects, around 70% of respondents are neutral when asked if the proposal is helpful for maintaining costs bounded. This percentage goes down to around 50% for more experienced architects while at the same time the agree or strongly agree responses go up to around 40% for this group (compared to the slightly less than 20% shown by less experienced architects). In my opinion, this difference is due to that experienced architects often have a clearer understanding of the trace between something that moves in the software process level and financial aspects. Nevertheless, maintaining costs within bounds is still one of the aspects where the majority of the respondents do not see a clear connection with the

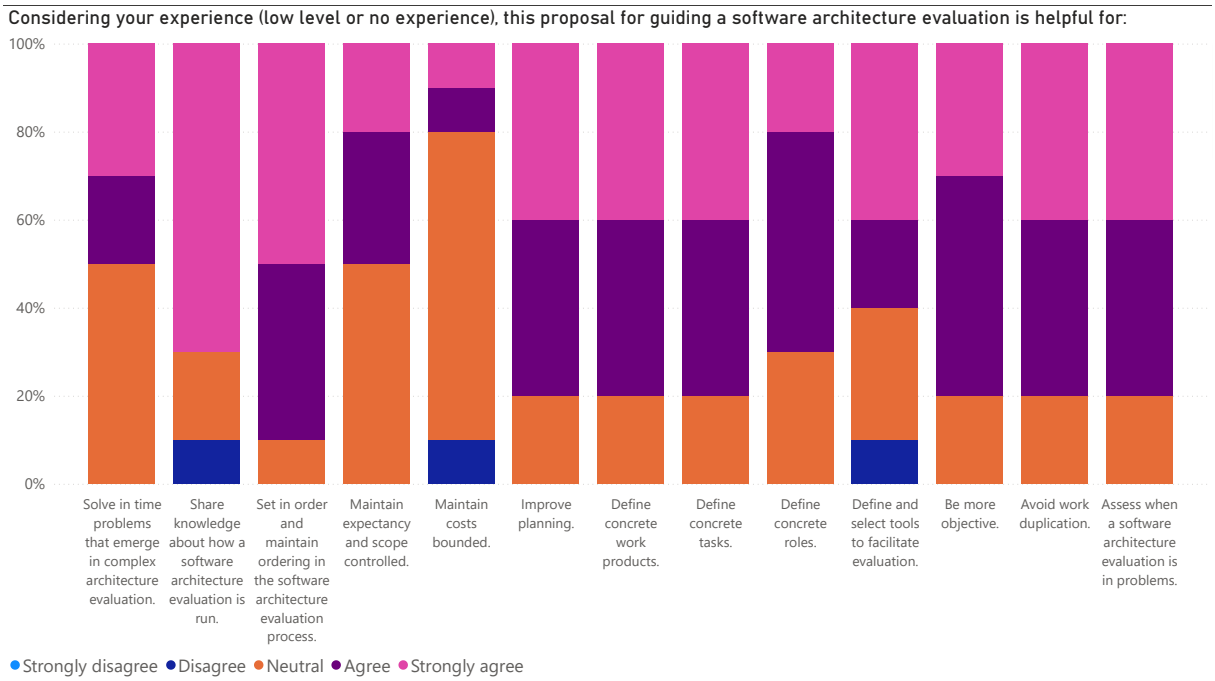


Figure 7.7: Helpfulness of the proposal (low or no experience in software architecture evaluation).

proposal.

The other aspects do not show very observable differences between the less experienced and more experienced architects.

7.3 Difference in Helpfulness by Experience: Statistical Analysis

The set of questions that asked practitioners to rate the helpfulness of the proposal in regard of thirteen aspects can be studied in relation to two groups of practitioners: less experienced and experienced. For the less experienced practitioners group I consider people with low or no experience in architecture evaluation. For the experienced group, I consider people with average to high experience.

To run a statistical analysis for the differences in responses, it is worth noting the following:

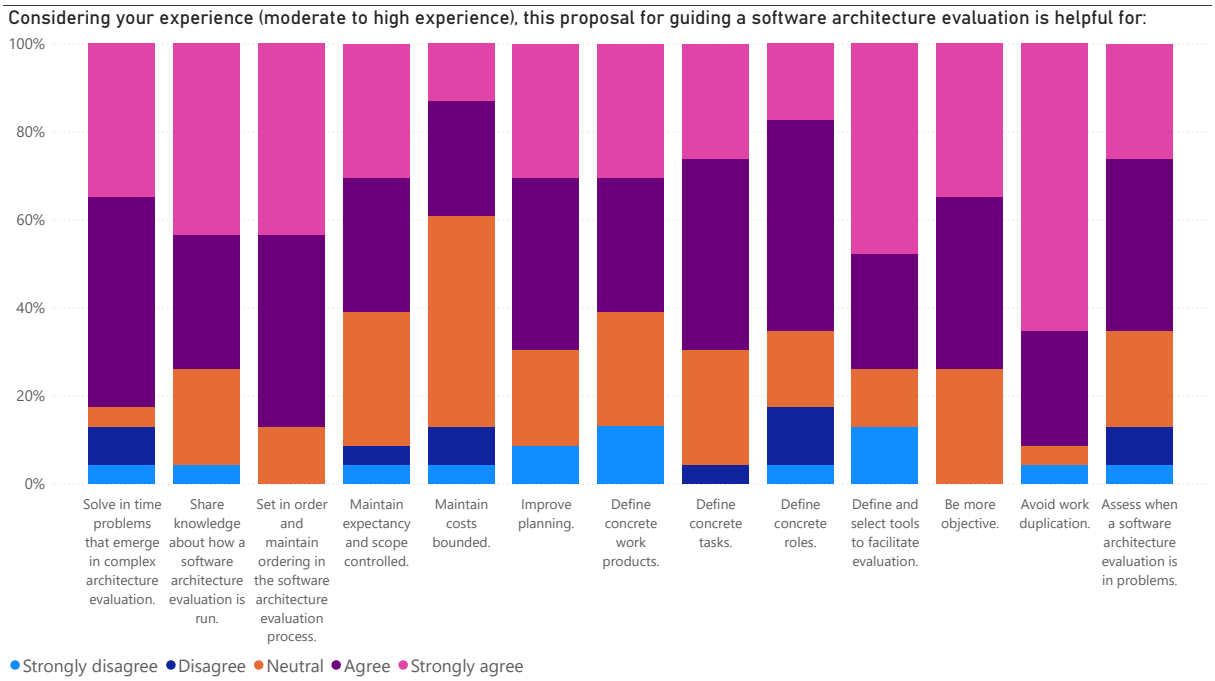


Figure 7.8: Helpfulness of the proposal (average to high experience in software architecture evaluation).

- The scale of the responses is categorical (from strongly disagree to strongly agree; 5-Likert) and ordinal; the responses are of type "ratings & feedback". Therefore, the responses can be ranked.

- The distance between the categories of responses is arguably even (e.g., the distance between strongly disagree and disagree is arguably very similar to the distance between disagree and natural).

- In the survey, practitioners were allowed to run the survey just once. They were not allowed to re-answer the survey nor the survey was run in two different points in time for the same practitioner. This ensures that the groups (no experience vs experienced) are independent.

- Practitioners that answered the survey did not share knowledge about the survey. Moreover, they came from different settings. Therefore, the answers can be assumed

as independent.

Given these characteristics, a rank-comparison statistical test such as the Mann-Whitney U test (also called the Wilcoxon rank-sum test) can be run. In this test, the hypotheses are the following:

- Null hypothesis (H_o): There is no difference of central tendency between the two groups in the population.
- Alternative hypothesis (H_1): There is a difference of central tendency between the two groups in the population.

The results are summarized in Table 7.1. These figures show that in none of the thirteen questions the null hypothesis can be rejected, meaning that there is no statistical support to claim that the proposal is more useful for non-experienced software architects. Nevertheless, these figures should be understood under the following aspects:

- The sample size is small. The small sample size is commonly reported to be an important factor in failing to reject the null hypothesis. Unfortunately, online surveys tend to have low response rates [116].
- A non-rejection of the null hypothesis cannot be analyzed in a binary fashion (yes vs. no) [117]. It is worth analyzing how far the p-value was from rejecting the null hypothesis in each case.

The Python code for this analysis is available in Appendix E and in [31] where I also made available an anonymized version of the raw data.

7.3.1 Analysis of the Differences Between the two Groups

As noted previously, the difference between the two groups can be analyzed to avoid a simple hypothesis rejection/no-rejection analysis. For comparison, the group of less experienced practitioners is called Group 1 and the other is called Group 2. Table 7.2 presents:

- The question related to the results that are being compared.

Table 7.1: Mann-Whitney U test statistic and p-value for each question.

Question 1: Set in order and maintain ordering in the software architecture evaluation process. Null Hypothesis not rejected: Statistics=123.50, p=0.73.
Question 2: Avoid work duplication. Null Hypothesis not rejected: Statistics=85.00, p=0.19.
Question 3: Maintain expectancy and scope controlled. Null Hypothesis not rejected: Statistics=104.00, p=0.67.
Question 4: Maintain costs bounded. Null Hypothesis not rejected: Statistics=100.00, p=0.53.
Question 5: Assess when a software architecture evaluation is in problems. Null Hypothesis not rejected: Statistics=141.00, p=0.29.
Question 6: Be more objective. Null Hypothesis not rejected: Statistics=115.50, p=1.00
Question 7: Improve planning. Null Hypothesis not rejected: Statistics=133.00, p=0.47.
Question 8: Solve in time problems that emerge in complex architecture evaluation. Null Hypothesis not rejected: Statistics=93.50, p=0.38.
Question 9: Define concrete tasks. Null Hypothesis not rejected: Statistics=136.00, p=0.39.
Question 10: Define concrete roles. Null Hypothesis not rejected: Statistics=127.50, p=0.61.
Question 11: Define concrete work products. Null Hypothesis not rejected: Statistics=140.00, p=0.31.
Question 12: Share knowledge about how a software architecture evaluation is run. Null Hypothesis not rejected: Statistics=134.00, p=0.43.
Question 13: Define and select tools to facilitate evaluation. Null Hypothesis not rejected: Statistics=104.50, p=0.68.

- The p-value, already presented in Table 7.1. The table is ordered in ascending order by this column (i.e., at the bottom it is the highest p-value).

- The average of the responses using ranks 1 to 5 (strongly disagree = 1, strongly agree = 1) for Group 1 (less experienced practitioners).

- The average of the responses using ranks 1 to 5 (strongly disagree = 1, strongly agree = 1) for Group 2 (experienced practitioners).

- The difference between the two averages.

The figures in Table 7.1 show that "avoid work duplication" exhibits the lowest p-value, while "be more objective" exhibits the highest p-value. The interpretation here is that, at least with the available data, the "avoid work duplication" is the aspect that was closer to exhibit statistical support for the difference. Analyzing this with the groups average, I intuit that avoiding work duplication is a potential benefit observed by both groups. I would have expected that less experienced practitioners felt "more guided" by the proposal than the experienced ones. However, the data do not support this. At the same time, this is consistent with the claims of this thesis that this proposal try to alleviate the operational aspect of evaluating software architectures.

The following two aspects, "assess when a software architecture evaluation is in problems" and "define concrete work products" show similar p-values, and the average of the perceived helpfulness is higher for less experienced practitioners than experienced practitioners. I interpret this result as consistent with my initial thoughts because I have observed that experienced practitioners often face those issues with an "ad-hoc" manner, mostly due to their experience in previous cases. A similar argument for why the perceived helpfulness of the proposal for defining concrete work products ("define concrete work products") is higher for less experienced: experienced practitioners tend to define work products in more ad-hoc ways, in this case based on their experiences, but also aligned with their own organizational context (which raises the question: are architecture evaluation work products really mandated by the architecture evaluation method or approach?).

The following aspects exhibit higher p-values and I argue that for this analysis these cases are too far from eventually rejecting the null hypothesis in favor of the alternative. In other

words, in such cases the probability that these differences occurred by random circumstances is too high for being analyzed in this section.

7.4 Chapter Remarks

In a compliment to the case studies, the proposal was presented to practitioners related to the software architecture domain and they were asked to answer a survey. The responses show that Alphas Business Goals and Quality Attributes are seen as the most important for evaluating a software architecture. Alpha Evaluation Adoption takes the last place in the relative importance.

The results also allow to conclude that solving problems, sharing knowledge about how to run an architecture evaluation, improve planning, define concrete work products, tasks, and roles, and being more objective are among the aspects for which the proposal is seen more helpful. At the same time, maintaining costs bounded is the aspect where there is not a clear understanding of how the proposal could help.

Statistical inference for analyzing the difference between the less experienced and more experienced groups of practitioners show that there is no clear evidence that responses differ between the groups. However, this can be influenced by the low number of responses and, when analyzing the p-values, it can be observed that avoiding work duplication, assessing when a software architecture evaluation is in problems, and defining concrete work products are the aspects closer to a p-value that would allow to support that there is a difference between the perceived helpfulness for less experienced and more experienced groups.

More study of this phenomenon is expected and is left for future work.

Table 7.2: Summary of the differences in central tendency for the answers of the two groups.

Question	p-value	Group 1 Average	Group 2 Average	Difference
Avoid work duplication.	0.19	4.2	4.5	0.3
Assess when a software architecture evaluation is in problems.	0.29	4.2	3.7	0.5
Define concrete work products.	0.31	4.2	3.7	0.5
Solve in time problems that emerge in complex architecture evaluation.	0.38	3.8	4.0	0.2
Define concrete tasks.	0.39	4.2	3.9	0.3
Share knowledge about how a software architecture evaluation is run.	0.43	4.3	4.1	0.2
Improve planning.	0.46	4.2	3.8	0.4
Maintain costs bounded.	0.53	3.2	3.3	0.1
Define concrete roles.	0.61	3.9	3.6	0.3
Maintain expectancy and scope controlled.	0.67	3.7	3.8	0.1
Define and select tools to facilitate evaluation.	0.68	3.9	4.0	0.1
Set in order and maintain ordering in the software architecture evaluation process.	0.73	4.4	4.3	0.1
Be more objective.	1.00	4.1	4.1	0

Chapter 8

Threats to Validity

The validity in research concerns with the truth of the claims derived from the empirical experience [118], including experimental and non-experimental ones (e.g., case studies and surveys).

A threat to validity is any identifiable issue that puts the claims from an empirical experience at risk [32][118]. Researchers in both quantitative [118] and qualitative [119] domains agree that if validity is an issue in the empirical experience, two aspects are key: (1) there is no objective truth; therefore, validity analysis is mostly qualitative, and (2) it is neither possible nor practical to rule out every validity threat.

Researchers typically need to take into account threats to the following validity types: conclusion validity, internal validity, construct validity, and external validity [32][118]. Some of these validity types are adapted for qualitative research [119]. For example, conclusion validity analysis tends to be more quantitative in experimental designs [118], while in non-experimental designs such as case studies, researchers take a more qualitative approach [32].

Validity is a property of the knowledge claims rather than the methods [118] and therefore the threats need to be taken into account in relation to the specific empirical experience [32]. This implies that in the design of an empirical experience, a tradeoff between these validity types is expected, and in the majority of the cases some threats are simply accepted [118].

In the following sections, I discuss each validity type in the context of the two case studies and the survey presented in this thesis. In each section, I present and discuss the threats that I consider important for this work.

8.1 Conclusion Validity (Reliability)

Conclusion validity concerns with issues that affect the claims that are observed in the empirical experience [118]. In case studies, this type of validity is closely related to "reliability" [32]. Reliability concerns with the extent to which the results observed in the case depend on the empirical experience and not on the researcher. In other words, reliability in a case study calls for ensuring that if another researcher replicates the case study, the results are consistent.

In this study, the main concerns with conclusion validity are extraneous factors affecting the case (e.g., meetings disruptions, disagreements between stakeholders, among others) and overestimation (also called researcher bias [119]), that is, when the researchers value the observed results (positive or negative) too highly.

The first concern is about external events that disrupt the architecture reviews. For example, disruptions in evaluation meetings and disagreements between stakeholders. Such external events can make stakeholders take a pessimistic stance about the evaluation experience. Disruptions in evaluation meetings were not frequent in both cases. However, in the case of the architecture evaluation in the military institution there was a moment in which the disagreement between stakeholders became a challenge because evaluation participants disagreed about how some evaluation activities were being conducted. The proposal of this thesis (i.e., the Alphas) was valuable for regaining attention in the evaluation effort.

The second concern is about the risk of overestimating the observations. Overestimation is also called researcher bias [119] and can occur when the researcher values the positive or negative results too highly. In the presented case studies, overestimation can emerge in two issues: first, the natural desire from the researcher to validate his/her beliefs (belief and confirmation biases that I had even before starting the cases) and second, the inclusion of the researcher in the cases which affect the way the activities are carried out (in the cases, I was involved in both evaluations). The way to mitigate these threats in this work is by following published and accepted guidelines for designing and running case studies, and by insisting in following the proposed approach rather than relying on ad-hoc activities. Undoubtedly, this threat is hard, if not impossible to mitigate to zero.

Conclusion validity threats can also be observed in the survey. The most important concern is that the survey results are interpreted and the interpretation is undoubtedly influenced or biased by the researcher and its experience. One way to alleviate this concern

is to ask for an explanation and argumentation in each answer. However, this clearly could easily double the effort required for answering the survey, which would have worsened the response rate. In this work, I decided to resolve this tradeoff in favor of the response rate.

8.2 Internal Validity

Internal validity concerns with the degree that the claims come from the empirical experience. In other words, the degree to which the observation that the use of this proposal serves its purpose to guide and assess the progression of an architecture evaluation effort is true. Threats to this kind of validity typically are influences that the researcher will never know [32].

The lack of experimental control which is one of the key reasons for designing a case study instead of other empirical methods [32] make internal validity one of the most compromised types in case studies [118].

The most important concerns about internal validity in this work are the selection of subjects and the history in the evaluation case (history refers to stakeholders actively or passively learning from the case [118]).

The first concern emerges from the fact that both cases occurred in the context of real-world settings and products, where the selection of subjects was not feasible. Instead, the selection of people was a natural consequence of the stakeholders involved in the engineering of both systems. Fortunately, part of this threat was self-alleviated since all participants in both case studies had not been previously trained in the proposal for architecture evaluation.

The second concern is about the motivation for learning about software architecture evaluation while the experience is running. Depending on the organizational setting and the product, an architecture evaluation could last several days or even weeks. While learning about architecture evaluation is beneficial for stakeholders, it can hinder internal validity. The question that emerges here is whether the observable effects of using this proposal are mostly due to the proposal itself or if there is a hidden influence from stakeholders progressively learning about architecture evaluation. The approach taken to minimize this validity threat was to keep the "offline" times (i.e., the times between the evaluation activities) as reduced as possible.

Internal validity can affect a survey in the sense that practitioners could have been distracted or, even worse, practitioners did not really answer the survey (perhaps, random

responses were given, or someone else answered the survey). Unfortunately, the only countermeasure to this threat is to analyze meta-aspects such as the completeness of the responses (e.g., if some responses are missing, the survey should be discarded unless a practitioner provides a compelling reason in the open text field available for thoughts and comments).

8.3 Construct Validity

Construct validity concerns with the consistency between theory and observation [32] in the empirical experience. The threats to this validity type are important to take into account in empirical research to assure that the knowledge claims from the empirical experience are derived from the application of a correctly defined technique [118].

The most important concerns about construct validity in this work are overly optimistic expectations from an architecture evaluation, learning enthusiasm, and evolution of the proposal.

The first concern is observed when the architecture evaluation practice is presented to stakeholders. The expectations can be overly optimistic. For example, I have observed in various evaluation experiences that stakeholders would like to use an architecture evaluation for the sole purpose of identifying architectural technical debt. This poses a risk in the consistency between the architecture evaluation constructs and the observation and knowledge claims from the empirical experience. The problem arise when accepting that an architecture evaluation can be used to identify architectural technical debt, which is not an intended use for architecture evaluations. Construct validity threats such as this example might arise from technical opinions and intuition, or even from political and economic sources [118].

The second concern is about the participants to be very eager to learn about the architecture evaluation practice. This enthusiasm is a known threat to construct validity [32] as it might make these participants prone to bias their understanding of the practice to their own expectations.

The third concern emerges from the evolution of the proposal. I am also learning from the case studies and thus the proposal was slightly modified to adjust for improvements. While this might eventually pose a risk to construct validity (the construct is evolving), I limited the adjustments to improvements in the descriptions of the Alphas. No new Alphas were added between the cases (which would have been considered as a big change in the construct

represented by the proposal).

In addition, a solid foundation on software architecture evaluation from both literature research and reflective practice, and the use of the SEMAT Kernel as a thinking and modeling framework are the two strategies that I used in this work to alleviate the threats to this validity type. The proposal of this work is in itself a guiding artifact that also alleviates construct validity threats as it deter stakeholders to take other paths in the evaluation effort.

Construct validity also affects survey design. The main concerns of survey design are whether the questions are truly asking what the researcher wants to elicit and if the questions are sufficiently well designed to avoid obliging respondents to select specific responses. To counteract these concerns, the use of recommendations for survey design (see [64, 63, 65]) is the strategy used in this work.

8.4 External Validity

External validity concerns with the extent to which the claims from the empirical experience can be generalized [32] over variations in people, settings, method or treatment, and outcomes [118]. This type of validity is particularly important in this work as case studies typically expect the results to be applicable in real-world settings that differ from the settings in which the case was studied. Most, if not all, knowledge claims (i.e., the effect) derived from the application of some practice (i.e., the cause) depend on the setting to some degree [118].

The most important concerns about external validity are the interaction of the claims from the study with the people involved in the case, the interaction of the claims with the setting, and the fact that both product evaluated were legacy systems.

The first concern about external validity is about the interaction of the claims from the study with the people involved in the case. The threat in this case is the risk that more experienced people could eventually prefer and use more ad-hoc approaches because it is the way in which they have evaluated architectures and know the many "perils and pitfalls" of such enactments.

The second concern about external validity emerges in the interaction of the claims from the study with the settings. In both cases, a more classic style of development (i.e., more-waterfall approach) was observed. The question here is whether the claims from the experience still hold in more agile environments. I also consider in this concern that both software

architectures were part of legacy systems and not new developments. A natural question here is whether the claims from the experience will be similar in a setting in which a new software system is being developed.

Much of the external validity concerns typically motivate further research [118]. In particular, the application of this proposal in a more agile setting motivates me to continue further research (see Chapter 9).

This type of validity is especially important in the survey of this work. The generalization of the results is highly dependent on the context in which the results are expected to be generalized. For example, there can be some places where architecture evaluation is a mature and very precisely defined process. In such cases, I'm not reasonably sure that the results from the survey will still hold. In this type of validity threat, the main concern is to make practitioners and researchers aware of the potential problems when generalizing these results.

8.5 Chapter Remarks

Validity threats are issues that pose one or more risks to the claims made in a research work. In this chapter, I discuss the main threats to validity regarding the typical four kinds: conclusion validity, internal validity, construct validity, and external validity.

The main concerns about sustaining the validity of the claims of this work are discussed in relation to these four types of threats. The case studies and the survey are the main subjects of discussion in this chapter.

Chapter 9

Conclusions

9.1 Thesis Conclusions

Software architecture evaluation is a key software quality practice, yet it is not straightforward and is often done in informal, ad-hoc ways. To contribute to facilitating the running of such architecture reviews, this thesis identifies (1) the essential elements that practitioners need to consider when evaluating a software architecture and (2) the discrete states that represent the evolution of each of these elements in an architecture evaluation. This work presents a SEMAT-Kernel-based representation of the software architecture evaluation practice as a set of Alphas and States that represent the essential elements of the software architecture evaluation practice and the discernible progression stages of each element, respectively. The decision of using the SEMAT Kernel as a modeling framework relies upon the SEMAT Kernel extensibility as well as that it provides a concrete form to describe practices in terms of "essential elements" that express both the characteristics of each element of the practice and their progression paths. The Alphas constitute a concrete mean that can eventually be used by practitioners and researchers for understanding, guiding, and assessing and auditing the progression of architecture evaluations.

In addition to the modeling characteristics that are relevant to this study, the SEMAT Kernel was also chosen because of its actionable approach. I have extensive experience using the original Alphas from the SEMAT Kernel to assess the progression of various software engineering efforts. Its actionable approach enables development teams to concretely de-

termine the current position in the progression of a software engineering effort and which specific objectives the team needs to achieve to continue progressing the endeavor.

This thesis focused on the following research questions: (1) what are the essential elements that need to be considered when evaluating software architectures, and (2) what are the discernible progression stages of these essential elements that can be used to guide and assess the progression of the evaluation of software architectures. After several software architecture evaluations in which I adopted a reflection-in-action research approach, as well as by reviewing the current software architecture evaluation literature and conducting semi-structured interviews with stakeholders, the following Alphas were determined as the essential elements of the software architecture evaluation practice: Business Goals, Quality Attributes, Architecture Description, Architecture Decisions, and Evaluation Adoption. As indicated in the SEMAT Kernel, an Alpha comprises a set of States, with each State representing a discernible status in the progression of a software practice. Therefore, the States in the Alphas of this proposal represent the progression paths through discernible status in an architecture evaluation endeavor.

I conducted two additional architecture evaluations for real-world software products, using a case study approach aimed at evaluating in practice the suitability of the proposed Alphas for guiding and assessing the progression of architecture evaluations. In both cases, the software products are considered legacy and under a more classic, although not exactly waterfall, development style.

It was observed that the proposed mechanism allowed in such contexts to focus on the key aspects of an architecture evaluation and maintain an actionable guide for the progression of an architecture evaluation. Given that the progression paths are described as expected items, each State in each Alpha can be used as check points for auditing an architecture evaluation. Key observed insights also include: it was possible to run a software architecture evaluation using only the Alphas (without relying on an evaluation method), as well as using the Alphas complementing an evaluation method (which I regard as a good indicator that the proposal is consistent with how a well-known evaluation method organize and implement such endeavor); the use of the Alphas helped involved stakeholders to regain focus in the evaluation activities when the work became tangled in a stakeholder-massive ATAM-based review; a more autonomous architecture evaluation was observed because of the actionable guidance that the progression path of each Alpha provides; finally, a more consistent structure

was also observed for the evaluation reports (both documents and presentations).

I remark that these observations were made in two contexts in which teams are well formed and mature and the development approach is considered as "classic" (although not exactly a "waterfall" approach), rather than agile. I believe "classic" environments differ from agile contexts enough to consider studying this proposal in such environments, which is left for future research. At the same time, this belief calls for caution when using this proposal in agile environments, where rapid delivery often interferes with typically slower quality practices such as software architecture evaluation.

In this thesis work, I also conducted a survey with practitioners from a broader variety of domains. The proposal was presented to these practitioners and then I asked them to answer questions regarding the importance of the Alphas and the helpfulness of the proposal in relation to thirteen aspects. Survey results show that practitioners frequently agree that (1) setting and maintaining order in the architecture evaluation process, (2) sharing knowledge about how a software architecture is run, (3) avoiding work duplication, and (4) defining concrete tasks, roles, and work products are the aspects where this proposal is most useful. At the same time, maintaining costs bounded has been cited as the least-facilitated aspect. I have not thoroughly studied this last aspect; however, I believe that this can be the effect of technical people having a low awareness of cost-related issues probably due to a lack of a widely accepted relationship between software quality aspects and financial aspects. Although this is a very interesting topic, I do not consider it a direct future research direction of this work.

The difference between responses considering two groups (less experienced and more experienced software architects) was also analyzed. While I could not find statistical support to argue that the responses between the two groups statistically differ, it is interesting to see that solving problems in a complex architecture evaluation was one of the aspects where groups exhibited more differences in responses: more experienced architects tend to agree more in the helpfulness of the proposal for this aspect. Statistically speaking, avoiding work duplication and assessing when a software architecture evaluation is in problems are the aspects that exhibited the lowest p-values.

The main hypothesis of this work is: *"the progression through the proposed Alphas and States leads to completing, and assessing the progression health of a software architecture evaluation endeavor",* and the specific hypotheses derived from the main hypothesis are

(1) the proposed model leads people with experience in software architecture (including evaluation) to better assess a software architecture evaluation endeavor (2) the proposed model leads people with no prior experience in software architecture (including evaluation) to better understand the progression growth path (i.e., what is required to do) in a software architecture evaluation endeavor. At this point, I can argue that the main hypothesis is supported by the evidence gathered, at least in the context in which the proposal has been tried. The evidence supports the claim that the proposal is suitable for guiding, assessing, and auditing the progression of architecture evaluations, and the survey data also exhibit agreement that the proposal is helpful for defining concrete tasks, work products, and roles — that is, the three essential elements of processes. Moreover, the survey shows strong agreement in evaluation planning improvement. In relation to the specific hypotheses, I can argue that the hypotheses are also supported. For the first specific hypothesis, the survey shows that the more experienced group of software architects frequently agree that the proposal is helpful for assessing that a software architecture evaluation is in problems, being more objective, and solving problems in time. These aspects are related to assessing the health of an architecture evaluation. In addition, the evidence gathered in the case studies also support this hypothesis. For the second specific hypothesis, the survey shows stronger agreement in the helpfulness of the proposal for sharing knowledge about how an architecture evaluation should be run in the less experienced group. There is also agreement from that group in the aspect of evaluation planning improvement, which is also related to the progression path. As with any kind of generalization of results, generalizing these findings to other contexts should be done with caution.

9.2 Limitations of the Proposal

Some limitations of the proposal include: lack of elasticity for the application of the proposal in other software architecture related practices (e.g., identifying architectural technical debt) and a potential unsuitability for guiding small software systems' architectures. In the first case, I tried to apply this mechanism to identify technical debt. Although this experience has not yet been published or included in this thesis, the situated evidence allows me to conclude preliminarily that the proposal is too cumbersome for the sole purpose of identifying architectural debt. In the second case, I recognize that even small software systems have an

architecture. The proposal might be too heavy for evaluating such architectures: in these cases, a more ad-hoc approach would suffice.

9.3 Future Work

As of this writing, I remark four future research directions.

Better understand guidance of software architecture evaluations in agile environments

Both cases presented in this thesis occurred in development contexts exhibiting a more classic development style, although some efforts to adopt more agile approaches were observed. As of this writing, this proposal has been used in more agile environments and with smaller software systems, but its use has not been reflectively observed, as it was done in the two cases presented in this paper.

In relation to this research direction, I would also like to determine if some prescribed or recommended ordering for using the Alphas can be defined and if some benefits could be observed by following a specified ordering. In my experience, it has always been very natural to start with the first State of Alpha "Evaluation Adoption". However, I believe that, in some contexts, this might not be the case. For example, in contexts in which architecture evaluations are prescribed at the organizational level and thus teams do not have much control over deciding which evaluation method or approach is used, the first State of the Alpha "Evaluation Adoption" is considered already reached before the evaluation starts. In a case like this example, the Alpha "Evaluation Adoption" will not be the first priority.

Define concrete essentials for the architectural technical debt identification

In other minor evaluations in which I have used the Alphas from this proposal, I have observed that using them for the sole purpose of identifying architecture technical debt is too heavy. Nevertheless, the observations also tell that essentials such as the Alphas Architecture Description and Business Goals are naturally required for architectural technical debt identification, and thus these Alphas are necessary. Key issues arise from the interdependence

of the Alphas and thus I believe that I can work in another version of this proposal oriented towards identifying architectural technical debt in existing software systems.

Identification and definition of indicators to assess how well an architecture evaluation ends

In most cases where this proposal has been used, the evaluation has concluded when all the Alphas reached the last State. However, in some other minor and less formal architecture evaluations in which I have used the proposal, these reviews can still be considered as finished even if some Alphas have not reached their last State. In such cases, the evaluation meets the expectations while some Alphas still exhibit States to achieve (i.e., pending States). I believe that this proposal can be complemented with a set of indicators that could help practitioners assess the maturity of the practice, assuming that in less mature cases, an architecture evaluation will be considered finished without necessarily reaching the last State of each Alpha.

Explore the suitability of the proposal for teaching and learning about software architecture evaluations

In addition to the use of this proposal for guiding and assessing the progression of architecture evaluations, I have also used the proposed Alphas to (1) guide an architecture evaluation in an undergraduate informatics engineering thesis work and to (2) organize the software architecture evaluation teaching material in a software architecture undergraduate course. Although these experiences have not been treated as case studies, preliminary insights show that the proposal is also beneficial for students to gain concrete understanding and advice about the software architecture practice, and eventually for instructors to concretely define a teaching path for the course.

Appendices

Appendix A

Glossary of Terms

The following definitions are important for this thesis:

- **Method:** An implementation of an operation [120][121] needed to perform a set of cohesive tasks of a process [121]. Given that an operation may comprise other operations [122], in this thesis, a method refers to the set of actions that are required to perform some tasks of a process. An architecture evaluation method defines a set of actions to accomplish an architecture evaluation (for example: running a scenario brainstorming, articulating quality attributes, among other tasks). In SEMAT, a method includes one or more practices, which are described in terms of essential elements called Alphas [27][29].

- **Software Architect:** A role responsible for making decisions [19] about, and maintaining the architecture of a software system. Common tasks associated with this role include the analysis, design, and evaluation of software architectures [13]. The role can be fulfilled by one or more people in an organization.

- **Software Architecture:** The set of principal design decisions [1] that define the structure of the software system [4][78][82]. The architecture can be considered an artifact useful for reasoning about the software and its properties [2].

- **Software Architecture Evaluation:** A software engineering practice, part of the software architecture design [13], used to assess the fitness of a software architecture for the purpose that someone or an organization has with the software system [6].

- **Software Engineering:** The application of scientific and technological knowledge, methods and experience to the design, implementation, testing, and documentation of software [120]. For this thesis, software architecture and its related practices (e.g., software architecture evaluation) are part of software engineering.

- **Software Practice:** A specific approach for achieving something [28] in software engineering, described in terms of essential elements [29] that contribute to the execution of a method [29] or process [123][120].

- **Software System:** A set of software (computer programs, procedures, documentation, and data [120]) and hardware that provide its main value to stakeholders by the execution of the software [28].

Appendix B

The Software Architecture Evaluation Domain

A domain model establishes the concepts and axioms for a subject. For the subject software architecture evaluation a domain model can be devised[1].

Figure B.1 uses a simplified UML class model for representing the domain model of software architecture evaluations. Key aspects from this model are that an **Architecture Representation** is required as it allows reasoning about one or more **Architecture Decision,** a critical in any architecture evaluation.

Another interesting aspect derived from this domain model is that **Architecture Decision** is considered as a "first-class entity." **Architecture Decision** could have been considered as mere attributes of, for instance, **Software Architecture.** Instead, this model recognize them as entities because decisions are not only considered what define a software architecture in modern definitions, but they also are the key element that are worked in any architecture evaluation. An **Architecture Decision** can be said to be "challengeable," that is, it can be presented and challenged by other stakeholders. Considering design decisions as first-class entities also appears in the discussion brought by Bosch in [78].

[1]The phrase "all models are wrong, but some are useful" is attributed to British statistician George E.P. Box. A domain model does not escape from this statement as it is a human-made construct that serves an engineering purpose. Therefore, the presented domain model is appropriate for describing the context of software architecture evaluations, but it should not be seen as the only "correct" model.

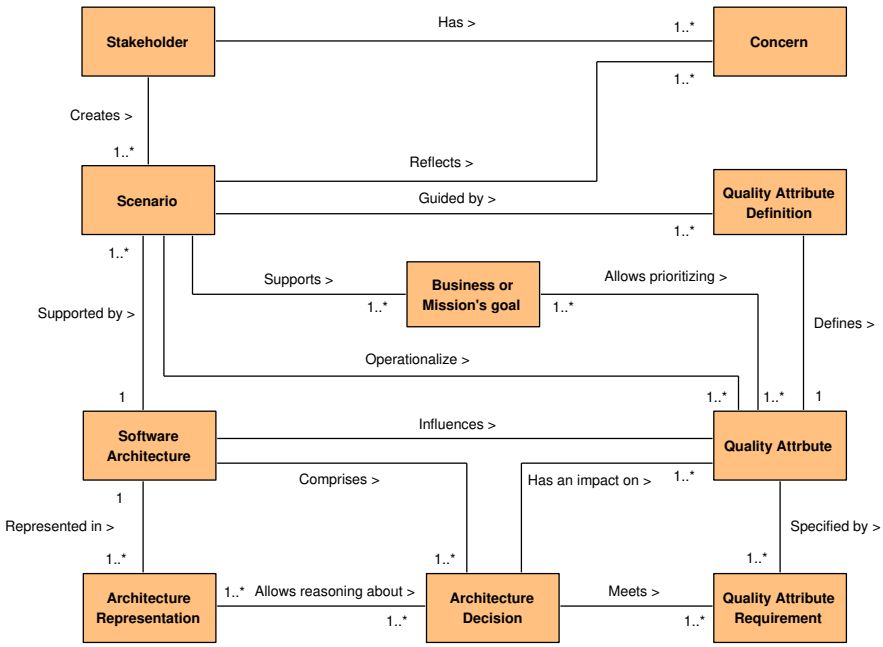


Figure B.1: Software Architecture Evaluation Domain. Own work (in progress).

Appendix C

The Software Architecture Evaluation Process

A software architecture evaluation can be described with process notation. The diagram in figure C.1 provides a BPMN[1] representation for a generic software architecture evaluation effort.

This representation is consistent with the "alphas" in [53] and also with the process-view descriptions presented in [124] and [44].

This description is not intended to provide a process-instance-level description of software architecture evaluation efforts. Rather it is a general description applicable to all architecture evaluation initiatives.

Methods are expected to provide concrete steps for each one of the tasks presented in this process, thus providing the required elements for an architecture evaluation process instance.

The four phases identified are describes as follows:

1. **Scope the evaluation effort:** An architecture evaluation is a practical approach for reviewing an architecture with some goals in mind. As noted by some authors, evaluating an architecture should be a "cheap way to avoid disaster" [6]. Thus, it does not make sense to have evaluation efforts lasting more than a handful amount of time relative to the software engineering total effort. The way we avoid evaluations becoming

[1] "Business Process Model & Notation (BPMN) - `https://www.omg.org/bpmn/`

uncontrolled processes is by scoping the evaluation effort which means defining a goal and setting the expectations.

2. **Set up the evaluation effort:** Before conducting the evaluation a set-up is expected. In this set-up phase we expect initial documentation to be prepared (as required by many evaluation methods). We also expect to meet all "entrance requirements." For example, when dealing with critical information we expect signing non-disclosure agreements before conducting the evaluation.

3. **Conduct the evaluation:** This task represents all the activities carried out for evaluating the architecture. This is where the evaluation methods have more visible impact. Conducting evaluation includes "sub tasks" such as reviewing and perfecting already prepared documentation, planning and conducting meetings, brainstorming sessions, and challenging architecture decisions.

4. **Package evaluation results:** The last, formally defined phase in an architecture evaluation is packaging results. Here packaging means some form of reporting: it can be a written report, a presentation, o some artifacts stored as models in a modeling software. Partial results are welcomed because they maintain stakeholders engaged and motivated. Therefore, this phase should be seen as a formal one in which results are delivered, but partial results can be delivered at any stage.

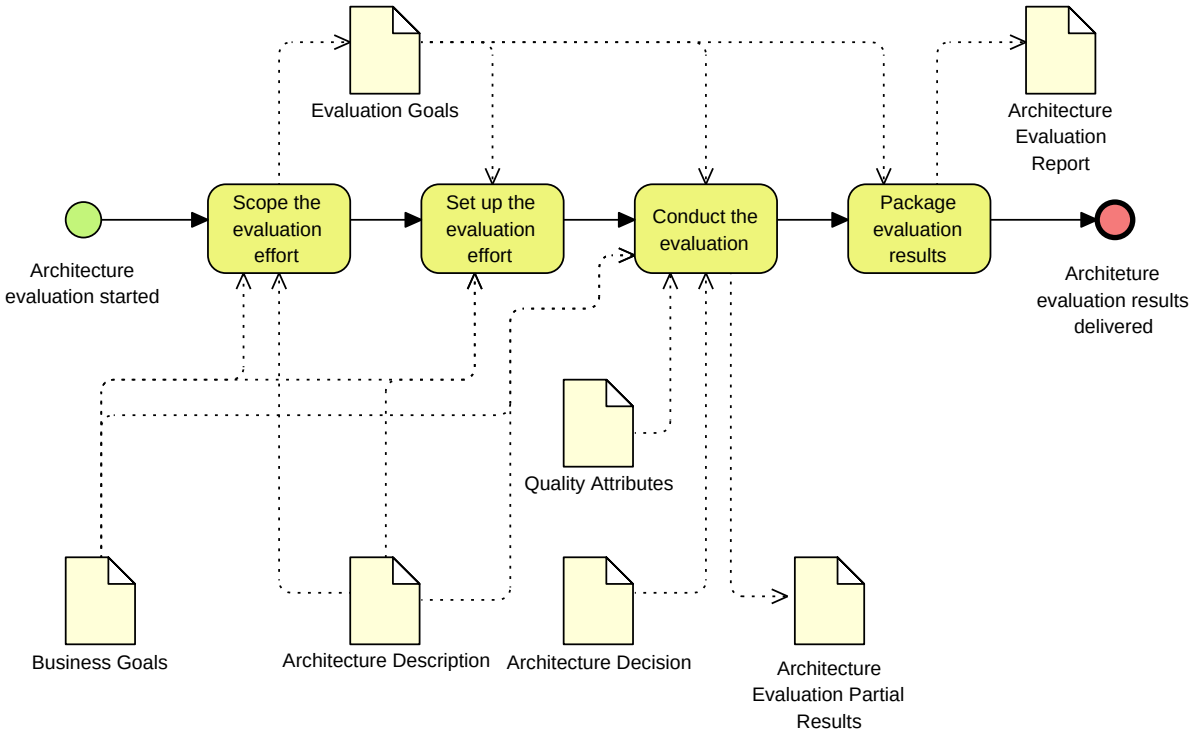


Figure C.1: Software Architecture Evaluation Process. Own work.

Appendix D

Indications For Case Study Replication

This appendix presents three key elements for replication purposes.

The Alpha State Cards

The most important element for the replication of the case studies is the set of Alpha cards. They are provided in printable form as a PDF file in [31]. The file includes cards for each State in each Alpha, that is, a total of sixteen Alpha State Cards.

Researchers who want to replicate these case studies can print these cards and use them to guide a software architecture evaluation, starting from the first state in each Alpha and progressing by following the next States. The checklists of a State Card must be verified before claiming that the State has been reached.

Architecture Decision Records

Architecture Decision Records (ADRs) are important for the architecture evaluation as well as for the case study. The ADRs are used to describe in a structured way the architecture decisions, and they embody key observable, case-study-related aspects.

The template used in this work characterizes each identified architecture decision with the following structure:

- **Title:** the name of the architecture decision written as a noun phrase.

- **Context:**a brief itemized explanation of the context in which the architecture decision has been made. Decision forces can be mentioned here.
- **Decision:** a brief itemized explanation of the architecture decision itself.
- **Status:** an indicator that tells if the architecture decision is implemented, deprecated, or subject to improvement.
- **Consequences:** an itemized explanation of the consequences of the architecture decision.

The ADRs can be stored using markdown language (which is supported and interpreted by GitHub); see an example in [31].

Architecture Evaluation Report Structure

The report of a software architecture evaluation is a key work product because it condenses all the architecture-related information collected in the evaluation activities.

In the cases presented in this article, each report was about 30 pages long and had the following structure:

- An Introduction or Preamble section, describing the circumstances in which the architecture evaluation takes places, the motivation for evaluating the architecture, the approach or approaches used, and the main findings.
- A Business or Mission Motivation section, describing the business or mission context in which the software product has been conceived. This section also describes the goals the organization has with the software system (for example, if the organization wants to reduce costs from a database system contract).
- A Requirements/Decisions mapping section, presenting the qualities (e.g. from a quality model), their prioritization, and their relation to the scenarios or architecturally significant requirements.
- A Decisions section, presenting the analyzed decisions in relation to the software qualities and their trade-offs, decision status, and main risks. In this section, it is key to argue how such requirements are supported or inhibited by the architecture.

- A final section summarizing the findings and presenting the main conclusions of the evaluation.

Appendix E

Mann-Whitney U Test Python Code

```
import pandas as pd
from scipy.stats import mannwhitneyu


pd.set_option('display.max_columns', None)

# lookup dictionary for translating questions to english
answers_lookup = {
    "Para ordenar y mantener ordenado el proceso de evaluación de arquitectura de software.":
        "Set in order and maintain ordering in the software architecture evaluation process.",
    "Para evitar redundancia de trabajo.":
        "Avoid work duplication.",
    "Para mantener expectativas y alcance controlados.":
        "Maintain expectancy and scope controlled.",
    "Para mantener costos acotados.":
        "Maintain costs bounded.",
    "Para determinar cuándo una evaluación de arquitectura de software está en problemas.":
        "Assess when a software architecture evaluation is in problems.",
    "Para ser más objetivo.":
        "Be more objective.",
    "Para mejorar la planificación.":
        "Improve planning.",
    "Para resolver a tiempo problemas que emergen en una evaluación de arquitectura compleja.":
        "Solve in time problems that emerge in complex architecture evaluation.",
    "Definir tareas concretas.":
        "Define concrete tasks.",
    "Definir roles concretos.":
        "Define concrete roles.",
    "Definir productos de trabajo concretos.":
        "Define concrete work products.",
    "Compartir conocimiento acerca de cómo llevar a cabo una evaluación de arquitectura de software.":
        "Share knowledge about how a software architecture evaluation is run.",
```

```
    "Definir y seleccionar herramienta(s) para facilitar la evaluación.\xa0":
        "Define and select tools to facilitate evaluation."
}

# define equivalence between likert answers and a number.
# as noted in the mann-whitney u test, the distance between each answer level should be the same.
likert_equivalence = {
    "Muy desacuerdo": 1,
    "Desacuerdo": 2,
    "Neutral": 3,
    "De acuerdo": 4,
    "Muy de acuerdo": 5
}

# a pandas dataframe is used to process the survey raw data
df = pd.read_excel("survey_raw.xlsx")

# use only columns with answers in the 13 questions that will be compared
df_filtered = df.iloc[:, [13, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54]]

# the anaysis is grouped in participants with low or no experience and medium to high experience
df_low_no_experience = df_filtered[
    (df_filtered["Evaluación de arquitecturas de software (formal o informal)"] == "Sin experiencia") |
    (df_filtered["Evaluación de arquitecturas de software (formal o informal)"] == "Poca experiencia")
    ]

df_medium_to_high_experience = df_filtered[
    (df_filtered["Evaluación de arquitecturas de software (formal o informal)"] == "Alto nivel de experiencia") |
    (df_filtered["Evaluación de arquitecturas de software (formal o informal)"] == "Moderado nivel de experiencia") |
    (df_filtered["Evaluación de arquitecturas de software (formal o informal)"] == "Nivel de experiencia medio")
    ]

print(df_low_no_experience.head())

# first column is asking for the experience
questions_count = len(df_filtered.columns) - 1

points = 1
while points <= questions_count:
    df_low_no_experience.loc[:, "Points " + str(points)] = df_low_no_experience.iloc[:, points].map(likert_equivalence)
    points += 1

points = 1
while points <= questions_count:
    df_medium_to_high_experience.loc[:, "Points " + str(points)] = df_medium_to_high_experience.iloc[:, points].map(
        likert_equivalence)
    points += 1

print(df_low_no_experience)
```

```
dict_to_df = {
    "Question": [],
    "p-value": [],
    "Average low-no exp": [],
    "Average medium-to-high exp": []
}
points = 1
while points <= questions_count:
    batch_1 = df_low_no_experience.iloc[:, points + questions_count].tolist()
    batch_2 = df_medium_to_high_experience.iloc[:, points + questions_count].tolist()

    average_low_no_experience = sum(batch_1)/len(batch_1)
    average_medium_to_high_experience = sum(batch_2)/len(batch_2)

    stat, p_value = mannwhitneyu(batch_1, batch_2)
    print("*Question: " + answers_lookup[(df_low_no_experience.columns[points])] + str("*"))
    print('Statistics=%.2f, p=%.2f' % (stat, p_value))
    dict_to_df["Question"].append(answers_lookup[(df_low_no_experience.columns[points])])
    dict_to_df["p-value"].append(p_value)
    dict_to_df["Average low-no exp"].append(average_low_no_experience)
    dict_to_df["Average medium-to-high exp"].append(average_medium_to_high_experience)

    alpha = 0.05
    if p_value < alpha:
        print('Reject Null Hypothesis (Significant difference between two samples)')
    else:
        print('Do not Reject Null Hypothesis (No significant difference between two samples)')

    points = points + 1


final = pd.DataFrame(dict_to_df).sort_values(by="p-value")
final["Difference"] = abs(final["Average medium-to-high exp"] - final["Average low-no exp"])

print(final)
```

Appendix F

Alpha States and Checklists

In this appendix, I present the complete checklists for each State in each Alpha. The checklists are also available as Alpha State Cards in [31]. For a State to be achieved, all the items in the specific checklist must be satisfied. A more relaxed rule could eventually be defined, meaning that a State could be marked as achieved if a "good-enough" part of the checklist is fulfilled. In this paper, the results were observed by following the rule that a checklist must be completely fulfilled to reach a State. More study is necessary to determine if a "good-enough" rule can be defined.

Items in the checklists are identified for reference purposes only, especially in case studies of Chapter 6.

F.1 Alpha: Quality Attributes

F.1.1 State: Identified

- QA.S1.1 - Quality attributes are explicitly identified.
- QA.S1.2 - Some form of quality attributes operationalization is observed (e.g., scenarios).
- QA.S1.3 - Quality attributes are recorded.

F.1.2 State: Tradeoffs understood

- QA.S2.1 - The relative importance or priority of quality attributes is established.
- QA.S2.2 - The determined prioritization is used to focus on key attributes.
- QA.S2.3 - Tradeoffs between quality attributes are analyzed and understood.

F.1.3 State: Addressed

- QA.S3.1 - Prioritized quality attributes were taken into account in the evaluation.
- QA.S3.2 - Tradeoffs recorded and explained.

F.2 Alpha: Architecture Decisions

F.2.1 State: Tacit identification

- AD.S1.1 - Decisions are identified in tacit form.
- AD.S1.2 - Decisions are not required to be recorded or explicitly defined.
- AD.S1.3 - There is some knowledge about the rationale of the decision.
- AD.S1.4 - There is some knowledge about who made the decision.

F.2.2 State: Explicit identification

- AD.S2.1 - Decisions are communicated in a clear and explicit characterized way.
- AD.S2.2 - There are efforts to record decisions, although records can be ad-hoc.
- AD.S2.3 - It is clear both who made the decision and the rationale behind it.
- AD.S2.4 - The relationships between decisions are identified.

F.2.3 State: Recorded and addressed

- AD.S3.1 - Decisions are recorded in a specific format such as ADRs.
- AD.S3.2 - Decisions are subject to version control/configuration management.
- AD.S3.3 - Key decisions have been reviewed according to the chosen evaluation approach.

F.3 Alpha: Architecture Description

F.3.1 State: General overview

- ADS.S1.1 - An architecture description exists in the form of a general overview (most of the time this overview shows the components deployment).
- ADS.S1.2 - The architecture description does not present any specific architecture component-connector configuration.

F.3.2 State: Models developed

- ADS.S2.1 - The architecture description presents specific (ad-hoc) configurations for the system.
- ADS.S2.2 - The description includes important views, and the viewpoints are justified.
- ADS.S2.3 - The description can be expressed by means of informal language (e.g., whiteboard diagrams).

F.3.3 State: Completed

- ADS.S3.1 - The architecture description has reached a state that can be considered complete for the system.
- ADS.S3.2 - The description is expected to be expressed using description-oriented languages such as Archi, UML, SysML, ADLs, among other alternatives.

F.4 Alpha: Business Goals

F.4.1 State: Identified

- BG.S1.1 - The business or mission goals are explicitly identified.
- BG.S1.2 - Architecture drivers (both business and mission) are identified.

F.4.2 State: Principles established

- BG.S2.1 - Statements for the definition of the architecture are clear and explicitly identified.
- BG.S2.2 - The principles are used to analyze part or all of the architecture.

F.4.3 State: Addressed

- BG.S3.1 - Business goals are used in the evaluation of the architecture.
- BG.S3.2 - The results explain the justification for how the architecture would help meet the goals.

F.5 Alpha: Evaluation Adoption

F.5.1 State: Method or approach chosen

- EA.S1.1 - Evaluation methods have been reviewed.
- EA.S1.2 - The criteria for choosing an evaluation method have been signed off.
- EA.S1.3 - The selection of an evaluation method has been signed off.

F.5.2 State: Method or approach integrated

- EA.S2.1 - The method chosen has been explained to the evaluation stakeholders.

- EA.S2.2 - Activities or steps defined by the evaluation method are in execution according to the plan and, if required, tailored activities.
- EA.S2.3 - Partial results are being delivered to stakeholders.

F.5.3 State: Working well

- EA.S3.1 - The planned activities of the evaluation are carried out with controlled deviation from the plan.
- EA.S3.2 - Consistent use of tools.
- EA.S3.3 - Consistent results from the evaluation initiative are being delivered.
- EA.S3.4 - Stakeholders agree that the evaluation goals are being met.

F.5.4 State: Organizational memory accrued

- EA.S4.1 - A retrospective analysis has been performed.
- EA.S4.2 - Effort, practices, and what worked well have been recorded.
- EA.S4.3 - The organizational memory has been updated.

Bibliography

[1] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software Architecture: Foundations, Theory and Practice*. Addison-Wesley, 2007.

[2] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice, 4th Edition*. SEI series in software engineering, Addison-Wesley Professional, 2021.

[3] O. Sievi-Korte, S. Beecham, and I. Richardson, "Challenges and recommended practices for software architecting in global software development," *Information and Software Technology*, vol. 106, pp. 234–253, 2019.

[4] N. Rozanski and E. Woods, *Software Systems Architecture: Working with Stakeholders using Viewpoints and Perspectives*. Addison Wesley, 2 ed., 2011.

[5] J. Bosch, *Design and use of software architectures: adopting and evolving a product-line approach*. USA: ACM Press/Addison-Wesley Publishing Co., 2000.

[6] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*. SEI Series in Software Engineering, Boston, MA: Addison-Wesley, 2001.

[7] E. Klotins, T. Gorschek, and M. Wilson, "Continuous software engineering: Introducing an industry readiness model," *IEEE Software*, vol. 40, no. 4, pp. 77–87, 2023.

[8] C. Ciceri, D. Farley, N. Ford, A. Harmel-Law, M. Keeling, C. Lilienthal, J. Rosa, A. von Zitzewitz, R. Weiss, and E. Woods, *Software Architecture Metrics*. O'Reilly Media, 2022.

[9] B. Fitzgerald and K.-J. Stol, "Continuous software engineering: A roadmap and agenda," *Journal of Systems and Software*, vol. 123, pp. 176–189, 2017.

[10] R. C. Soares, R. Capilla, V. dos Santos, and E. Y. Nakagawa, "Trends in continuous evaluation of software architectures," *Computing*, vol. 105, pp. 1957–1980, Sept. 2023.

[11] R. C. Soares, V. d. Santos, and E. Y. Nakagawa, "Continuous evaluation of software architectures: an overview of the state of the art," in *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, SAC '22, (New York, NY, USA), p. 1425–1431, Association for Computing Machinery, 2022.

[12] L. Dobrica and E. Niemela, "A survey on software architecture analysis methods," *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 638–653, 2002.

[13] C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, and P. America, "A general model of software architecture design derived from five industrial approaches," *Journal of Systems and Software*, vol. 80, no. 1, pp. 106–126, 2007.

[14] R. N. Taylor and A. van der Hoek, "Software design and architecture the once and future focus of software engineering," in *Future of Software Engineering (FOSE '07)*, pp. 226–243, 2007.

[15] H. Cervantes and R. Kazman, *Designing Software Architectures: A Practical Approach*. Addison-Wesley Professional, 2nd ed., 2024.

[16] J. Madison, "Agile architecture interactions," *IEEE Software*, vol. 27, no. 2, pp. 41–48, 2010.

[17] S. Bellomo, I. Gorton, and R. Kazman, "Toward agile architecture: Insights from 15 years of atam data," *IEEE Software*, vol. 32, no. 5, pp. 38–45, 2015.

[18] V. Reijonen, J. Koskinen, and I. Haikala, "Experiences from scenario-based architecture evaluations with atam," in *Software Architecture* (M. A. Babar and I. Gorton, eds.), (Berlin, Heidelberg), pp. 214–229, Springer Berlin Heidelberg, 2010.

[19] N. Ford, M. Richards, P. Sadalage, and Z. Dehghani, *Software Architecture: The Hard Parts*. O'Reilly Media, 2021.

[20] S. Becker, M. Trifu, and R. Reussner, "Towards supporting evolution of service-oriented architectures through quality impact prediction," in *2008 23rd IEEE/ACM International Conference on Automated Software Engineering - Workshops*, pp. 77–81, 2008.

[21] U. van Heesch, V.-P. Eloranta, P. Avgeriou, K. Koskimies, and N. Harrison, "Decision-centric architecture reviews," *IEEE Software*, vol. 31, no. 1, pp. 69–76, 2014.

[22] P. Cruz, H. Astudillo, R. Hilliard, and M. Collado, "Assessing migration of a 20-year-old system to a micro-service platform using atam," in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pp. 174–181, 2019.

[23] P. Cruz, L. Salinas, and H. Astudillo, "Quick evaluation of a software architecture using the decision-centric architecture review method: An experience report," in *Software Architecture* (A. Jansen, I. Malavolta, H. Muccini, I. Ozkaya, and O. Zimmermann, eds.), (Cham), pp. 281–295, Springer International Publishing, 2020.

[24] P. Cruz, G. Ulloa, D. S. Martin, and A. Veloz, "Software architecture evaluation of a machine learning enabled system: A case study," in *2023 42nd IEEE International Conference of the Chilean Computer Science Society (SCCC)*, pp. 1–8, 2023.

[25] P. Cruz and H. Astudillo, "Positive side-effects of evaluating a software architecture," in *Software Architecture. ECSA 2024 Tracks and Workshops* (A. Ampatzoglou, J. Pérez, B. Buhnova, V. Lenarduzzi, C. C. Venters, U. Zdun, K. Drira, L. Rebelo, D. Di Pompeo, M. Tucci, E. Y. Nakagawa, and E. Navarro, eds.), (Cham), pp. 167–177, Springer Nature Switzerland, 2024.

[26] D. Schon, *The reflective practitioner. How professionals think in action.* London: Temple Smith, 1983.

[27] I. Jacobson, P.-W. Ng, P. McMahon, I. Spence, and S. Lidman, "The essence of software engineering: The semat kernel: A thinking framework in the form of an actionable kernel," *Queue*, vol. 10, p. 40–51, oct 2012.

[28] Object Management Group, OMG, "Essence – Kernel and Language for Software Engineering Methods 1.2," 2018.

[29] I. Jacobson, H. B. Lawson, P.-W. Ng, P. E. McMahon, and M. Goedicke, *The Essentials of Modern Software Engineering: Free the Practices from the Method Prisons!* Association for Computing Machinery and Morgan & Claypool, 2019.

[30] P. Cruz, H. Astudillo, and C. M. Zapata-Jaramillo, "Extending the semat kernel to represent and assess software architecture evaluations," in *2023 XLIX Latin American Computer Conference (CLEI)*, pp. 1–8, 2023.

[31] P. Cruz, "Software Architecture Evaluation SEMAT-based Representation Repository." `https://github.com/pcruzn/arch-eval-semat`, 2026.

[32] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln, *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012.

[33] N. A. Ernst, J. Klein, M. Bartolini, J. Coles, and N. Rees, "Architecting complex, long-lived scientific software," *Journal of Systems and Software*, vol. 204, p. 111732, 2023.

[34] R. Kazman, G. Abowd, L. Bass, and P. Clements, "Scenario-based analysis of software architecture," *IEEE Software*, vol. 13, no. 6, pp. 47–55, 1996.

[35] J. Cámara, R. de Lemos, M. Vieira, R. Almeida, and R. Ventura, "Architecture-based resilience evaluation for self-adaptive systems," *Computing*, vol. 95, pp. 689–722, Aug. 2013.

[36] R. Kazman, M. Klein, and P. Clements, "Atam: Method for architecture evaluation," Tech. Rep. CMU/SEI-2000-TR-004, Aug 2000. Accessed: 2024-Apr-25.

[37] P. Bengtsson and J. Bosch, "Scenario-based software architecture reengineering," in *Proceedings. Fifth International Conference on Software Reuse (Cat. No.98TB100203)*, pp. 308–317, 1998.

[38] N. Harrison and P. Avgeriou, "Pattern-based architecture reviews," *IEEE Software*, vol. 28, no. 6, pp. 66–71, 2011.

[39] J. Knodel and M. Naab, "Software architecture evaluation in practice: Retrospective on more than 50 architecture evaluations in industry," in *2014 IEEE/IFIP Conference on Software Architecture*, pp. 115–124, 2014.

[40] J. Maranzano, S. Rozsypal, G. Zimmerman, G. Warnken, P. Wirth, and D. Weiss, "Architecture reviews: practice and experience," *IEEE Software*, vol. 22, no. 2, pp. 34–43, 2005.

[41] R. Kazman and L. Bass, "Making architecture reviews work in the real world," *IEEE Software*, vol. 19, no. 1, pp. 67–73, 2002.

[42] S. Ferber, P. Heidl, and P. Lutz, "Reviewing product line architectures: Experience report of atam in an automotive context," in *Software Product-Family Engineering* (F. van der Linden, ed.), (Berlin, Heidelberg), pp. 364–382, Springer Berlin Heidelberg, 2002.

[43] M. Ali Babar, L. Bass, and I. Gorton, "Factors influencing industrial practices of software architecture evaluation: An empirical investigation," in *Software Architectures, Components, and Applications* (S. Overhage, C. A. Szyperski, R. Reussner, and J. A. Stafford, eds.), (Berlin, Heidelberg), pp. 90–107, Springer Berlin Heidelberg, 2007.

[44] J. Knodel and M. Naab, *Pragmatic Evaluation of Software Architectures*. Springer Publishing Company, Incorporated, 1st ed., 2016.

[45] H. Obbink, P. Kruchten, W. Kozaczynski, H. Postema, A. Ran, L. Dominick, R. Kazman, R. Hilliard, W. Tracz, and E. Kahane, "Software Architecture Review and Assessment (SARA) Report (Version 1.0)," 2002.

[46] ISO/IEC/IEEE, "ISO/IEC/IEEE 42030:2019 software, systems and enterprise — architecture evaluation framework," 2019.

[47] ISO/IEC/IEEE, "ISO/IEC/IEEE 42020:2019 software, systems and enterprise — architecture processes," 2019.

[48] I. Jacobson, B. Meyer and R. Soley, "The SEMAT Initiative: A Call for Action." http://www.drdobbs.com/architecture-and-design/the-semat-initiative-a-call-for-action/222001342 (rev: 2024/04/01), 2009.

[49] T. Päivärinta and K. Smolander, "Theorizing about software development practices," *Science of Computer Programming*, vol. 101, pp. 124 – 135, 2015. Towards general theories of software engineering.

[50] I. Jacobson, S. Huang, M. Kajko-Mattsson, P. Mcmahon, and E. Seymour, "Semat–three year vision," *Program. Comput. Softw.*, vol. 38, p. 1–12, jan 2012.

[51] P. Ray and P. Pal, "Extending the semat kernel for the practice of designing and implementing microservice-based applications using domain driven design," in *2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T)*, pp. 1–4, 2020.

[52] V. Savić and E. Varga, "Extending the SEMAT Kernel with the TDD practice," *IET Software*, vol. 12, pp. 85–95, Apr. 2018.

[53] Double-Blind, "Extending the semat kernel to represent and assess software architecture evaluations," in *2023 XVLIV Latin American Computer Conference (CLEI)*, 2023.

[54] J. Nørbjerg and Y. Dittrich, "The never-ending story–how companies transition to and sustain continuous software engineering practices," *Journal of Systems and Software*, vol. 213, p. 112056, 2024.

[55] R. Stake, *Multiple Case Study Analysis*. Guilford Publications, 2013.

[56] T. A. Abma and R. E. Stake, "Science of the particular: An advocacy of naturalistic case study in health research," *Qualitative Health Research*, vol. 24, no. 8, pp. 1150–1161, 2014. PMID: 25028158.

[57] C. Wohlin and A. Rainer, "Is it a case study?—a critical analysis and guidance," *Journal of Systems and Software*, vol. 192, p. 111395, 2022.

[58] D. R. Wallace and M. V. Zelkowitz, "Experimental models for validating technology," *Computer*, vol. 31, pp. 23–31, may 1998.

[59] R. Yin, *Case Study Research and Applications: Design and Methods*. SAGE Publications, 2017.

[60] C. Robson, *Real World Research.* Wiley, 2024.

[61] D. Silverman, *Doing qualitative research : a practical handbook / David Silverman.* London ;: SAGE, third edition. ed., 2010.

[62] F. Shull, "Sharing your story," *IEEE Software*, vol. 30, pp. 4–7, may 2013.

[63] S. L. Pfleeger and B. A. Kitchenham, "Principles of survey research: part 1: turning lemons into lemonade," *SIGSOFT Softw. Eng. Notes*, vol. 26, p. 16–18, Nov. 2001.

[64] B. A. Kitchenham and S. L. Pfleeger, "Principles of survey research part 2: designing a survey," *SIGSOFT Softw. Eng. Notes*, vol. 27, p. 18–20, Jan. 2002.

[65] B. A. Kitchenham and S. L. Pfleeger, "Principles of survey research: part 3: constructing a survey instrument," *SIGSOFT Softw. Eng. Notes*, vol. 27, p. 20–24, Mar. 2002.

[66] T. CMMI Product Team, "Cmmi for development, version 1.3," Tech. Rep. CMU/SEI-2010-TR-033, Nov 2010. Accessed: 2024-Jan-18.

[67] J. Herbsleb, D. Zubrow, D. Goldenson, W. Hayes, and M. Paulk, "Software quality and the capability maturity model," *Commun. ACM*, vol. 40, p. 30–40, jun 1997.

[68] ISO/IEC/IEEE, "ISO/IEC/IEEE 42030:2019 software, systems and enterprise — architecture evaluation framework," 2019.

[69] ISO/IEC/IEEE, "ISO/IEC/IEEE 42020:2019 software, systems and enterprise — architecture processes," 2019.

[70] A. van Lamsweerde, "Goal-oriented requirements engineering: a guided tour," in *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, pp. 249–262, 2001.

[71] N. B. Harrison and P. Avgeriou, "Leveraging architecture patterns to satisfy quality attributes," in *Software Architecture* (F. Oquendo, ed.), (Berlin, Heidelberg), pp. 263–270, Springer Berlin Heidelberg, 2007.

[72] I. Gorton, *Essential Software Architecture*. Springer Publishing Company, Incorporated, 2nd ed., 2011.

[73] ISO/IEC, "ISO/IEC 25010:2023 systems and software engineering — systems and software quality requirements and evaluation (square) — product quality model," 2023.

[74] J. Axelsson and M. Skoglund, "Quality assurance in software ecosystems: A systematic literature mapping and research agenda," *Journal of Systems and Software*, vol. 114, pp. 69–81, 2016.

[75] F. Bachmann, L. Bass, M. Klein, and C. Shelton, "Designing software architectures to achieve quality attribute requirements," *Software, IEE Proceedings* -, vol. 152, pp. 153 – 165, 09 2005.

[76] A. H. Dutoit, R. McCall, I. Mistrik, and B. Paech, *Rationale Management in Software Engineering*. Berlin, Heidelberg: Springer-Verlag, 2006.

[77] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *SIGSOFT Softw. Eng. Notes*, vol. 17, p. 40–52, oct 1992.

[78] J. Bosch, "Software architecture: The next step," in *Software Architecture* (F. Oquendo, B. C. Warboys, and R. Morrison, eds.), (Berlin, Heidelberg), pp. 194–199, Springer Berlin Heidelberg, 2004.

[79] M. Che, D. E. Perry, and G. Yang, "Evaluating architectural design decision paradigms in global software development," *International Journal of Software Engineering and Knowledge Engineering*, vol. 25, no. 09n10, pp. 1677–1692, 2015.

[80] G. Me, C. Calero, and P. Lago, "Architectural patterns and quality attributes interaction," in *2016 Qualitative Reasoning about Software Architectures (QRASA)*, pp. 27–36, 2016.

[81] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, and M. Ali Babar, "A comparative study of architecture knowledge management tools," *Journal of Systems and Software*, vol. 83, no. 3, pp. 352–370, 2010.

[82] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, pp. 109–120, 2005.

[83] R. Capilla, A. Jansen, A. Tang, P. Avgeriou, and M. A. Babar, "10 years of software architecture knowledge management: Practice and future," *Journal of Systems and Software*, vol. 116, pp. 191–205, 2016.

[84] M. A. Babar, T. Dingsyr, P. Lago, and H. van Vliet, *Software Architecture Knowledge Management: Theory and Practice*. Springer Publishing Company, Incorporated, 1st ed., 2009.

[85] D. Tofan, "Tacit architectural knowledge," in *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ECSA '10, (New York, NY, USA), p. 9–11, Association for Computing Machinery, 2010.

[86] G. Borrego, A. L. Morán, R. R. Palacio, A. Vizcaíno, and F. O. García, "Towards a reduction in architectural knowledge vaporization during agile global software development," *Information and Software Technology*, vol. 112, pp. 68–82, 2019.

[87] I. Nonaka and H. Takeuchi, *The Knowledge-Creating Company*. Oxford University Press, 1995.

[88] S. Eilon, "What is a decision?," *Management Science*, vol. 16, no. 4, pp. B172–B189, 1969.

[89] ISO/IEC/IEEE, "ISO/IEC/IEEE 42010:2022 software, systems and enterprise — architecture description," 2022.

[90] F. Salger, "Software architecture evaluation in global software development projects," in *On the Move to Meaningful Internet Systems: OTM 2009 Workshops* (R. Meersman, P. Herrero, and T. Dillon, eds.), (Berlin, Heidelberg), pp. 391–400, Springer Berlin Heidelberg, 2009.

[91] R. Verdecchia, P. Kruchten, P. Lago, and I. Malavolta, "Building and evaluating a theory of architectural technical debt in software-intensive systems," *Journal of Systems and Software*, vol. 176, p. 110925, 2021.

[92] E. Woods and N. Rozanski, "Unifying software architecture with its implementation," in *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ECSA '10, (New York, NY, USA), p. 55–58, Association for Computing Machinery, 2010.

[93] G. Fairbanks, "Software architecture is a set of abstractions," *IEEE Software*, vol. 40, pp. 110–113, jul 2023.

[94] E. Han, "Setting business goals & objectives: 4 considerations," 2023. Accessed: 2025-03-20.

[95] R. Kazman and L. Bass, "Categorizing business goals for software architectures," Tech. Rep. CMU/SEI-2005-TR-021, Dec 2005. Accessed: 2024-Jul-31.

[96] L. Bass and P. Clements, *Business Goals and Architecture*, pp. 183–195. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[97] P. Clements and L. Bass, "Business goals as architectural knowledge," in *Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge*, SHARK '10, (New York, NY, USA), p. 9–12, Association for Computing Machinery, 2010.

[98] L. Bass and P. Clements, " The Business Goals Viewpoint ," *IEEE Software*, vol. 27, pp. 38–45, Nov. 2010.

[99] D. Gross and E. Yu, "Evolving system architecture to meet changing business goals: an agent and goal-oriented approach," in *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, pp. 316–317, 2001.

[100] P. Clements and L. Bass, "Relating business goals to architecturally significant requirements for software systems," Tech. Rep. CMU/SEI-2010-TN-018, May 2010. Accessed: 2024-Jul-31.

[101] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *The NFR Framework in Action*, pp. 15–45. Boston, MA: Springer US, 2000.

[102] M. T. Sletholt, J. E. Hannay, D. Pfahl, and H. P. Langtangen, "What do we know about scientific software development's agile practices?," *Computing in Science & Engineering*, vol. 14, no. 2, pp. 24–37, 2012.

[103] D. Morey, M. T. Maybury, and B. Thuraisingham, *Knowledge Management: Classic and Contemporary Works*. The MIT Press, 12 2000.

[104] D. Winkler, S. Biffl, and M. A. Babar, "An empirical investigation of scenarios gained and lost in architecture evaluation meetings," in *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '08, (New York, NY, USA), p. 348–350, Association for Computing Machinery, 2008.

[105] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, pp. 131–164, Apr. 2009.

[106] R. E. Stake, *The art of case study research.* The art of case study research., Thousand Oaks, CA, US: Sage Publications, Inc, 1995. Pages: xv, 175.

[107] S. M. Hussain, S. N. Bhatti, and M. F. Ur Rasool, "Legacy system and ways of its evolution," in *2017 International Conference on Communication Technologies (ComTech)*, pp. 56–59, 2017.

[108] C. Wagner, *Model-Driven Software Migration: A Methodology Reengineering, Recovery and Modernization of Legacy Systems*. Springer Vieweg, 2014.

[109] M. Abdellatif, A. Shatnawi, H. Mili, N. Moha, G. E. Boussaidi, G. Hecht, J. Privat, and Y.-G. Guéhéneuc, "A taxonomy of service identification approaches for legacy software systems modernization," *Journal of Systems and Software*, vol. 173, p. 110868, 2021.

[110] J. Bisbal, D. Lawless, B. Wu, and J. Grimson, "Legacy information systems: issues and directions," *IEEE Software*, vol. 16, no. 5, pp. 103–111, 1999.

[111] K. Bennett, "Legacy systems: coping with success," *IEEE Software*, vol. 12, no. 1, pp. 19–23, 1995.

[112] I. Yarza, M. Azkarate-askatsua, P. Onaindia, K. Grüttner, P. Ittershagen, and W. Nebel, "Legacy software migration based on timing contract aware real-time execution environments," *Journal of Systems and Software*, vol. 172, p. 110849, 2021.

[113] M. Barbacci and W. Wood, "Architecture tradeoff analyses of c4isr products," Tech. Rep. CMU/SEI-99-TR-014, Jul 1999. Accessed: 2024-Aug-2.

[114] P. Cruz and H. Astudillo, "Assessing development teams and organizations with the semat kernel: Lessons learned from a real-world experience," in *2017 36th International Conference of the Chilean Computer Science Society (SCCC)*, pp. 1–4, 2017.

[115] P. Cruz and H. Astudillo, "Using the semat kernel for software process assessment and practices implementation in an software process improvement initiative," in *2018 37th International Conference of the Chilean Computer Science Society (SCCC)*, pp. 1–7, 2018.

[116] M.-J. Wu, K. Zhao, and F. Fils-Aime, "Response rates of online surveys in published research: A meta-analysis," *Computers in Human Behavior Reports*, vol. 7, p. 100206, 2022.

[117] A. Ciapponi, J. Belizán, G. Piaggio, and S. Yaya, "There is life beyond the statistical significance," *Reproductive Health*, vol. 18, no. 1, p. 55, 2021.

[118] W. Shadish, T. Cook, and D. Campbell, *Experimental and Quasi-experimental Designs for Generalized Causal Inference*. Houghton Mifflin, 2002.

[119] J. A. Maxwell, *Qualitative research design: An interactive approach.* Qualitative research design: An interactive approach., Thousand Oaks, CA, US: Sage Publications, Inc, 1996.

[120] ISO/IEC, "ISO/IEC 24765:2017 systems and software engineering — vocabulary," 2017.

[121] ISO/IEC, "ISO/IEC 12207:2026 systems and software engineering. software life cycle processes," 2016.

[122] ISO/IEC, "ISO/IEC 15940:2013 systems and software engineering. software engineering environment services," 2013.

[123] ISO/IEC, "ISO/IEC 33001:2015 information technology — process assessment — concepts and terminology," 2015.

[124] G.-B. Jeong and G.-B. Kim, "A study on software architecture evaluation," in *Computational Science and Its Applications - ICCSA 2006* (M. L. Gavrilova, O. Gervasi, V. Kumar, C. J. K. Tan, D. Taniar, A. Laganá, Y. Mun, and H. Choo, eds.), (Berlin, Heidelberg), pp. 1032–1041, Springer Berlin Heidelberg, 2006.