Centro de Informática
UFPE
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

FAROL: A LIGHTWEIGHT DECISION-MAKING FRAMEWORK FOR DEFINING SOFTWARE ARCHITECTURE

By

FAGNER FERNANDES CANDIDO DA SILVA

M.Sc. Dissertation

Federal University of Pernambuco
secpos@cin.ufpe.br
https://portal.cin.ufpe.br/pos-graduacao

RECIFE/2023

Fagner Fernandes Candido da Silva

FAROL: A LIGHTWEIGHT DECISION-MAKING FRAMEWORK FOR DEFINING SOFTWARE ARCHITECTURE

A M.Sc. Dissertation presented to the Center for Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Advisor: Vinícius Cardoso Garcia

RECIFE
2023

Fagner Fernandes Candido da Silva

“FAROL: A LIGHTWEIGHT FRAMEWORK FOR DECISION-MAKING IN SOFTWARE ARCHITECTURE”

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação. Área de Concentração: Linguagens de Programação e Engenharia de Software.

Aprovado em: 31/10/2023

BANCA EXAMINADORA

gov.br Documento assinado digitalmente
KIEV SANTOS DA GAMA
Data: 17/11/2023 10:07:01-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Kiev Santos da Gama
Centro de Informática / UFPE

gov.br Documento assinado digitalmente
LINCOLN SOUZA ROCHA
Data: 17/11/2023 10:15:28-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Lincoln Souza Rocha
Departamento Computação/UFC

gov.br Documento assinado digitalmente
VINICIUS CARDOSO GARCIA
Data: 17/11/2023 12:55:45-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Vinicius Cardoso Garcia
Centro de Informática / UFPE
(**orientador**)

I dedicate this dissertation to all my family, friends and professors who gave me the necessary support to get here.

Acknowledgements

I have been blessed during my life so far. My family, especially my mother, Rosa, ingrained in my core the importance of studying and working hard. Most of the things that I conquered so far were possible due to this mindset.

I want to thank those not present in this sphere of existence but who carved their importance in my life: my father Fernando and my uncle Ednaldo (who was like a father as well) for supporting me during the early days of life.

My advisor, Professor Vinícius Garcia, helped a lot in the late phase of this work, providing me guidance and highlighting aspects to increase the quality of the dissertation at levels that I never knew were possible.

I am also grateful to Jackson Raniel for helping me define the methodology and conduct my survey study. I was utterly lost, and you showed me the way.

My bosses at work, Rodrigo Pereira and Laureano Montarroyos, for supporting me in conciliating working and studying to conclude this dissertation. With that comprehension, this challenge was possible to be completed.

My colleagues at work, Lucas and Williams, for boasting my daily motivation during this challenging pandemic. Alexandre, thank you for showing me that it is possible to conclude your master's degree with a little baby at home. Ana guided me on UX/UI practices and helped me with some graphics for my research.

Despite living abroad, my friend Rafael Ribeiro always told me I needed to finish this phase of my life. Sometimes, I even doubted that this kind of thing was possible, but with maximum effort, this conclusion was possible.

My dear friend Bruno Wolf, for turning an idea into a concrete object. The FAROL framework logo was beautifully conceived thanks to his capable hands.

Finally, I want to thank my two kids, Isadora and Guilherme, for being my primary motivation providers. Their energy is, at the same time, fantastic and very tiring. However, through studying, I can show them how important it is to keep growing despite the odds.

Você pode encontrar as coisas que perdeu, mas nunca as que abandonou.

—J.R.R. TOLKIEN

Resumo

O desenvolvimento de software é uma atividade multifacetada. Isso faz com que a tomada de decisão arquitetural seja uma questão desafiadora. Nas fases iniciais, os profissionais muitas vezes navegam num complexo labirinto de considerações, ficando à beira da tomada de decisões críticas. Essas decisões possuem um elevado potencial de impactar no ciclo de vida do sistema, podendo até mesmo, inviabilizá-lo. Sendo assim, como escolher uma proposta candidata de arquitetura de software sem perder aspectos críticos, respeitando a especificidade e a finalidade do software? Um framework de decisão arquitetural leve (FAROL) é proposto neste trabalho para preencher esta lacuna de conhecimento. A motivação e o problema de pesquisa destacam a complexidade desse tipo de decisão, o impacto da falta de processos estruturados e a necessidade de lógica de decisão. As perguntas de pesquisa desse trabalho abordam essas questões e examinam o raciocínio dos praticantes sobre decisões, fatores de influência, práticas de documentação e princípios considerados. A revisão de literatura analisa conceitos de arquitetura de software e sua relação com a tomada de decisão. Ele diferencia arquitetura e design, explica estilos e padrões e cobre teorias de cognição dual. Técnicas de decisão e métodos de documentação, como registros de decisões arquitetônicas, são explorados. Nove princípios para o raciocínio de decisão são descritos. A metodologia pesquisa especialistas brasileiros em TI para obter insights sobre as práticas de decisão atuais. As perguntas abordam a confiança no processo, documentação, desafios, princípios, fatores que influenciam à tomada de decisão. Ameaças à validade são analisadas. A comparação com as estruturas existentes destaca a possibilidade de personalização no uso do framework. Estudos de caso demonstram a aplicação do FAROL para selecionar arquiteturas. As limitações incluem a necessidade de mais avaliações empíricas em contextos e a dependência da experiência do profissional. No entanto, FAROL fornece uma metodologia abrangente para promover decisões informadas e melhoria contínua. Por fim, o FAROL visa trazer rigor sistemático ao processo ambíguo de escolha arquitetônica. Ele oferece orientação prática ancorada em teorias multidisciplinares e adaptada para projetos do mundo real. Ao elucidar o complexo cenário de decisões, o FAROL permite que as equipes naveguem pelos tradeoffs arquiteturais.

Palavras-chave: Tomada de Decisão; Framework de Tomada de Decisão; Arquitetura de Software; Design de Software; Decisões Arquiteturais.

Abstract

Software development is a multifaceted activity. This makes architectural decision making a challenging matter. In the early stages, professionals often navigate a complex maze of considerations, leaving them on the brink of making critical decisions. These decisions have a high potential to impact the life cycle of the system, and may even make it unfeasible. Therefore, how to choose a candidate software architecture proposal without losing critical aspects, respecting the specificity and purpose of the software? A lightweight architectural decision framework (FAROL) is proposed in this work to fill this knowledge gap. The motivation and research problem highlight the complexity of this type of decision, the impact of the lack of structured processes and the need for decision logic. The research questions in this work address these questions and examine practitioners' reasoning about decisions, influencing factors, documentation practices, and principles considered. The literature review analyzes software architecture concepts and their relationship with decision making. It differentiates architecture and design, explains styles and patterns, and covers theories of dual cognition. Decision techniques and documentation methods, such as records of architectural decisions, are explored. Nine principles for decision reasoning are described. The methodology surveys Brazilian IT experts to gain insights into current decision practices. The questions address confidence in the process, documentation, challenges, principles, factors that influence decision making. Threats to validity are analyzed. Comparison with existing structures highlights the possibility of customization in the use of the framework. Case studies demonstrate the application of FAROL to select architectures. Limitations include the need for more empirical assessments across contexts and the reliance on practitioner experience. However, FAROL provides a comprehensive methodology to promote informed decisions and continuous improvement. Ultimately, FAROL aims to bring systematic rigor to the ambiguous process of architectural choice. It offers practical guidance anchored in multidisciplinary theories and adapted for real-world projects. By elucidating the complex decision landscape, FAROL allows teams to navigate architectural tradeoffs.

Keywords: Decision-Making; Decision-Making Framework; Software Architecture; Software Design; Architectural Decisions.

List of Figures

List of Tables

List of Acronyms

Contents

1

INTRODUCTION

Software development is a multifaceted activity. In the early stages, practitioners often navigate a complex maze of considerations, standing on the precipice of critical decision-making. At the heart of this complexity lies software architecture—a domain teeming with intricate decisions that, once made, can significantly shape a software's lifespan. This centrality of software architecture has spurred the design of solutions to grapple with the associated complexities (VALIPOUR et al., 2009).

Software architecture touches our daily lives, from instant messaging and ordering food to scheduling appointments and virtual meetings. Nevertheless, it is crucial to understand the distinctions between software architecture and software design, even as the terms are often used interchangeably.

However, embedded within software architecture is a pivotal process—the decision-making mechanism. Though still in the early stages of a comprehensive understanding of software architecture, decision-making profoundly influences various design elements TANG et al. (2017).

This complex process, with roots in disciplines like economics (KAHNEMAN, 2003), often operates within the constraints of bounded rationality, considering decision-makers cognitive limitations (SIMON, 1990).

1.1 MOTIVATION

"Which design is the best for software?" The answer often hovers around the ambiguity: "It depends." The intricacies of software design require a balance of various competing interests and constraints, particularly in catering to stakeholders' and customers' needs.

For instance, striking a harmonious chord between security, application performance, and user experience in an enterprise setting is vital (JAYERATNAM, 2022). Such pivotal choices in software development are termed as Architectural Decisions (ADs), which mandate an overarching system perspective for their formulation (MALAN; BREDEMEYER, 2002).

How to choose a software architecture candidate proposal without missing critical aspects, respecting software specificity and purpose? This question incites a profound restlessness in

IT experts. According to several studies available in the literature, many subjects are part of software architecture, such as:

1. Architectural Style and Patterns (SHAW; GARLAN (1996); SHARMA; KUMAR; AGARWAL (2015));
2. Constraints (BASS; KAZMAN; CLEMENTS (2012));
3. Components and Interfaces (TAYLOR; MEDVIDOVIC; DASHOFY (2008));
4. Decision Rationale (TYREE; AKERMAN (2005); KLEIN (2008); PRETORIUS et al. (2018));
5. Documentation (HGRACA (2017); TANG; LIANG; VLIET (2011); JANSEN; AVGERIOU; van der Ven (2009));
6. Data Architecture (INMON; LINSTEDT; LEVINS, 2019);
7. Integration with External Systems KAZMAN; WOODS; CARRIERE (1998);
8. Requirements (de Boer; van Vliet (2009); MEI (2000));
9. Scalability and Performance Considerations (SMITH; WILLIAMS (2002); LIU (2011))
10. Technology Stack (FALATIUK; SHIROKOPETLEVA; DUDAR (2019));

Amidst this complexity, proper documentation and rationale retention are essential for effective architecture decisions. Documentation (TANG et al., 2006) and rationale (TYREE; AKERMAN, 2005) are vital to combat knowledge decay over time, enable governance and system evolution, communicate decisions, promote reuse and avoid costly rework - highlighting the need for lightweight yet systematic approaches.

Considering this outlook, ADs, due to their profound system-wide impact, demand thorough consideration, ensuring alignment with user requirements and overarching business goals.

Unfortunately, many architectural development processes overlook the explicit documentation of these decisions, making them implicitly embedded within models (TYREE; AKERMAN, 2005). This emphasizes the pivotal role of documentation in elucidating software workings and the rationale behind certain ADs (HGRACA, 2019).

Therefore, software architecture comprises a system's core structure and essential design decisions (BABAR et al., 2009); (DUTOIT et al., 2006). Decision-Making frameworks are powerful enablers in this process of selection of a software architecture candidate.

The complexity of architectural decision-making highlights the need for a clear and structured methodology. Software practitioners require guidance in making these choices. This thesis addresses this pressing issue by exploring how to choose a software architectural candidate effectively.

1.2 PROBLEM STATEMENT

While extensive research has probed into frameworks that bolster decision-making in software architecture (PRETORIUS et al., 2018), (FALESSI et al., 2011), a consensus remains elusive. There is a pressing need for software architects to have robust, efficient processes to sieve through architectural alternatives.

Another relevant aspect of Architectural Decision (AD) is technical debt. Namely, the invisible result of past decisions about software that negatively affect its future (KRUCHTEN; NORD; OZKAYA, 2012). The impact of ever-increasing debt, maintenance costs during the software lifetime, and reducing its scalability and user experience are other vital points that can impair the software lifecycle.

Considering this scenario, several key questions must be investigated to tackle the complexity problem in architectural choices. This research delves into one central question and four auxiliary questions surrounding architecture decision practices and influencers as explained in section 1.2.1.

1.2.1 Research Questions

Ensuring that knowledge of multiple architectural approaches minimizes biases in decision-making. Evaluating non-functional requirements are primary drivers in architectural decision-making (AMELLER; FRANCH, 2014). Against this backdrop, the main research question guiding this thesis is:

Main Research Question *How to choose a software architecture candidate proposal without missing critical aspects, respecting software specificity and purpose?*

To provide a comprehensive answer to this overarching question, the following secondary research questions have been delineated:

Secondary Research Question 1 *How do software practitioners reason when they make software architectural decision-making?*

Secondary Research Question 2 *What are the potential influence factors for the architectural decision-making process?*

Secondary Research Question 3 *Which principles do software architects take into consideration when making architectural design decisions?*

Secondary Research Question 4 *How architectural design decisions are documented?*

With these research questions framing the core issues, the objectives of this thesis encompass proposing a comprehensive framework to address these challenges and gathering empirical insights into current decision-making approaches.

Addressing these questions requires an intimate understanding of current methodologies and the underlying principles that shape decision-making in software architecture. Hence, the objectives of this thesis include:

Main Research Objective *This work objective proposes a decision framework for selecting/evaluating software architecture candidates.*

Secondary Research Objective 1 *This work objective is to understand which elements influence reasoning on software practitioners.*

Secondary Research Objective 2 *This work objective is to identify which factors influence the architectural decision-making process.*

Secondary Research Objective 3 *This work objective is to identify which principles are involved in decision-making on software architecture.*

Secondary Research Objective 4 *This work objective is to understand how architectural design decisions are documented.*

1.3 OUT OF SCOPE

Once we contribute by identifying and analyzing software architecture principles and factors that impair the decision-making process in this area, a set of related aspects will be left out of its scope.

While this thesis focuses on developing a decision-making framework, some related aspects of software architecture are outside its scope. In particular, this work does not directly address the following two areas:

1. **Software evaluation techniques**. We believe that several software evaluation techniques are more suitable for each context. However, trying to fit all constraints on a single method seems precarious. For example, KIM et al. (2007) conceived a lightweight value-based method. Software Engineering Institute (SEI) developed and refined other techniques, such as Architecture Trade-off Analysis Method (ATAM) and Cost Benefit Analysis Method (CBAM).

2. **Architectural Description Languages (ADLs)**. The advent of software architecture as a body of knowledge and a topic for research was accompanied by the creation of numerous notations for attempting to capture software architectures. The creation of formal notations for representing and analyzing architectural design has been the object of study since (GARLAN; PERRY, 1995).

Within its core focus, however, this thesis provides multiple key contributions surrounding architecture decision practices and the proposed framework itself, as demonstrated in section 1.4.

1.4 STATEMENT OF THE CONTRIBUTIONS

As a result of the work presented in this thesis, the following contributions may be enumerated:

a) The proposition of lightweight framework for AD.

b) An overview of software architecture definitions and related concepts;

c) An overview of the decision-making process and its relationship with software architecture decisions;

d) A survey research to evaluate architectural decision-making in Brazil;

e) The identification of the principles that software practitioners use when making architectural decisions;

f) The identification of the factors that exerts influence negatively/positively on architectural decisions;

g) An empirical survey research evaluation in the context of IT experts in Brazil and a questionnaire instrument designed for survey research studies;

h) An valid survey instrument that can be used in future studies;

1.5 OUTLINE

To present these contributions, the remainder of this thesis is structured into the following chapters: Chapter 2 provides background on software architecture definitions, architectural styles and patterns, and contrasts architecture vs. design. Chapter 3 reviews decision-making theories and techniques for software architecture, including dual cognition models, principles, and rationale documentation. Chapter 4 presents the methodology used to ground this work and explains our research questions, the survey's questions related to the objectives, and the threats to the validity. Chapter 5 presents survey results on current practices and pain points. Chapter 6 introduces the proposed lightweight architecture decision framework, including its structure, theoretical grounding, and illustrative examples. Chapter 7 concludes with a summary and directions for future work.

2

SOFTWARE ARCHITECTURE: An Overview

2.1 Introduction

The role of software architecture in the engineering of software-intensive systems is becoming increasingly important and widespread (BARAIS et al., 2008). Software architecture has been a subject of discussion in academia and the software industry, but a consensus on its definition and application has yet to be achieved.

The term architecture itself is commonly borrowed from another area of knowledge and, according to FOWLER (2003) *"... is a word we use when we want to talk about design but want to puff it up to make it sound important"*. According to BASS; KAZMAN; CLEMENTS (2012) *"architecture is foremost an abstraction of a system that selects certain details and suppresses others"*.

Nevertheless, as highlighted in Figure 2.1, we can perceive software architecture as a bridge between requirements and code, and this bridge is deeply related to software project success/failure (VALIPOUR et al., 2009). That is why knowing the core concepts around software architecture is essential.

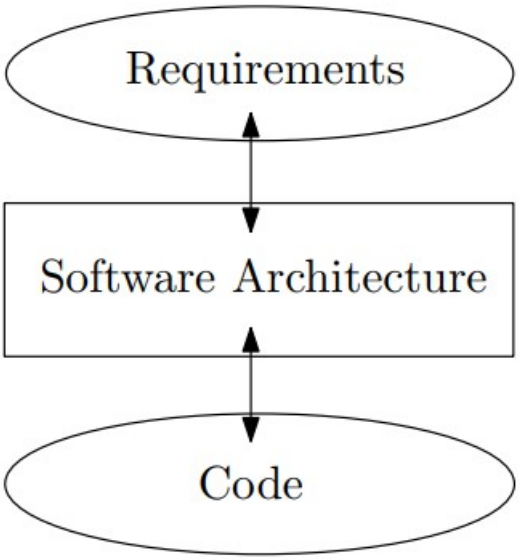


Figure 2.1: Software Architecture as bridge

For example, software architecture and design terms are used interchangeably. However,

to prevent mistakes, we need to establish some boundaries between them and differentiate each from another. Therefore, there is one section for each concept describing its definitions. Further on, we need to reason about the differences.

Similarly, architectural styles and patterns are two other terms that must be clarified. Understanding and defining these concepts are part of the complexity of being a software architect. (HGRACA, 2017) reminds us that *patterns and styles are not mutually exclusive, they are complementary, and they all can teach us something, although, as usual, they should be used only when needed*.

2.2 Definitions

Software architecture definition as own software evolves through time. Several of these definitions are made through analogies to existing disciplines like classical building architectural discipline.

Since 1968, the term Software Crisis was coined (FELDHAUSEN, 2020), software specialists and engineers have been studying software development state. In the 1970s, Brooks wrote about the importance of architecture (BROOKS, 1986), which leveraged the search for practitioners and researchers to look at this field of knowledge.

SEI compiled several of these definitions and classified them into three different domains: modern, classical, and bibliographical (SEI, 2010). Each definition of Software Architecture focuses on the software's internal organization and external dependencies. GACEK et al. (1995) and BASS; KAZMAN; CLEMENTS (2012) definitions are vastly quoted on literature. These definitions can be seen below:

According to GACEK et al. (1995) a software system architecture comprises:

- A collection of software and system components, connections, and constraints.
- A collection of system stakeholders' need statements.
- A rationale that demonstrates that the components, connections, and constraints define a system that, if implemented, would satisfy the collection of system stakeholders' need statements.

BASS; KAZMAN; CLEMENTS (2012) defines software architecture as *"The set of structures needed to reason about the system, which comprises software elements, relations among them, and properties of both."* . This definition also highlights software organization and its interaction with other components.

Besides these two definitions, we would like to highlight another two that mention the importance of rationale and abstractions in decision-making.

Ralph Johnson, quoted by FOWLER (2003), says that *"architecture is the decisions that you wish you could get right early in a project but that you are not necessarily more likely to get them right than any other"*.

As elegantly defined by FIELDING (2000), software architecture provides high-level abstractions in the form of coarse-grained processing, connecting, and data elements, their interfaces, and their configurations.

2.3 Software Design

Software Design and Software Architecture are two terms that are used interchangeably. Even in the literature, software design does not have an unambiguous definition. RALPH; WAND (2009) suggests that the conceptual model of design should be categorized as seen in figure 2.2.

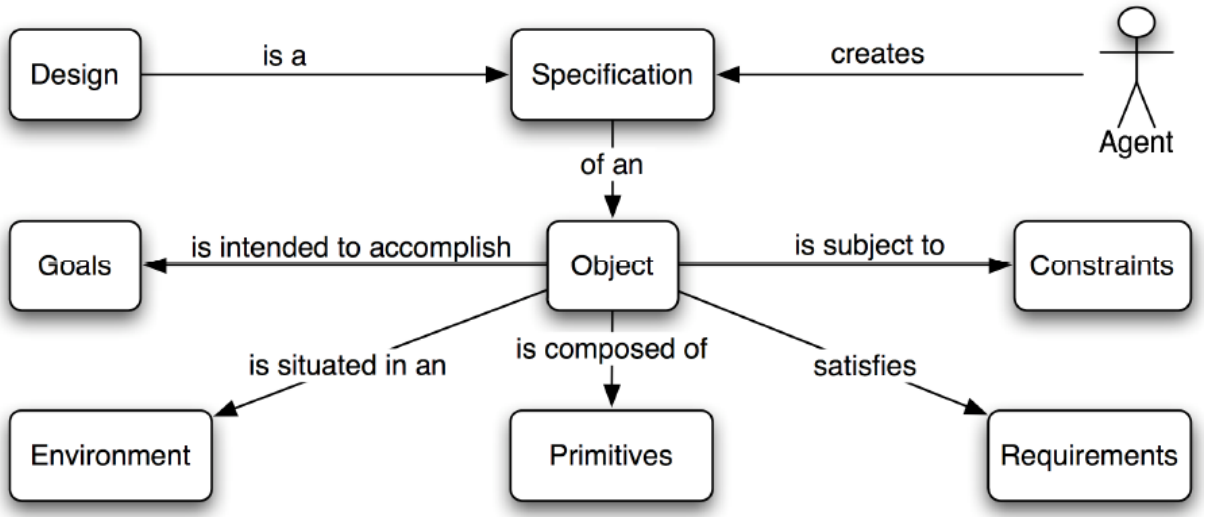


Figure 2.2: Conceptual Model of Design. Source: (RALPH; WAND, 2009).

LEHMAN; STENNING; TURSKI (1984) depicts software design as *"from 'topmost' specification down to final implementation - is viewed as a chain of uniform steps, each step being a transformation between two linguistic levels"*. This definition describes software design as a set of actions enabling software practitioners to develop the software.

For EDEN; HIRSHFELD; KAZMAN (2006), the science of software design is concerned with the description of programs. This definition seems elusive, but they describe several classes of abstraction concerning software: strategic statements, tactical statements, and implementation statements. Each of these statements describes constraints on the structure of programs.

On the other hand, TAYLOR; MEDVIDOVIC; DASHOFY (2008) explains that software design is concerned with both the external, functional aspects of a software product and its internal constituents. This definition involves external and internal elements of the software itself, and it needs to be clarified between the terms.

For CONTENT TEAM (2022):

> Software design is one of the initial phases of the software development life cycle. In this phase, you analyze and identify the methods that your developers will use.

Additionally, server-side or client-side will be built according to stakeholder and customer requirements.

As we can see, software design definition shares several responsibilities with software architecture. The system organization and how each component interacts are some examples of how closely related these two areas are. In the next section, we will make distinctions between each term. However, some elements differentiate one from another.

2.3.1 Software Architecture x Software Design

According to EDEN; HIRSHFELD; KAZMAN (2006), the terms' architecture' and 'design' are used in overlapping ways by the research and the industrial communities. This kind of misunderstanding appears with design and architectural decisions, according to BOER et al. (2007). However, comparing the overall aspect of previous definitions, we can consider both concepts as generalizations on how software is organized and how its components work together.

That means that depending on how elaborated the architect/software engineer's description is, the distinction between architecture and design becomes fuzzy.

We agree with CONTENT TEAM (2022) distinction between the terms where while the software architecture identifies the components and elements that need to be included in the software, the software design focuses on how the software will be built. This way, we can define software architecture and software design as two separate parts of the software development process, and they depend on each other for success.

In short, software architecture focuses on software elements (components, connectors, configuration, constraints) and their organization (what will be built). Software design focuses on software behavior, e.g., synchronous or asynchronous communication, relational or non-relational database, and oriented-object or functional paradigm (how things will be done).

2.4 Architectural Styles

Architectural style and pattern definitions are other good examples of closely related concepts used without proper characterization. According to PERRY; WOLF (1992), an architectural style is less constrained and less complete than a specific architecture.

For GARLAN; SHAW (1993), an architectural style *"... defines a family of such systems in terms of a pattern of structural organization. More specifically, an architectural style determines the vocabulary of components and connectors that can be used in instances of that style, together with a set of constraints on how they can be combined"*.

ALLEN (1995) describes the architectural style as a collection of patterns and idioms that are effective for the domain in which designers are working.

TAYLOR; MEDVIDOVIC; DASHOFY (2008) says that an architectural style can be considered a set of constraints on the architectural elements and their interactions, thus defining a set or family of architectures that satisfy them.

Despite being reasonably close to software architecture, an architectural style is not architecture, but it conveys a helpful image of the system and imposes constraints on architecture (SANTOS et al., 2020).

Architectural styles are named collections of constraints on configurations of architectural elements and are believed to bring economies of scale in applying software architecture techniques to software development (MEHTA; MEDVIDOVIC, 2003).

2.4.1 List of architectural styles

During the history of software engineering, some attempts have been made to classify architectural styles. Garlan and Shaw GARLAN; SHAW (1993) proposed the following classification:

1. Pipes and Filters
2. Data Abstraction and Object-Oriented Organization
3. Event-based, Implicit Invocation
4. Layered Systems
5. Repositories
6. Table Driven Interpreters
7. Other Familiar Architectures
8. Heterogeneous Architectures

In a similar fashion ZHU (2005) described some architectural styles in table 2.1:

Architectural Styles	
Data Flow	1. General Data Flow 2. Pipe-and-filter 3. Batch sequential processing
Independent Components	4. General Independent Components 5. Event-based implicit invocation systems 6. Communicating processes
Call and Return	7. General call and return 8. Layered systems 9. Data abstraction: the abstract data type and object-oriented
Data Centered	
Virtual Machine	

Table 2.1: Architectural Styles classified by Zhu (2005)

SHARMA; KUMAR; AGARWAL (2015) proposed a complete categorization of all existing architectural styles, which is given in Table 2.2.

Application Type	Architectural Style
Shared Memory	1. Black Board 2. Data centric 3. Rule Based
Distributed Systems	1. Client-Server 2. Space-based architecture 3. Peer to peer 4. Shared nothing architecture 5. Broker 6. Representational state transfer 7. Service-oriented
Messaging	1. Event-Driven 2. Asynchronous messaging 3. Publish-subscribe
Structure	1. Component-based 2. Pipes and filters 3. Monolithic application based 4. Layered
Adaptable System	1. Plug-ins 2. Reflection 3. Microkernel
Modern System	1. Architecture for Grid Computing 2. Multi-tenancy Architecture 3. Architecture for Big-Data

Table 2.2: List of architectural styles proposed by Sharm et al. (2015)

2.5 Architectural Patterns

Usually, when we think about a pattern, we tend to reason about something that repeats itself over time. As defined by GAMMA et al. (1994), a pattern addresses a recurring design problem that arises in specific design situations and presents a solution to it.

According to BUSCHMANN et al. (1996), architectural pattern is defined as *"...expressing a fundamental structural organization schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them"*.

On the same note, for DHADUK (2020) an architectural pattern can be called

> ... an outline that allows you to express and define a structural schema for all kinds of software systems. It's a reusable solution that provides a predefined set of subsystems, roles, and responsibilities, including the rules and roadmap for defining relationships among them. It helps you address various software engineering concerns such as performance limitations, high availability, minimizing business risk, etc.

2.5.1 Classification of architectural patterns

In 2017, MALLAWAARACHCHI (2017) published a web article describing ten common architectural patterns. He analyzed the pros and cons of each one of them and compiled the following list.

1. Layered pattern
2. Client-server pattern
3. Master-slave pattern
4. Pipe-filter pattern
5. Broker pattern
6. Peer-to-peer pattern
7. Event-bus pattern
8. Model-view-controller pattern
9. Blackboard pattern
10. Interpreter pattern

The same list was discussed by TECH (2021), highlighting several reasons enterprises are adopting these patterns nowadays.

Following a similar classification, WALKER (2022) described 14 architectural patterns to increase efficiency, productivity, and speed, optimize development costs, improve planning, and more.

1. Circuit Breaker
2. Client-Server
3. Command Query Responsibility Segregation (CQRS)
4. Controller Responder
5. Event Sourcing
6. Layered
7. Microservices
8. Model View Controller (MVC)
9. Publisher-Subscriber (Pub-Sub)
10. Saga
11. Sharding
12. Static Content Hosting
13. Strangler
14. Throttling

Architectural patterns share a similarity with design patterns since they are a solution to a recurring problem. However, as said by WILLIAMS (2022), design patterns represent a way to structure classes to build the best internal structure, while architectural patterns define a solution for a variety of quality attributes and contemplate multiple components in a software system in a broader scope.

2.6 Architectural Styles x Architectural Patterns

XU et al. (2006) understood the relationship between architectural style and architectural pattern as *"... architecture style to refer to the types of constructs that are used in the designed architecture, while the architectural pattern proposed here concentrates more on modeling non-functional requirements in architecture level and linking them with the designed architecture."* .

An essential part of an architecture pattern is to focus on the problem and context, solving the problem in that context. An architecture style focuses on the architecture approach, with more lightweight guidance on when a particular style may or may not be helpful (PAULA; FALVOJR, 2016).

The table 2.3 shows a comparison table between architectural styles and patterns.

Category	**Architectural Patterns (Shaw & Garlan, 1996)**	**Architectural Styles (MALLAWAARACHCHI, 2017)**
Dataflow Systems	Batch Sequential Pipes and Filters	Pipe-Filter Pattern
Call-and-Return Systems	Main Program and Subroutine OO Systems Hierarchical Layers	Model-View-Controller Layered
Independent Components	Communicating Processes Event Systems	Event-Bus
Virtual Machines	Interpreters	Interpreter
Repositories	Databases Hypertext Systems Blackboards	Blackboard
Client-Server		Client-Server
Master-Slave		Master-Slave
Broker		Broker
Peer-to-Peer		Peer-to-Peer

Table 2.3: List of architectural styles x patterns

In short, as summarized by MONROE et al. (1997) *"... architectures, architectural styles, objects, and design patterns capture complementary aspects of software design. Although the issues and aspects of software design addressed by these four approaches overlap somewhat, none completely subsumes the other"*.

2.7 Software Architecture and Decision-Making

Moving from the broader context of software architecture, it is essential to delve into the specific process of decision-making, which plays a crucial role in this field. As explained by

TRAN et al. (2014), from the strategic view of making an AD, it is necessary to have a long-term impact on these decisions, for example, future operations and maintenance efforts.

Considering that the next chapter will be about decision-making, we want to highlight how this architectural decision process is crucial from the software's lifetime. According to ORLOV; VISHNYAKOV (2017), software architectural decisions significantly impact the software development process and the quality of developed software systems.

There is the seminal case of Twitter's early architecture, in which most of the information was directly stored and queried from the database. This choice reflected poorly on the quality of their service since failures happened frequently. It was necessary to segregate the tweets timelines as explained by SAM (2022). This decision significantly decreased workload, allowing their service to scale better.

2.8 Architectural Evolution and New Challenges

Certain architectural styles and patterns have stood the test of time and continue to be relevant nowadays, for example, layered architecture. The schematics of this kind of architecture highlight the importance of organizing and isolating code components in each layer, which reinforces the separation of concerns and loose coupling. Onion architecture is one kind of layered architecture built on a domain model in which layers are connected through interfaces (KAPOOR, 2022).

COCKBURN (2005) proposed hexagonal architecture that is based on layered patterns. This architecture is structured in a way that all of these external systems can be separated from the core app/business logic and made to communicate with it in a technology-agnostic way (SENECKI; GOIK, 2023). Therefore, considering the mutual aspects of these architectures, we can perceive some common characteristics, such as separation of concerns and modularity.

Due to the popularization of cloud computing and containerization, microservices architecture is another popular approach for developing distributed systems. This adoption become so aggressive that according to PLANTROU (2022), the microservices market is predicted to triple between 2020 and 2026. However, growth leads to new issues like dealing with the complexity of managing communication and data consistency.

Dealing with this complexity is no trivial matter. For instance, observability is one attempt to understand the inner state of software through external outputs. Therefore, LI et al. (2022) believes that observability is an essential requirement for microservices systems.

According to CUMMINGS (2023), software observability requires generating and collecting actionable telemetry coupled with analysis and visualization for understanding. It is necessary to prepare critical elements to achieve that degree of observability:

1. Software's instrumentation to generate application telemetry;

2. Visibility of software infrastructure for supporting telemetry;

3. Collecting and processing the telemetry data;
4. Storage of telemetry data for efficiency, performance, and historical comparison;
5. Analysis and visualization of telemetry data for understanding.

Considering the patterns and styles presented in this study, we can perceive that technical evolution leads to new problems and new patterns to solve these problems. Circuit break, for example, can help maintain system stability by detecting failures and preventing the failure from overwhelming the system. Throttling, on the other hand, can prevent services from being overwhelmed by too many requests and ensure fair resource usage.

Each pattern offers unique benefits in microservices, addressing specific challenges such as service decoupling, data consistency, fault tolerance, and scalability. Their practical implementation can significantly enhance the efficiency, productivity, and speed of development in a microservices-based system.

2.9 Summary of this chapter

This chapter provided the necessary background to comprehend the overall aspect of software architecture and its definition throughout history. Then, some concepts that are closely related to it. Software design, architectural styles, and patterns are characterized to avoid confusion.

After understanding how complex the very definition of software architecture is and its importance, in the next chapter, we can study decision-making and its relationship with software architecture, underlining how decisions are made, how documentation takes place, and their weight on software's lifecycle and maintenance.

3

DECISION-MAKING: In the Context of Software Architecture

Software engineers and architects often falter when considering optimal software design due to its inherent complexity. Making the right decisions upfront is a difficult task. There are several requirements, constraints, and challenges to address simultaneously. Inappropriate design decisions are often hard to reverse, leading to high costs and poor software product quality.

Decision-Making (DM) theory has been studied for centuries by philosophers, mathematicians, economists, and statisticians. The process of making a decision is the fruit of several cognitive steps. One of the core steps is problem structuring. In this step, the decision-maker specifies the possible actions, the state of the world relevant to the decision, and the outcomes contingent on both the chosen action and the states of the world that can occur (SLOVIC; LICHTENSTEIN; FISCHHOFF, 1988).

There are several attempts to understand decision-making decisions in software development. Research in this area includes multi-criteria methods (VASSILEV; GENOVA; VASSILEVA, 2005), surveys of current decision-making techniques (FALESSI et al., 2011), studies on group decision-making (REKHAV; MUCCINI, 2014), and more.

TANG; KAZMAN (2021) proposed a more agnostic approach that involves nine principles related to design reasoning. This method allows us to check if the design decision was well made and if the findings are grounded on reasonable parameters, such as timetable, design constraints, risks, etc.

This chapter presents an overview of the decision-making literature, describing various related topics, mapping principles that IT experts should follow during the process of making a decision, reasoning about this new field of decision-making called Data-Driven Decision Making (DDDM), and proposes a state of art framework to achieve better architectural design decisions.

3.1 Background on Decision-Making

Decision-making literature is vast and has been studied for many decades. According to VLIET; TANG (2016), decision-making studies range from four types of research: decision-making and argumentation, design rationale, design decision-making, and group decision-making.

Decision-making primarily involves the search for optimal solutions in various aspects of

our lives. However, as noted by PRETORIUS (2019), human factors underlying decision-making are relatively complex and challenging to grasp.

As described by SIMON (1955), humans tend not to make perfectly rational decisions. Therefore, each decision related to software architecture carries tremendous significance and can impact the project's success. Moreover, capturing design decisions during the project lifecycle is a great challenge, according to (LEE; KRUCHTEN, 2007).

The architectural decision-making process is subjected to several influence factors (GROHER; WEINREICH, 2015). Some elements are directly related to enterprise structures (agile or traditional), and others are related to the people responsible for the decision. However, we can benefit from another branch of study: the decision-making process in an uncertain scenario.

In the early stages of software development, uncertainty is a common occurrence. Requirements change, technical staff may be reallocated, and stakeholders' needs shift. These factors contribute to a fast-paced environment increasing uncertainty.

As defined by MARCHAU et al. (2019) *uncertainty refers to the gap between available knowledge and the knowledge decision-makers would need in order to make the best policy choice*.

While various research domains inform us about the complex factors affecting decision-making, there is an essential gap concerning how these theories apply to software architecture. The next chapter will bridge this gap by investigating the unique demands that architectural decisions impose on the decision-making process. Specifically, we will delve into bounded rationality and explore the role of intuitive and naturalistic decision-making in software architecture. This examination is particularly pertinent given that software architecture involves complexities and uncertainties, from rapidly changing requirements to the reallocation of technical staff.

3.2 Software Architecture and Decision Making

Having discussed the general landscape of decision-making, we now shift our focus to the role of decision-making within the context of software architecture. It is essential to recognize that software architectural decisions are governed by their own rules, influenced by specific constraints, and subjected to their kinds of uncertainties.

Within this framework, we will interrogate how bounded rationality and intuitive and naturalistic decision-making manifest in architectural choices. By employing these theories, we aim to elucidate how software practitioners successfully navigate the maze of software development constraints and uncertainties.

Many argue that decision-making is not rational and that people stop reasoning as soon as a satisfactory solution is found (VLIET; TANG, 2016). This behavior is referred to as 'bounded rationality' by SIMON (1996). This means that individuals' rationality is limited by the information they have, the cognitive limitations of their minds, and the finite amount of time they have to make a decision.

According to the literature, there are two types of decision-making: naturalistic and intuitive (KLEIN, 2009). Each one of them is more suitable depending on the context. KAHNEMAN (2011) uses the terms System 1 and System 2. System 1 (the intuitive) is fast, instinctive, emotional, and evolutionary, very old. System 2 (the rational) is slower, more deliberative, logical, evolutionary, and more recent.

3.2.1 Intuitive Thinking

Betsch and Roth, in the book INTERNATIONAL HANDBOOK OF THINKING AND REASONING (2017), said that the concept of intuition is a risky endeavor. This happens because psychologists agree that intuition is a phenomenon of paramount importance characterized by distinct properties but disagree about what those properties are.

Using intuition or gut feeling is firmly rooted in human decision history. However, intuition is not magic. According to MATZLER; BAILOM; MOORADIAN (2007)

> Intuition is a highly complex and highly developed form of reasoning that is based on years of experience and learning and facts, patterns, concepts, procedures, and abstractions stored in one's head.

Therefore, when using this form of cognition, software practitioners rely on various elements, such as previous experiences, Quality Attributes (QA), or Non Functional Requirements (NFR), to achieve success in software development.

3.2.2 Naturalistic Thinking

On the other hand, naturalistic thinking is an approach where decisions are made using rationality and principles of optimal performance. This approach resulted in the creation of Naturalistic Decision Making (NDM) research subject. As described by KLEIN (2008), its origins are around 1989, and at the beginning, researchers were not concerned with formal decision-making models; they began by conducting field research to discover the strategies people used.

Such statistical and mathematical strategies are done to ensure the decision-making process is based on objective criteria instead of subjective ones. However, it is necessary to structure the problem to work in such a way. According to ZANNIER; CHIASSON; MAURER (2007), the naturalistic approach is more commonly used whenever a software design problem is more structured. The NDM focus on field settings and its interest in complex conditions provide insights for practitioners of human factors in improving performance (KLEIN, 2008).

3.2.3 Intuitive and Naturalistic Thinking

KAHNEMAN (2003) classified the architecture of cognition in two systems. Furthermore, the definition of each one of them is depicted as follows:

> *The operations of System 1 are fast, automatic, effortless, associative, and often emotionally charged; they are also governed by habit, and are therefore difficult to control or modify. The operations of System 2 are slower, serial, effortful, and deliberately controlled; they are also relatively flexible and potentially rule-governed.*

Using this same classification PHILLIPS et al. (2016) wrote about thinking styles and their relationship with decision making. The characterization of the cognitive process using a dual process theory can be seen as follows:

> *Dual-process theories of cognition attribute certain variations in decision-making behavior to the relative involvement of two distinct types of information processing: (a) intuitive processing that is automatic, fast, preconscious, associative, autonomous and does not require working memory, and (b) reflective processing that is relatively slow, effortful, conscious, analytical, rule-based, and requires working memory.*

When we relate this theory with software architecture, according to PRETORIUS et al. (2018), naturalistic decision-making is used to select a satisfactory design solution rather than an optimal one.

In conformity with the literature, both systems of cognitive thinking are part of the human decision-making process. Some researchers, such as SCHRIEK et al. (2016), claim that the naturalistic approach is more suitable for software architecture design, although both systems are used in practice.

3.2.4 Data-Driven Decision Making

A type of naturalistic decision-making is gaining much traction due to the nature of complex environments and data availability. It is called Data-Driven Decision Making (DDDM). It can be defined as:

> *... the process of using data to inform your decision-making process and validate a course of action before committing to it (STOBIERSKI, 2019).*

Data lakes, data warehouses, and business intelligence are good examples of how data is processed to provide more insights to decision-makers. The need to look to the past before deciding something for the future can provide many benefits, such as cost-saving, proactive, and grounded decisions, which highlights the importance of this approach.

Despite the importance of reflecting on these decision-making systems, another essential element in AD is the act of recording these decisions. We will discuss more about this topic in section 3.3.

3.3 Documenting Architectural Decisions

In the first chapter of the study, we explored the concept of software architecture and how it is represented. This section will briefly discuss architectural decision documentation and how to document architectural decisions.

The recognition of the need for documenting as described by TANG et al. (2006) highlights how necessary software documentation is. Failing to record the design rationale during architectural decisions can lead to design erosion or knowledge vaporization, as noted by (GROHER; WEINREICH, 2015). Moreover, as described by HGRACA (2019), when we need to explain to someone how the application works, documentation is needed.

KOPP; ARMBRUSTER; ZIMMERMANN (2018) research revealed that developers perceive architectural decision-capturing practices and tools as chronophages (time wasters). With no integration with their toolchain, Integrated Development Environment (IDE), such devices have a negative impact on productivity, quality, and motivation.

Therefore, several attempts have been made to document architectural decisions more organically. One of them is the Markdown Architectural Decision Records (MADR). Markdown, a text format, enables standard version control systems like Git. Another popular alternative for recording architectural decisions is the Y's statements.

3.3.1 Y's Statements

Y's Statements are a light template for architectural decision capturing. It was created by ZIMMERMANN (2020), who worked as a consulting IT architect at IBM, following the ARC 100 template.

As appointed by SOC (2020) ADs, answer "why?" questions about the design and justify why an option is chosen. In its essence, it is a representation of an architectural record composed of six sections:

- context: functional requirement (story, use case) or arch. component,
- facing: non-functional requirement, for instance, a desired quality,
- we decided: decision outcome (arguably the most important part),
- and neglected alternatives not chosen (not to be forgotten!),
- to achieve: benefits, the full or partial satisfaction of requirement(s),
- accepting that: drawbacks and other consequences, for instance, impact on different properties/context and effort/cost (both short term and long term).

Y's Statements are visualized with the letter 'Y,' pronounced like the English word 'why.' This naming convention is depicted in Figure 3.1.

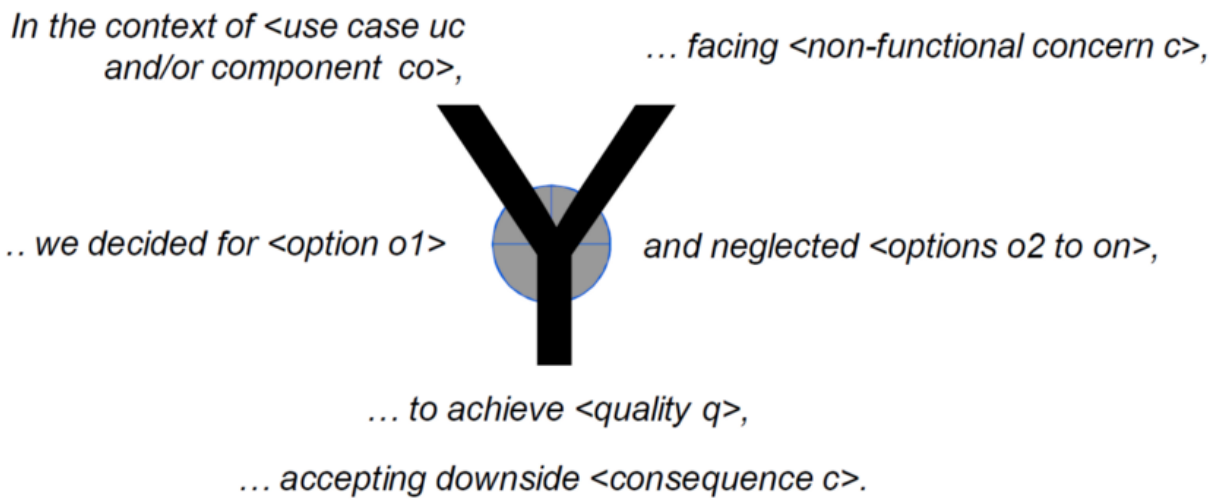


Figure 3.1: Y's Statements Example - SOC (2020)

3.3.2 Architectural Decision Records

An Architectural Decision Record (ADR) is a document describing a team's choice about a significant aspect of the software architecture they are planning to build (AWS, 2022). They were first introduced by Michael Nygard's blog post NYGARD (2011).

An ADR usually consists of a short text file describing a specific architecture decision. It can be written in plain text, AsciiDoc/Markdown format, or a wiki page template. In figure 3.2, we can see an example of ADR template proposed by Michael Nygard.

Having explained these possibilities of documenting AD, it is necessary to explore some intricacies in making decisions. TANG; KAZMAN (2021) proposed a systematic approach to software design decision-making, and in section 3.4, we will unveil the relationship between these principles and software architecture.

3.4 Decision-Making Principles of Software Design

TANG; KAZMAN (2021) outlined nine principles that should be used in order to avoid flawed decisions. He called it the nine principles of DM. They are:

(P1) Use Facts: Instead of relying on assumptions and half-guessed solutions, keep in mind that facts and evidence are the foundations of logical decisions.

(P2) Check Assumptions: Sometimes, we cannot have all the facts, and assumptions have to be made. However, checking and validating those assumptions can create solid evidence to support our decisions.

(P3) Explore Contexts: Contexts are conditions that influence software decisions. For example, project budget, development resources, legal obligations, industry norms, user expectations, and past decisions. Exploring contextual factors can broaden our design considerations.

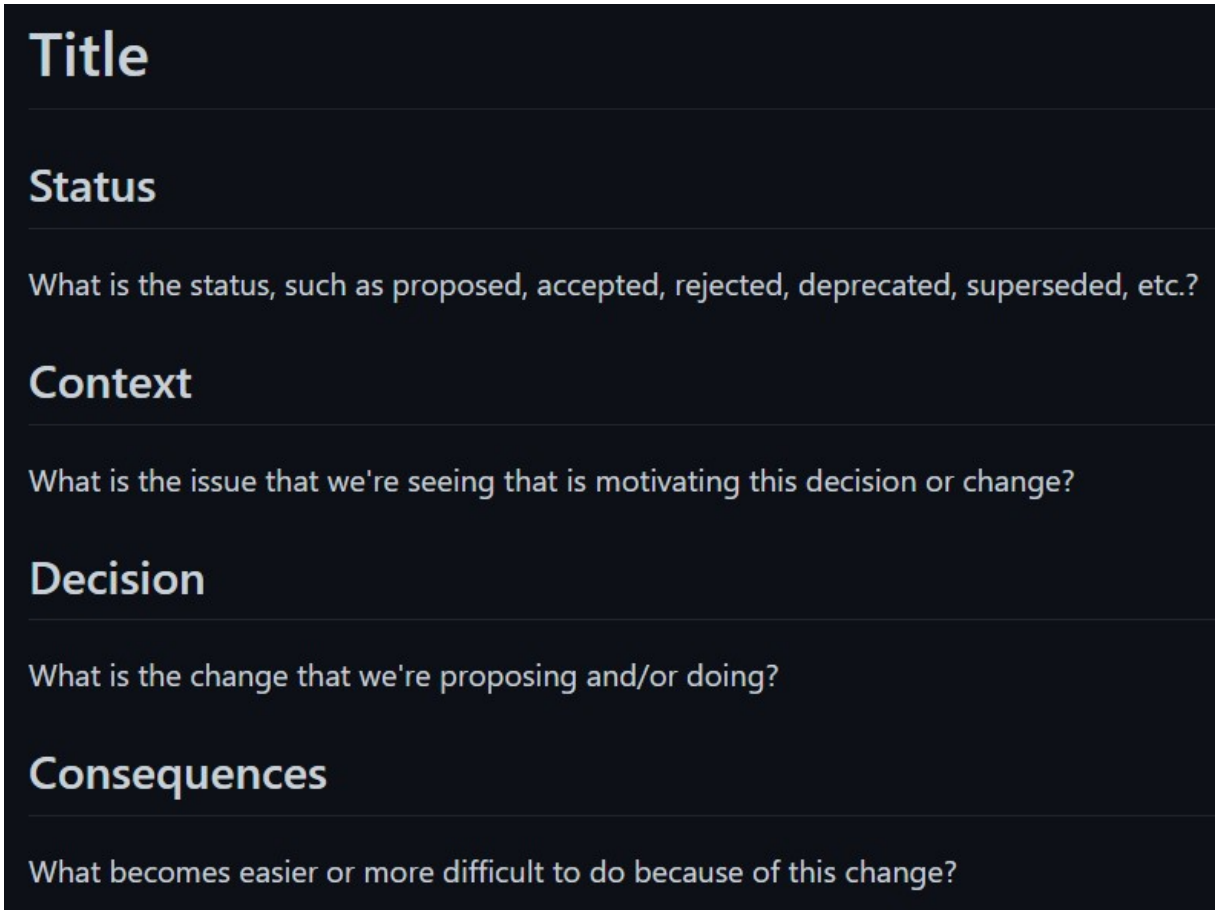


Figure 3.2: ADR Example

(P4) Anticipate Risks: Despite being a challenging task, estimating and planning for risks allow software designers to have an alternative plan if something goes wrong.

(P5) Assign Priorities: Prioritization allows the most important things to be done in the correct order without a hitch.

(P6) Define the Time Horizon: The decisions taken during the project must be contextualized in a time horizon that allows for their reassessment, considering the pros and cons at the time of each decision. In addition, it guides the time window in which decisions need to be made.

(P7) Generate Multiple Solution Options: Generating multiple solution options allows practitioners to avoid a common pitfall called anchoring bias. It helps a designer broaden choices and stimulate creativity.

(P8) Design Around Constraints: Constraints are limitations that set the boundaries of what a solution cannot do and have to abide by. Sometimes, it leads to novel solutions, relaxing parameters, and manipulating the context.

(P9) Weigh Pros and Cons: Trade-off evaluation is a way to help the designers to reason

about the relative benefits and drawbacks of adopting a solution.

Each of these principles highlights concerns underlying the decision-making process in the context of software architecture. All these principles can improve the probability of success when making these decisions. In the literature, several decision-making techniques are available for selecting architectural alternatives (FALESSI et al., 2011).

In the literature, decision-making pathways can be classified into two forms. The first is more coarse-grained and generic, providing a systematic and organized way to make decisions (decision-making frameworks). The second is more fine-grained, oriented to specific methods and approaches used to analyze, evaluate, and make decisions (decision-making techniques).

The following section will provide more explanation about these alternatives and how they can be related to software architecture.

3.4.1 Decision-Making Techniques

Methods for selecting software architecture alternatives usually are born from functional and quality attributes, a.k.a. NFR according to FALESSI et al. (2011). This author affirms that software architects need a reliable and rigorous process for selecting these alternatives and ensuring that decisions mitigate risks and maximize profit. While this statement has a good point, there are several concerns to address that experts tend to rely on familiar places or gut feelings.

However, attempts to reach a better balance between rational and naturalistic approaches have been made. SUHR (2000) describes one of the most popular methods of decision-making called Choosing By Advantages (CBA). This method is prevalent in software engineering, emphasizing the positive aspects of adopting a specific approach. In figure 3.3, we can see the common steps of using this method.

CBA is composed by the following stages according to SUHR (2000):

1. Identifying Alternatives: Begins with identifying different alternatives for a decision.
2. Listing Attributes: Lists the attributes of each alternative, representing the characteristics or features that will be compared.
3. Determining Advantages: Identifies the advantages of each attribute, defined as any aspect of an attribute that makes it preferable to the other attributes.
4. Assigning Importance Weights: Importance weights are assigned to each advantage to reflect its significance in decision-making.
5. Calculating Decision Scores: The decision scores for each alternative are calculated by summing the importance weights of that alternative's advantages.
6. Selecting the Best Alternative: The alternative with the highest decision score is selected.

This method has been widely used in industries such as Architecture Engineering and Construction (AEC) (ARROYO; TOMMELEIN; BALLARD, 2012) and construction

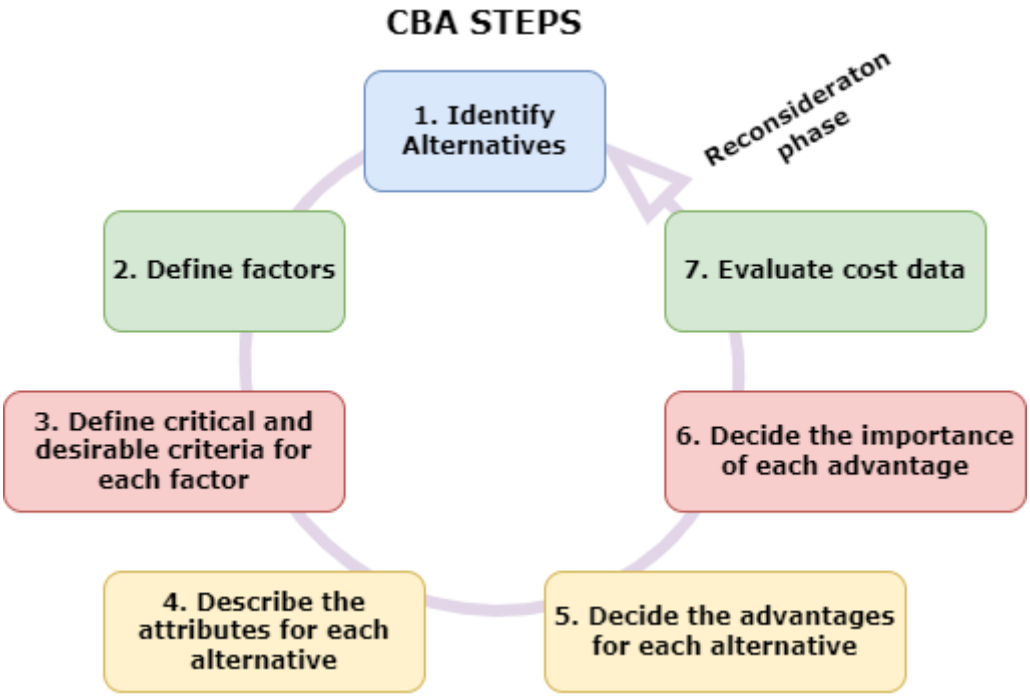


Figure 3.3: CBA Steps

(KARAKHAN; GAMBATESE; RAJENDRAN, 2016), and it has been shown to be more effective than other methods such as the Analytic Hierarchy Process (AHP) or Weighting Rating and Calculating (WRC) (ARROYO; TOMMELEIN; BALLARD, 2014).

Another method accomplished method for decision-making is Multi-Criteria Decision Analysis (MCDA). This method is used when conflicting criteria need to be considered simultaneously, and it provides a structured and systematic way to analyze complex decision problems (TRIANTAPHYLLOU; BAIG, 2005). There are still several decision-making techniques for improving assertiveness in architectural decisions. Table 3.1 describes several existing methods.

These techniques are relevant to AD since most of these techniques utilize heuristic methods that provide an especially powerful problem-solving and decision-making tool for humans who are unassisted by any computer other than their minds, hence must make radical simplifications to find even approximate solutions (SIMON, 1996).

Considering the discussion about DM techniques, it is crucial to understand more about DM frameworks. In the next section, we will discuss more about this topic.

3.4.2 Decision-Making Frameworks

A structured decision-making framework involves identifying and evaluating alternatives based on criteria or factors. When we say the term *framework*, we would like to highlight the definition of JABAREEN (2009) that says that conceptual framework as a network, or "a plane," of interlinked concepts that together provide a comprehensive understanding of a phenomenon or phenomena.

Table 3.1: Decision-Making Techniques, Application, and Authors

Decision-Making Technique	**Application**	**Author(s)**
ANP (Analytic Network Process)	Evaluating complex decision networks with interdependent elements.	Thomas Saaty (1980)
ATAM [1]	Evaluating software architecture trade-offs.	Bass et al. (2003)
Business Process Modeling (BPM)	Evaluating workflow options to improve efficiency in a manufacturing process.	Frederick Taylor (1991)
CBAM [2]	Evaluating adaptive systems with changing contexts.	Kazman (2001)
CBAM [3]	Incorporating cognitive biases in decision-making processes.	Moore (2003)
Choosing By Advantages (CBA)	Identifying the best solution based on a list of advantages and disadvantages.	SUHR (2000)
Cost-Benefit Analysis (CBA)	Project evaluation, public policy analysis, and investment decisions.	Jules Dupuit (1884), Alfred Marshall (1890)
Decision Trees	Evaluating the best option for a marketing campaign based on possible outcomes.	Abraham Wald (1945), Stuart and Hubert Dreyfus (1980)
Delphi Method	Forecasting future trends based on expert opinions and iterative rounds of feedback.	Norman Dalkey and Olaf Helmer (1950)
Game Theory	Analyzing competitive strategies in business negotiations.	VON NEUMANN; MORGENSTERN (2007)
Pareto Analysis	Identifying the most significant factors contributing to a problem.	Vilfredo Pareto (1896)
Real Options Analysis (ROA)	Evaluating the value of investment options with flexibility in decision timing.	MYERS (1977)
Six Thinking Hats	Exploring different perspectives when choosing a vacation destination.	BONO (1999)
SWOT Analysis	Assessing Strengths, weaknesses, opportunities, and Threats for Decision-making	STEWART; BENEPE; MITCHELL (1965)
TOPSIS [4]	Ranking alternatives based on their similarity to the ideal solution	HWANG; YOON (1981)

The framework typically includes a process for gathering information, analyzing options, and selecting the best course of action based on the available data and the decision-makers' goals and values. Decision-making frameworks are used in various contexts, including business, government, and personal decision-making. They are designed to help individuals and organizations make more informed and effective decisions by providing a systematic and transparent approach to the decision-making process.

Decision-Making Framework	**Author**	**Date**
Analytic Hierarchy Process (AHP)	Thomas L. Saaty	1971
Attribute-Driven Design (ADD)	Ralph E. Johnson	1994
Architecture-Level Modifiability Analysis (ALMA)	Len Bass, Paul C. Clements, Rick Kazman	2012
Architecture Tradeoff Analysis Method (ATAM)	SEI	2000
Cost-Benefit Analysis Method (CBAM)	SEI	2003
Knowledge Architecture (KA)	Richard Gronback	2009
Systematic Decision-making Framework	Nitin Upadhyay	2016

Table 3.2: Decision-Making Frameworks

Considering all decision-making frameworks, some clarification is due to highlight the differences between DM frameworks and techniques.

3.4.3 Decision-making Framework x Decision-making Techniques

Decision-making techniques and frameworks are tools used to facilitate the decision-making process, but they serve slightly different purposes and have distinct characteristics. Decision-making techniques refer to specific methods, strategies, or approaches that individuals or groups use to analyze options, evaluate alternatives, and choose the best course of action (TEAM, 2022).

These techniques are valuable when addressing specific decision-making challenges, such as comparing alternatives, evaluating risks, or assessing pros and cons. They help break down complex decisions into manageable steps.

Decision-making frameworks are broader structures or models that guide decision-makers in thinking systematically and holistically about their decisions. These frameworks provide a high-level approach to making decisions and help ensure that important aspects are considered. Decision-making frameworks often integrate multiple techniques and principles.

Frameworks are more appropriate than techniques when dealing with more significant, complex decisions requiring a systematic approach and considering multiple factors. They help decision-makers think broadly and strategically about their choices.

3.4.3.1 Limitations of Decision-making Framework and Decision-making Techniques

To reach a better degree of understanding is necessary to provide more information about limitations of each approach.

Limitations of Decision-Making Techniques:

1. Limited Scope: Many decision-making techniques are designed for specific types of decisions and may not be applicable or effective in different contexts. For instance, a cost-benefit analysis might be less useful in decisions where qualitative factors are more important than quantitative ones.

2. Dependence on Quality of Data: Techniques like statistical analysis rely heavily on the quality and completeness of data. Poor data can lead to inaccurate or misleading results.

3. Over-Simplification: Some techniques might oversimplify complex decisions, ignoring nuanced factors that don't fit neatly into the chosen method.

4. Cognitive Biases: Techniques are often applied subjectively, and the decision-makers' biases can influence the outcome, especially in techniques that involve significant human judgment.

5. Time and Resource Constraints: Certain techniques can be resource-intensive and time-consuming, making them impractical for quick or low-stakes decisions.

Limitations of Decision-Making Frameworks:

1. Rigidity: Some frameworks can be too rigid or prescriptive, not allowing enough flexibility for unique or unforeseen circumstances in decision-making.

2. Complexity: While aiming for comprehensiveness, some frameworks can become overly complex, making them difficult to understand and apply, especially for those without extensive experience or expertise.

3. Underestimation of Human Factors: Frameworks may not adequately account for human emotions, cultural differences, or ethical considerations, which can be critical in certain decision-making scenarios.

4. Implementation Challenges: The successful application of a framework often requires significant organizational change, buy-in from multiple stakeholders, and consistent application, which can be challenging to achieve.

5. One-Size-Fits-All Approach: Frameworks may adopt a generalized approach that may not suit all decision-making contexts, particularly those that are highly specialized or idiosyncratic.

3.5 Summary of this chapter

In this chapter, we discussed the decision-making process, its background, and its importance in the context of software architecture. We briefly talked about intuitive and naturalistic thinking and its role in reasoning. Further, we discussed the importance of documenting architectural decisions and two popular forms of doing that: Y's statements and ADR. At last, we explored some decision-making principles regarding software architecture, decision-making techniques definitions providing some examples, and decision-making frameworks. The next chapter will provide more information about the methodology used to ground this research.

4

METHODOLOGY

4.1 Survey Method

According to WOHLIN et al. (2012), two types of research paradigms have different approaches to empirical studies: exploratory research, where the research design is flexible (qualitative research), and explanatory research, where the procedure is fixed (quantitative analysis).

In this study, we chose the exploratory approach to interpret the phenomena of decision-making in software architecture. The empirical strategy used to conduct this study is the survey, and the primary means of gathering data is the questionnaire (web-based questionnaire).

As explained by MERRIAM; TISDELL (2015) *"the overall purposes of qualitative research are to achieve an understanding of how people make sense out of their lives, delineate the process (rather than the outcome or product) of meaning-making, and describe how people interpret what they experience."*

The survey method is an instrument for collecting and analyzing data that served as a subsidy to verify the validity and pertinence of the research questions. The choice of the survey was grounded on the necessity to describe certain aspects or characteristics of the population.

The specific characteristic we would like to unveil is the decision-making process used by software practitioners. What elements are used by software practitioners, and how do they reason through them to make sound architectural design decisions? When we say good, we mean how these decisions contribute positively to project/software success.

We selected survey research because, as described by GROVES et al. (2009), this method is *"... a systematic method for gathering information from (a sample of) entities for the purposes of constructing quantitative descriptors of the attributes of the larger population of which the entities are members".*

To reinforce the appropriateness of using this approach, we quote the definition of the survey by PFLEEGER; KITCHENHAM (2001) "a comprehensive system for collecting information to describe, compare or explain knowledge, attitudes, and behavior."

Another reason for using this method is explained by REA; PARKER (2014) when they explain that *"... the foremost advantage of the sample survey technique is the ability to generalize about an entire population by drawing inferences based on data drawn from a small portion of*

that population".

Therefore, survey research is a good and valid process to understand attitudes and behaviors concerning architectural decisions made by software practitioners.

4.2 Survey Instrument: Questionnaire

As appointed by BOYNTON; GREENHALGH (2004), questionnaires offer an objective means of collecting information about people's knowledge, beliefs, attitudes, and behavior. There are other valid instruments to collect data on survey research (interviews, phone calls, focus groups, face-to-face surveys, etc.).

We adopted the web questionnaire format to collect data because according to BHAT (2018) *"it is less time-consuming than the traditional way of gathering information through one-to-one interaction and is less expensive".*

The questionnaire was elaborated based on the studies of TANG et al. (2006), GROHER; WEINREICH (2015) and WEINREICH; GROHER; MIESBAUER (2015). Survey questions can be divided in two segments: one related to demographic data (questions 1 to 5) and another to architectural DM (questions 6 to 19) as depicted in table 4.1.

The first version A had more open questions about influencing factors and hardships in making architectural design decisions. However, after running pre-testing, in order to provide more accurate data, instrument B had to be refined to include more closed questions.

4.3 Research questions

The research questions in this study are presented in table 4.2. These questions explore several topics related to the architectural decision-making process (reasoning process, influence factors, challenges and principles), and these decisions are documented according to the literature.

After describing the research questions that motivate this study, we will explore the relationship between survey and research questions in section 4.4 and how these constructs interweave.

4.4 Survey and Research Questions Alignment

The survey questions aim to assess a range of constructs that provide insights into these aspects of architectural decision-making. The choice of specific question formats and scales is driven by their suitability for measuring the intended constructs.

For example, a 5-point Likert scale is used to evaluate confidence in making decisions, with scale anchors ranging from "not confident at all" to "completely confident." This scale intends to assess the self-efficacy and experience construct, which relates to SRQ1 on how practitioners reason through decisions. Responses indicating higher confidence levels represent greater comfort and familiarity in making architectural decisions based on experience.

#	Survey Questions
1	How old are you?
2	How long do you work with IT?
3	What is your job function?
4	What is your team size?
5	What is your formal degree of education?
6	Are you responsible for making AD at your workplace?
7	How long do you have been taking/documenting AD?
8	At what stage of the project are architectural decisions made?
9	How are architectural decisions made?
10	Describe how AD are decided in your workplace.
11	How confident are you when making an architectural decision?
12	When making an architectural decision, do you try to elect one more solution in your mind?
13	Do you use any tools to support architectural decision-making?
14	How are the architectural decisions that have been made documented?
15	How important do you consider documenting the architectural decisions that have been made?
16	Are the architectural decisions made revisited during the lifetime of the project?
17	Indicate how important the factors listed below are for the architectural decision-making process.
18	Indicate how impactful the challenges listed are during the architectural decision-making process.
19	Indicate how important the principles listed below are for the architectural decision-making process.

Table 4.1: Survey Questions

The question regarding when architectural decisions are made is presented in a multiple-choice format, with options covering typical project stages. The aim is to understand existing practices on decision timing and cycles, which provides context for SRQ1 on reasoning approaches. Options like "during each sprint" and "no specific point" indicate more continuous iterative decision-making.

Regarding influence factors, a 5-point Likert scale question asks participants to rate the impact of factors like business priorities, technical constraints, team experience, etc. This scale format allows measuring the relative effect of each potential influence on decisions, aligning with SRQ2 on influential factors. Higher ratings suggest a more significant perceived influence for that factor.

The question on challenges uses a similar 5-point impact scale to gauge different issues practitioners face during decision-making. The Likert scale helps assess the severity of each challenge consistently, supporting the investigation of SRQ2. Participants also had the option to enter other challenges as free text to cover factors not listed.

To identify principles for decision-making in SRQ3, respondents rated a list of principles from "not important" to "very important" on a 5-point scale. Principles that were deemed critical

#	Research Question
SRQ1	How do software practitioners reason when making software architectural decision-making?
SRQ2	What are the potential influence factors for the architectural decision-making process?
SRQ3	Which principles do software architects take into consideration when making architectural design decisions?
SRQ4	How architectural design decisions are documented?

Table 4.2: Research Questions

were interpreted as actively influencing the respondents' reasoning process. An open-ended question also allowed entering additional principles.

Lastly, for SRQ4 on documentation, a multiple choice question asked participants to select which methods they use to document architectural decisions. The options covered typical documentation approaches identified through background research on architecture decision practices and tools. Table 4.3 correlates each question used in our questionnaire with the research questions.

Survey Questions	RQ's
At what stage of the project are architectural decisions made?	SRQ1
How are architectural decisions made? ,	SRQ1
How confident are you when making an architectural decision?	SRQ1
When making an architectural decision, do you try to elect one more solution in your mind?	SRQ1
Do you use any tools to support architectural decision-making?	SRQ1
How are architectural decisions made documented?	SRQ4
How important do you consider documenting architectural decisions made?	SRQ4
Are the architectural decisions made revisited during the lifetime of the project?	SRQ1
What are the potential factors that influence the architectural decision-making process?	SRQ1, SRQ2
What are the challenges encountered during the architectural decision-making process?	SRQ2
Which of the principles below do you consider fundamental to the architectural decision-making process?	SRQ3

Table 4.3: Survey and Research Questions

These question formats and scales were designed to provide measurable constructs, eliciting valuable data to investigate the four research questions related to architectural decision practices and influences. The chosen response options reflect literature-backed tools and techniques for architecture decisions to strengthen content validity. The combination of closed and open-ended questions also enabled the gathering of quantitative ratings and qualitative insights from participants.

4.4.1 Pre-testing

The instrument (web questionnaire) used to collect data was created by the author taking consideration previous studies (GROHER; WEINREICH, 2015) (WEINREICH; GROHER; MIESBAUER, 2015). However, before using an instrument, it is essential to evaluate it KITCHENHAM; PFLEEGER (2002a). This evaluation is called pre-testing, and there are several reasons for doing it, as described by the same author: 1) To check that the questions are understandable; 2) To evaluate the reliability and validity; 3) To assess the response rate and effectiveness of the follow-up procedures; 4) To ensure that our data analysis techniques match our expected responses.

REA; PARKER (2014) emphasizes three critical factors when conducting the pretest: clarity, comprehensiveness, and acceptance. During the design process of this instrument, we submitted it for peer review to ensure the clarity and conciseness of each question.

The pre-testing was conducted during the latter part of November until the first week of December of 2022 with respondents of the same profile (software practitioners) on the private entrepreneur and other public companies aside from Federal Justice in Brazil.

Therefore, the survey was pre-tested over one week from 25/11/2022 to 02/12/2022. The mode used for pre-testing was an online survey using Google Forms. The first version of the questionnaire is available in the appendix A.

Twelve professionals with software architecture experience participated in the pre-test, providing valuable answers. After reviewing the answers collected in this pre-testing phase, we refined our questionnaire, including more questions related to our research objectives, including adopting a Likert scale and multiple-choice answers to avoid further misunderstandings.

4.4.1.1 Improvements between questionnaire versions

Adoption of Likert Scale response format: The initial open-ended questions were converted to a Likert scale and multiple choice formats based on suggestions to improve analyzability. One example of this scenario is present in the question about how long experts are responsible for making AD. Utilizing the Likert scale provides several benefits, such as:

1. Standardization for better analysis: Likert scales provide standardized response options that can be easily quantified and statistically analyzed as interval data, allowing for better analysis JOSHI et al. (2015).
2. Reduces response burden: Likert scales require less time and effort from respondents than open-ended questions (REVILLA et al., 2016). This scale improves data quality by reducing incomplete responses.
3. Mitigates social desirability bias: Studies show Likert scales elicit more honest responses on sensitive topics than open-ended formats where respondents may filter their answers (NEDERHOF, 1985).

4. Enables comparisons: Close-ended Likert scale questions allow for easy comparative analysis between respondents and across survey iterations (ALLEN; SEAMAN, 2007). Open-ended questions make comparisons more difficult.
5. Provides benchmarking: Familiar scale formats like 5- or 7-point Likert scales allow benchmarking against established norms and standards derived from previous research DAWES (2007).

Inclusion of neutral selection in survey: Including a "Not applicable" option in surveys, as supported by academic research, improves both data quality and user experience by providing participants with an ethical way to avoid irrelevant questions.

1. Reduces response bias: According to a study in the Journal of Official Statistics, forcing respondents to provide an answer even when the question seems inappropriate can introduce response bias. The "Not applicable" option reduces acquiescence bias or social desirability bias in responses (SMYTH et al., 2006).
2. Provides a way to avoid questions: HOTTOIS (1985) indicates that some respondents may perceive sensitive questions as an invasion of privacy. The "Not applicable" option offers an ethical escape to avoid answering intrusive questions.
3. Increases completion rates: DILLMAN; SMYTH; CHRISTIAN (2014) recommend including a "Not applicable" option to improve survey completion rates when respondents know they will not be forced to answer irrelevant questions.
4. Reduces satisficing: Some respondents engage in satisficing, or providing minimal effort in responding, to speed through surveys, which can reduce data quality (KROSNICK, 1991). The "Not applicable" option reduces satisfaction since respondents can quickly opt out of irrelevant questions.

Increase perception of relevant aspects: Three questions were added in the final version of the questionnaire. The first question was related to demographic information, specifically, the degree of formal education. The second and third questions were related to Documentation AD. One was about the importance of documenting AD, and the other was about the frequency of revisiting AD during the project's lifecycle.

4.4.2 Population and Sample

This study's target population comprises software engineers, experienced developers, software architects, team leaders, project managers, and IT managers responsible for making architectural design decisions in Brazil. In particular, we targeted individuals involved in the decision-making process. Initially, we invited public government professionals, particularly those in the Judiciary, by e-mail and contacted IT managers by phone. Nevertheless, the response

rate in pre-testing was lacking, so we pivoted to use Social Network Service (SNS) platforms such as Linkedin and Twitter to reach a more significant amount of the population.

The exclusion criteria used by this study were that the target population must be responsible or should participate in the process of making architectural design decisions. Those responsible for the teams but who do not participate in those kinds of decisions are not considered.

4.4.2.1 Sampling Method

According to KITCHENHAM; PFLEEGER (2002b), the sampling method must be rigorous to make strong inferences about the target population. In this study, we used a non-probabilistic sampling method known as convenience sampling. This method was chosen due to the ease of access to the primary researcher, and as indicated by KITCHENHAM; PFLEEGER (2002b), it involves obtaining responses from those who are available and willing to participate.

However, it is necessary to address some limitations of using this kind of non-probabilistic sampling method.

Generalizability: The findings obtained from a non-probabilistic sample may not be applicable to the broader population. Without randomization, it is impossible to definitively assess the likelihood of the sample being representative, thus weakening the external validity of the research (BRYMAN, 2015).

Sampling Bias: The subjective nature of non-probabilistic sampling may introduce bias into the selection process. For example, if the researcher selects subjects that are readily available or align with their expectations, it could skew the results. This bias can lead to over- or under-representation of particular groups, further undermining the study's validity (LAVRAKAS, 2008).

Potential Confounding Variables: Without randomization, there may be hidden biases or confounding variables that are not controlled for, which could affect the study's conclusions. This lack of control over extraneous variables can confound the results, making it challenging to establish causal relationships (BHATTACHERJEE, 2012).

Ethical Considerations: In some cases, non-probabilistic sampling can lead to ethical concerns, particularly if certain groups are systematically excluded from the sample. This can lead to an inequitable representation and potential misinterpretation of the phenomenon under study (MERTERNS; WILSON, 2018).

In conclusion, while non-probabilistic sampling offers practical advantages in specific research contexts, these must be weighed against the limitations related to generalizability, potential biases, control of confounding variables, and ethical considerations. Proper acknowledgment and mitigation of these limitations within the research design and analysis can enhance the credibility and integrity of the study.

4.5 Data Analysis

In order to work efficiently with the data created by this study, we used the RStudio framework. All data available in Google Forms was exported in Comma Separated Values (CSV) format. Thanks to this tool, it was possible to create R scripts to manage data parsing/transformation and analysis. The most intensive libraries used were: dplyr for data manipulation, such as omitting columns and converting values, and ggplot2 to generate graphics in a better format. All plots generated were validated using Adobe Color Wheel tool [1].

4.6 Summary of this Chapter

In this chapter, we characterized the methodology used to ground this work. We reasoned about why choosing survey research and how this method is the most appropriate for this study. Further, we described the correlation between the research questions and the questionnaire, its theoretical ground, and other essential aspects of survey research, such as population, sample, and sample method.

[1] The use of this tool allows us to check if there is a color conflict in graphics or if the color scheme is friendly for colorblind people

5

DATA ANALYSIS AND EVALUATION

This chapter will discuss data collected in this survey, and the results are presented. First, we will represent the population's demographic data. Second, we will discuss architectural decision data, including how software architecture decisions are made, how these decisions are documented, and influence factors, hardships, and principles used in decision-making. Third, we will evaluate the descriptive results of survey data and their relationship with the literature.

5.1 The survey sample characterization

From January to March of 2023, the online Google survey service, Google Forms, was used to design and host the web questionnaire for this survey. There were 50 participants, with 33 individuals being responsible for architectural decisions. The questionnaire comprised 19 questions, including five demographic questions and 14 questions pertaining to decision-making in software architecture.

5.2 Demographic data

This section presents the characterization of the sample's demographic data. Table 5.1 shows the job function. We targeted professionals who work with software architecture, and they are directly related to decision-making. We identified several roles in our survey, varying from technical (developers, engineers, and architects) to management (team lead, director, project managers, and others).

Role	Number of Experts
Architect	6
Consultant	1
Director	2
Other	3
Project Manager	3
Senior Developer	5
Software Engineer	7
Team Lead	6

Table 5.1: Job Function distribution

The survey participants' job distribution adheres to WEINREICH; GROHER; MIES-BAUER (2015) and GROHER; WEINREICH (2015) studies. Senior developers, architects, and team leads are the most prominent roles responsible for decision-making. From a team size perspective, as shown in Figure 5.1, most of the survey's participants' teams comprise 1-10 members.

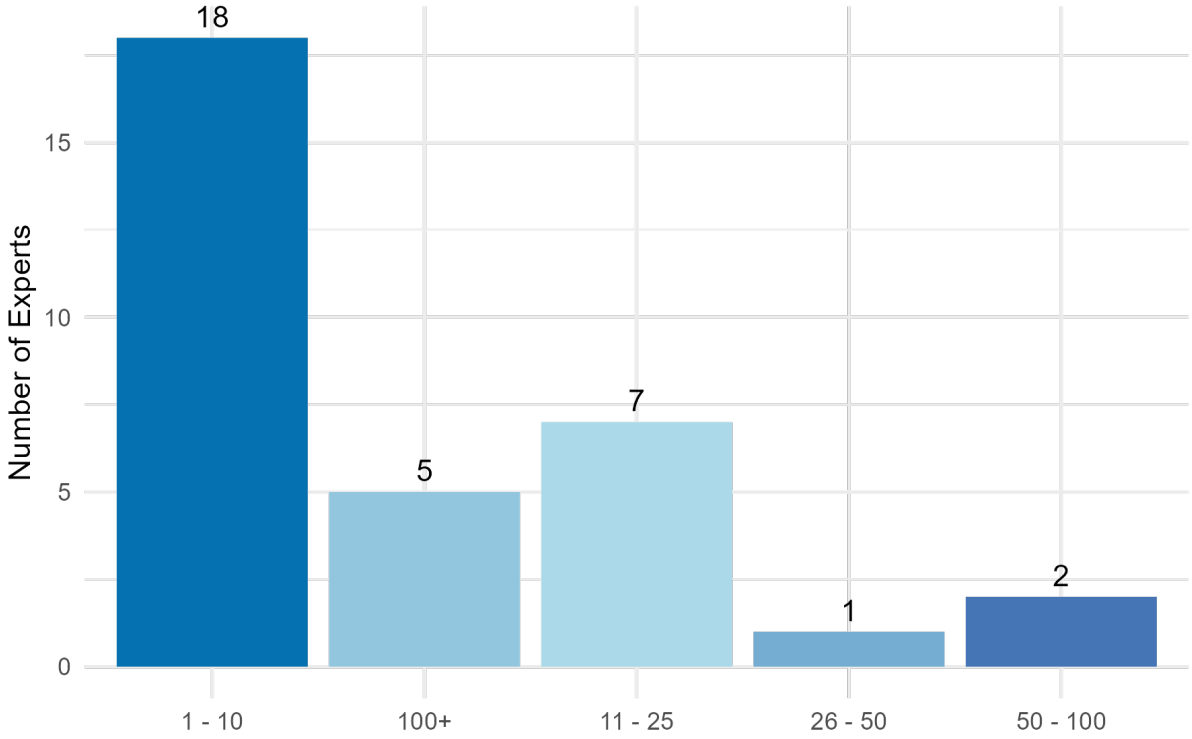


Figure 5.1: Team Size

Table 5.2 displays information about age, education level, and years of experience in IT. From the age data, we can perceive that 75% of the participants are in the age range of 31 years or more. From education level, 60% of participants have access to a higher degree of education (MBA or higher). 90% of the participants' professional experience in IT has been more than five years.

5.3 Results

This section will present the data related to reasoning, confidence degree, architectural principles, influence factors that impair/aid the decision-making process, and how these decisions are documented. Next, we present our considerations regarding the data collected in the survey and cross-referencing it with the research questions.

Age				
20 to 30 years	*31 to 40 years*	*41 to 40 years*	*More than 50 years*	
8	14	8	3	
Education Level				
Undergraduate Student	*Undergraduate*	*MBA*	*Master's Degree*	*PhD*
2	11	10	7	3
Years of Experience in IT				
1 to 5 years	*6 to 10 years*	*11 to 15 years*	*More than 15*	
3	13	5	12	

Table 5.2: Description of age range, educational level, and years of experience in IT

5.3.1 Architectural Decision-Making

Table 5.3 the architectural decision experience of participants, including their experience documenting them. From architectural decision experience, 72% of participants have ten or fewer years in this area.

In the literature, there must be a clear correspondence between the experts' experience level and successful architectural choices. Nonetheless, a lot of implicit and explicit knowledge is required to make appropriate architectural design decisions (FARENHORST; LAGO; VLIET, 2007).

Architectural Decision Experience				
1 to 2 years	*3 to 4 years*	*5 to 10 years*	*10 to 20 years*	*More than 20 years*
8	7	9	7	2

Table 5.3: Architectural decision

From the project phase, as depicted in Figure 5.2, nine out of 33 experts reported that architectural decisions are made at the project's beginning. Seven out of 33 explain that these decisions are made at each sprint/project iteration. However, 17 out of 33 remaining experts reported that these decisions are made with no specific point or could be done anytime.

Of the remaining experts, 2 of them reported that decisions are mixed. It can be done at a specific time but revisited later on, or these decisions are made according to each challenge they face. The last one mentioned that these decisions are made at technical refinement.

From the decision-making process, as depicted in Figure 5.3, thirty-one out of 33 experts (93%) reported that the team decides together on architectural decisions. This finding is consistent with (GROHER; WEINREICH, 2015) work, albeit with a smaller percentile (63%).

From the degree of confidence when making architectural decisions, illustrated in Table 5.4, 72,7% of respondents have a relatively high degree of confidence.

From the viewpoint of the number of architectural decision options, we can see in Table

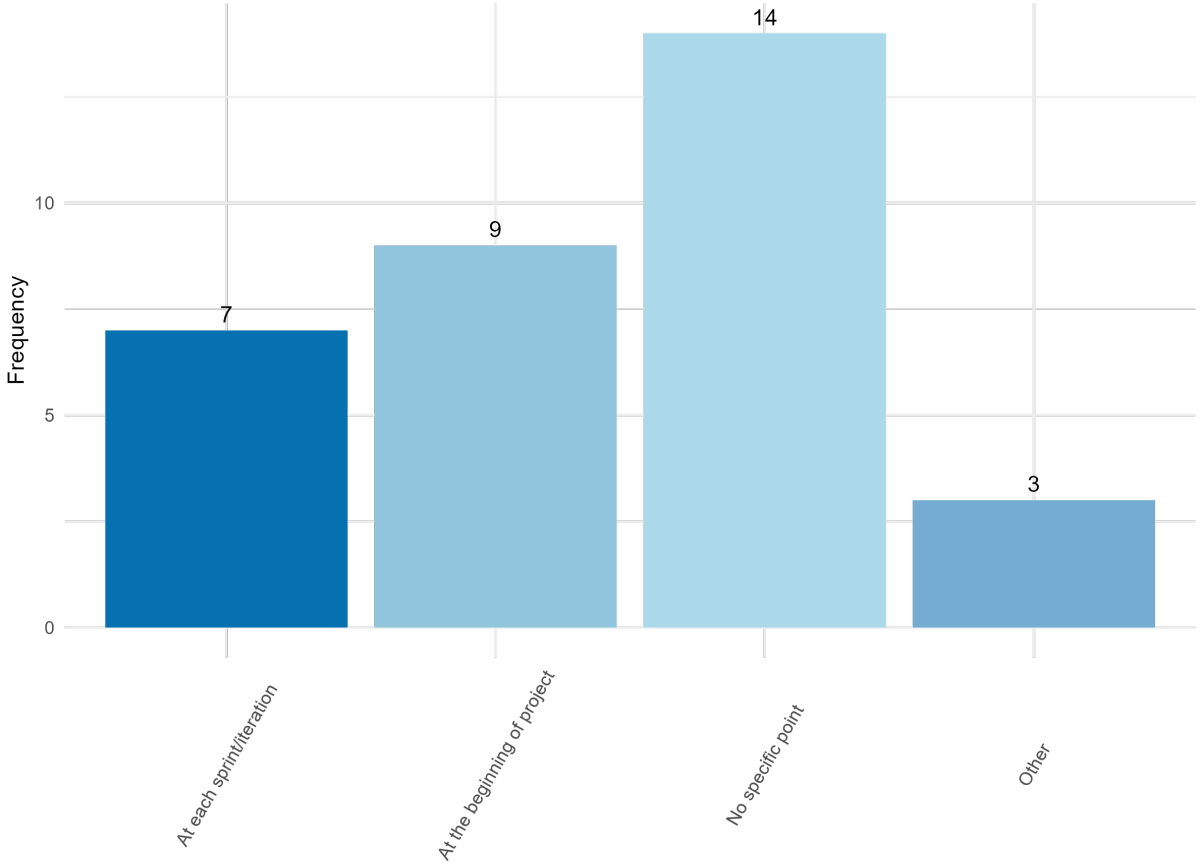


Figure 5.2: Project Phase

5.5 that 81,9% of respondents tend to think of more than one option when making architectural decisions. This finding is consistent with TANG et al. (2006) study about design rationale in design justification.

Regarding the documentation process of decision-making, 75% of the participants did not use any tool to aid their decision. Text Documents, Issue Management Systems, Code, and Wiki are popular options to document their design decisions, as shown in Figure 5.4.

The survey results revealed that an overwhelming majority of participants (97%) considered documenting architectural decisions necessary or critical, as seen in table 5.7. This finding aligns closely with previous studies that identified comprehensive documentation as a critical architecture design and development practice.

The increased documentation in larger teams resonates with studies on coordination challenges in large software projects (HERBSLEB; MOCKUS, 2003). For example, TYREE; AKERMAN (2005) study highlights how an important AD has numerous implications on a system and it can reverberate on others ADs as well.

TANG et al. (2006) emphasized the risks of undocumented design rationale, warning that lack of written records leads to "design erosion" over time as rationale becomes forgotten and context is lost. They advocated lightweight but systematic documentation to capture key decision details and tradeoffs. Our finding reinforces the continued imperative of documentation

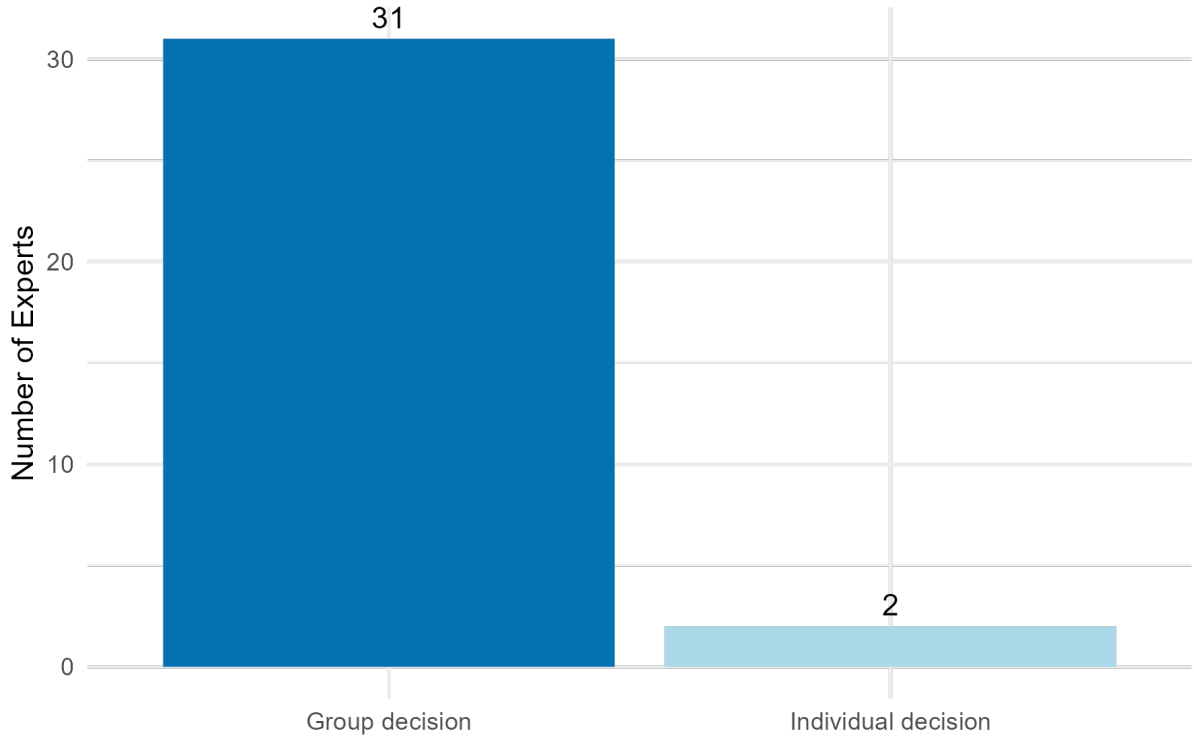


Figure 5.3: Decision-Making Process

	Not confident at all	Slightly confident	Somewhat confident	Fairly confident	Completely confident
	1	2	3	4	5
No. of respondents	1	1	7	19	5
Percentages	3	3	21.3	57.6	15.1

Table 5.4: Confidence degree in making architectural decisions

despite shifts towards agile development.

ZIMMERMANN; MIKSOVIC; KüSTER (2012) established architectural knowledge management and retention of design justifications as fundamental for system governance and maintenance. Practitioners in that study unanimously agreed that documentation was critical consistent with our findings. The consensus on the importance of documentation spans previous research and current practice.

This percentile is coherent with the need to revisit architectural decisions according to participants (64%) of this study, as shown in Table 5.6.

5.3.2 Influence Factors

This section will list the influence factors that might impact AD. Table 5.8 presents all these factors following a mixed characterization derived from TANG et al. (2006), GROHER;

	Never	Almost Never	Sometimes	Often	Always
	1	2	3	4	5
No. of respondents	0	1	5	10	17
Percentages	0	3	15.1	30.4	51.5

Table 5.5: Architectural Decision Options

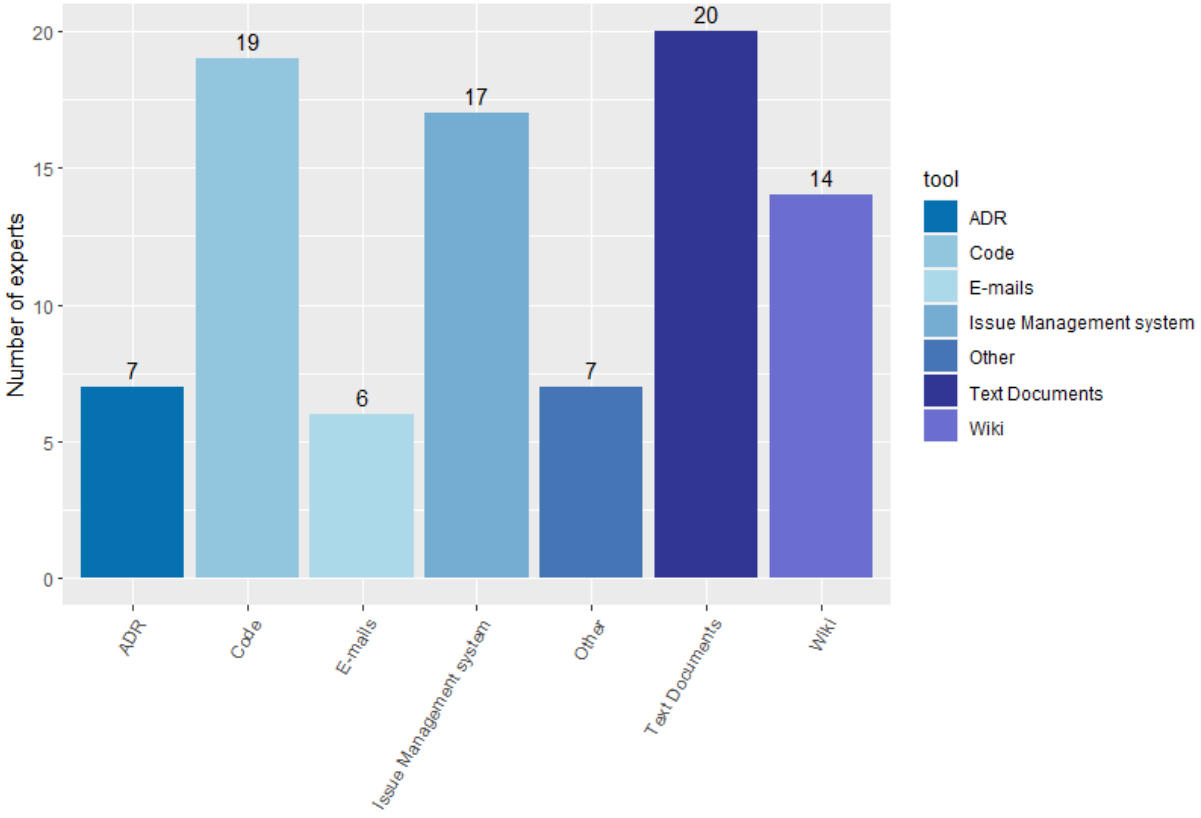


Figure 5.4: Documenting Architectural Decisions

WEINREICH (2015) and WEINREICH; GROHER; MIESBAUER (2015) studies.

We want to learn more about the factors that influence the decision-making process in software architecture since experts should be aware of multiple concerns and challenges when making these decisions, as shown in Figure 5.5.

The first factor, Company Size, is less influential than expected. However, according to GROHER; WEINREICH (2015), this factor can influence AD since larger organizations typically have a more complex structural organization and rely on standardized processes. Our results show that this finding seems inconclusive since only 51% of participants believe that company size has a degree of importance in AD is Important or higher.

On the other hand, the second one, the Business factor, is evaluated very highly by participants. 90% of study participants stated that the business significantly impacts AD. 63% even claim that this impact is very significant. Some of these aspects, for example, Time To Market (TTM), are already considered crucial to software architecture because, as explained

	Never	Almost Never	Sometimes	Often	Always
	1	2	3	4	5
No. of respondents	0	6	6	10	11
Percentages	0	18	18	30	34

Table 5.6: Architectural Decision Revisit

	Not important at all	Somewhat important	Moderately important	Very important	Extremely important
	1	2	3	4	5
No. of respondents	0	0	1	5	27
Percentages	0	0	3	15	82

Table 5.7: Architectural Decision Documentation Importance

by CARTER (2023), this business factor provides several perks such as competitive advantage, improvement of customer satisfaction, revenue growth.

The third factor, Organizational, comprises four aspects of team organization: Team Size, Team Organization, Processes and Practices, and Standards and Constraints (GROHER; WEINREICH, 2015). 75% of the survey participants believe this factor's degree of importance is Important or Very Important. Since these aspects are related to how teams are organized and interact with each other, this finding is consistent with current literature.

The influence of organizational standards and constraints was expected to be very high based on previous studies emphasizing their potent effects on architectural decisions (TANG et al. (2006); GROHER; WEINREICH (2015)). However, our results found that most participants rated this factor's importance moderately rather than very high.

The fourth factor, Technical, is very highly rated by experts. During the development process, experts must address NFR and user requirements without missing the time frame. When a company utilizes a framework to develop its products, this pattern reinforces and guides the decision-making process. This finding is consistent with literature since principles and standards, and constraints influence decision-making (GROHER; WEINREICH, 2015).

According to 51% of experts, the fifth factor, Cultural, is rated as Important or higher. This percentile of the population shows that this factor does not have a significant impact on decision-making. The sixth factor, Individual, seems to have an even lesser degree of importance since only 27% experts evaluated this factor as Important or higher. Let us consider personal experiences as an individual factor. This finding might indicate that when responding to the survey, the topic Individual was more complex than intended since WEINREICH; GROHER; MIESBAUER (2015) study affirms that personal experience is one of the most mentioned factors in making architectural decisions.

However, according to the experts, the seventh factor, Project, was one of the most critical factors. 87% of the participants highlighted that this factor is significant when making

Factors	
Company size	
Business	Business domain Business model Risk / Cost / Time Business Strategy
Organizational	Team Size Team organization Processes and Practices Standards and constraints
Technical	Principles Standards and constraints
Cultural	
Individual	
Project	Kind of project Duration
Others	Decision Scope Quality Attributes User requirements Existing Literature tool and technology available

Table 5.8: Influence Factors

architectural decisions. Compared to WEINREICH; GROHER; MIESBAUER (2015) study, the main drivers for architectural decisions vary depending on the system being developed and the organizational structure.

Decision scope, Quality Attributes, and User Requirements are rated as Very Important by 42% of survey participants. This fact confirms that these factors are very significant for the experts.

Quality Attributes are deemed necessary according to HEESCH; AVGERIOU; HILLIARD (2012) since quality goals primarily drive architectural decisions. User Requirements were also rated very highly. Software engineering models like the Twin Peaks emphasize requirements and architecture development in parallel rather than linear sequence (NUSEIBEH, 2001). This discovery reinforces user needs as a critical architecture input, aligning with participants' ratings.

Existing Literature received this very same percentile as the Moderately Important factor. Lastly, the Tools and Technology Available factor is one of the most important to decision-making since 45% of experts believe this factor is evaluated as the highest degree of importance.

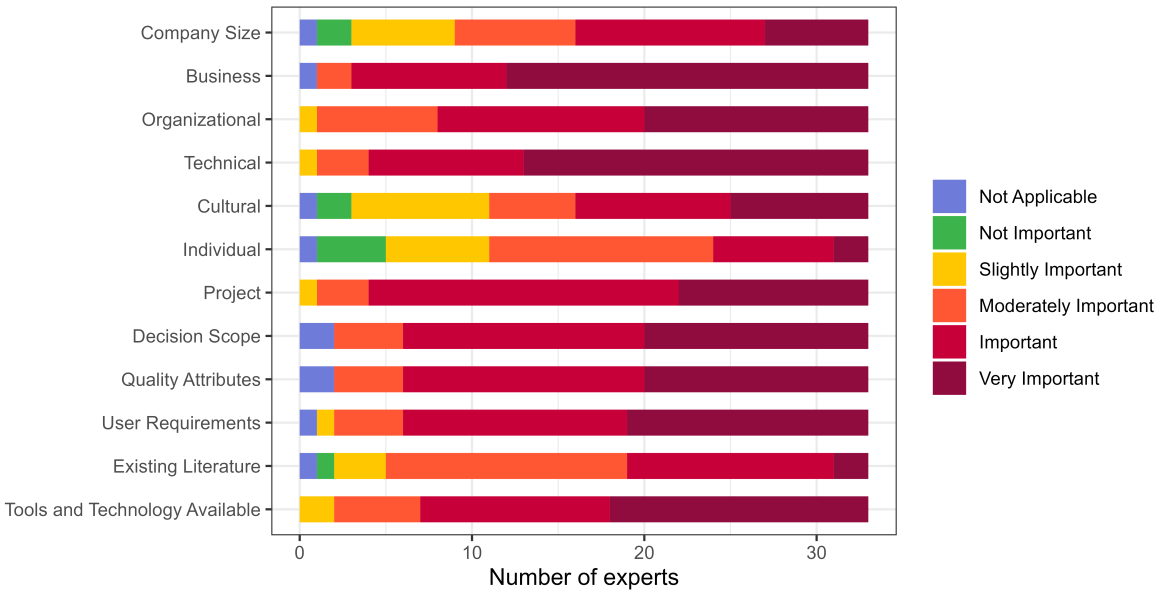


Figure 5.5: Influence factors as rated by experts

5.3.2.1 Divergences between past literature and influence factors findings

One survey result that contrasts with some previous studies is the rating of individual experience as a less significant influence factor in architectural decision-making. In an interview study, WEINREICH; GROHER; MIESBAUER (2015) found that practitioners frequently cited personal experience as affecting decisions. However, our study found individual experience ranked low, with only 27% of participants rating it as essential.

This divergence from the literature highlighting experience merits further examination. Some potential explanations can be considered: Firstly, WEINREICH; GROHER; MIESBAUER (2015) 's findings emphasized individual experience, whereas our question framed it more narrowly as just 'personal' experience. Architects may rely substantially on their technical knowledge but put little weight on other personal aspects like background or innate preferences.

5.3.3 Challenges in Decision-Making

As explained previously, architects face several challenges while making architectural decisions. This topic will explore what experts thought about some of these issues as presented in Table 5.9. Several of these problems have been studied in the literature (MOE; AURUM; DYBA, 2012); (REKHAV; MUCCINI, 2014).

For example, DASANAYAKE et al. (2015) highlights that software development methods incur challenges such as balancing upfront design and continuous design, flexibility and progress, and knowledge management.

Among the challenges presented in the survey shown in Figure 5.6, we can highlight four main drivers impairing architectural decisions: 1) lack of clarity, 2) insufficient deadline, 3) lack of business domain, and 4) conflicting NFR. The percentile of experts that find the challenges'

impact very high is above 50%.

72% of experts believe the first challenge's impact is very high. 66% of experts also find the second challenge's impact very high. 54% of experts find the third challenge's impact very high. At last, 51% of experts found the fourth one. Stakeholder conflict and legal contractual obligation challenges seem to have mixed results, indicating the need for further studies.

Challenges
Lack of clarity in Requirements
Conflicting NFR
Insufficient Deadline
Conflict between Stakeholders
Legal Contractual Obligations
Lack of familiarity in Business Domain

Table 5.9: Challenges in Decision-Making

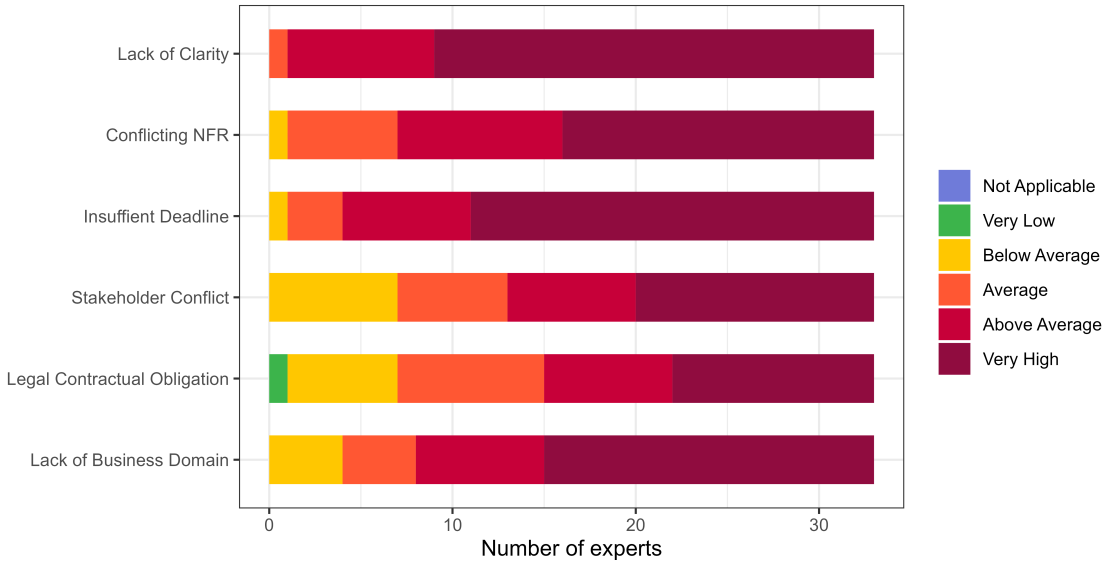


Figure 5.6: Challenges in Decision-Making rated by experts

5.3.4 Decision-Making Principles

TANG; KAZMAN (2021) proposed principles for making better software design decisions as shown in Table 5.10. In this survey, we investigated the relevancy of these principles for software experts in Brazil. These principles are closely related to Secondary Research Question 1 (SRQ1) since they impact how experts reason when making architectural decisions. Most of these principles can be converted into reflective questions to check how complaint their design reasoning is. Since most of the principles listed by Figure 5.7 actively determine the best course of action experts should take to fulfill user requirements and project needs, the results confirmed that all principles are relevant when making AD.

The first principle vouches that facts and evidences are the foundation of logical decisions (TANG; KAZMAN, 2021). 22 out of 33 of our respondents believe this statement is Very Important to AD, and 10 out of 33 rated it as Important, leading us to a percentile of 96% of experts with an evaluation of importance Important or higher.

The second principle warns us about the dangers of making assumptions without checking their veracity and the benefits of exploring contexts to broaden the number of possibilities of our design. 23 out of 33 experts rated it as Very Important, and 10 out of 33 considered it Important.

The third principle professes the importance of weighing pros and cons because they represent the arguments for and against each selection choice. This tradeoff evaluation allows us to decide why some features are more important than others and the dangers of choosing a particular approach. About 60% of the participants rated this principle as Very Important.

The fourth principle states that constraints are limitations that set the boundaries of what a solution can not do. Since boundaries are sometimes somewhat fuzzy, only 45% of the participants rated it as Very Important.

On the other hand, the fifth principle adheres to the idea that the first solution is not necessarily the best, especially when all user requirements and software features are unclear. 66% of experts agree with this statement, classifying it as Very Important.

The sixth principle defines the time relevant to a decision, and according to 66% of participants in this study, this aspect is rated as Very Important. That is understandable since, as explained before, TTM is a sensitive topic for most stakeholders.

The seventh principle alerts us to quantify the importance of choices. This urgency takes a more concrete form when defining a Minimum Viable Product (MVP) of a system. Focusing on the software's most essential parts is crucial because these elements give our clients more value. DUC; ABRAHAMSSON (2016) highlights how MVPs are used in the early stages of software startups and their role in supporting the design process, bridging communication gaps, and facilitating cost-saving activities, for example. 93% of participants classify this principle as Important or higher.

The eighth principle, Anticipate Risks, according to 81% of participants, is rated as Important or higher. Furthermore, as defined by TANG; KAZMAN (2021), a risk is the possibility of an undesirable outcome. Considering that, since 1968's, when the term software crisis was coined at NATO Software Engineering Conference, several projects did not end as well as expected because they were "always over budget, behind schedule, and unreliable" (GLASS, 1994). Risks are anecdotal evidence for experts and practitioners that hidden menaces can significantly impair software success.

5.3.5 Summary of the Findings

Regarding architectural decision-making, we found that most participants rely on group decisions, and these decisions are made at more than just the beginning of the project, being

Decision-Making Principles	
P1	Use facts
P2	Check Assumptions and Explore Contexts
P3	Weigh pros and cons
P4	Design around Constraints
P5	Generate Multiple Solution Options
P6	Define the Time Horizon
P7	Assign Priorities
P8	Anticipate Risk

Table 5.10: DM Principles

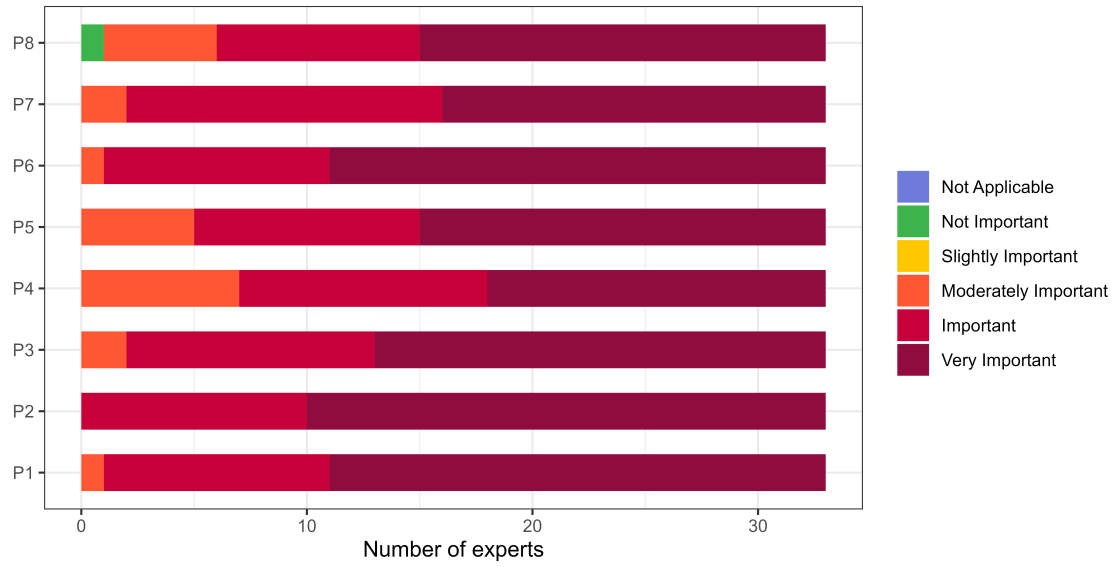


Figure 5.7: Decision-Making Principles rated by experts

necessary to revisit later on. Nearly all participants consider documenting architectural decisions important, but no specific tool for registering these decisions exists. Text documents and issue management systems are popular options for those who document their decisions.

When we look at the Challenges in Decision-Making, Lack of clarity in Requirements, Insufficient Deadline, Lack of familiarity in the Business Domain, and Conflicting NFR are the main drivers impairing AD.

Concerning Decision-Making principles, all principles have a significant degree of importance, but we can highlight principles P1, P2, P3, and P6. However, further investigation is necessary to aid the weight of these elements when making architectural decisions.

5.4 Threats to Validity and Results

In every research study, there are threats to validity that researchers need to address. Validity refers to the degree to which a survey instrument assesses what it purports to measure FINK (2010). A critical element of any empirical research study is to analyze and mitigate

threats to the validity of the results (FELDT; MAGAZINIUS, 2010) since these threats affect the possibility of generalization and replication of the study.

Following the systematic mapping on category threats to validity in Empirical Software Engineering (ESE) made by AMPATZOGLOU et al. (2019) we are going to discuss these validity threats: conclusion, internal, construct, and external.

5.4.1 Conclusion Validity

Originally called "statistical conclusion validity," this aspect deals with the degree to which conclusions reached (e.g., about relationships between factors) are reasonable within the data collected (AMPATZOGLOU et al., 2019). Threats to conclusion validity pertain to issues that affect the ability to draw accurate conclusions about relationships between the treatment and the outcome in an experiment.

In this study, we followed some procedures in order to mitigate conclusion threat: (1) search in important libraries in Computer Science (ACM, Springer, IEEE Xplore, and Science Direct) about the relationship between Decision Making and Software Architecture; (2) exclude out of context articles or articles that focus on a specific architecture approach, and (3) conducted a survey to confront experts opinion with literature review.

The sample size of software engineer studies is relatively small compared to other areas, such as Biology and Medicine. However, using our modest sample, we can perceive that those findings are accurate and appropriate due to methodological proceedings taken during the construction and application of this survey, i.e., Kitchenham and Pfleeger's series of publications about Principles of survey research (PFLEEGER; KITCHENHAM, 2001); (KITCHENHAM; PFLEEGER, 2002c); (KITCHENHAM; PFLEEGER, 2002d); (KITCHENHAM; PFLEEGER, 2002a); KITCHENHAM; PFLEEGER (2002b); (KITCHENHAM; PFLEEGER, 2003).

Some aspects are very relevant in decision-making. For example, when choosing an architectural solution attempt, the number of possibilities considered during the experts' discussion was always more than one. This number indicates that choosing just one solution is ill-advised, and revisiting is considered a good practice.

Further studies can be done to evaluate the relationship between the architectural choices made by experts and software acceptance by users. Considering the time constraint to conclude this work and the uniqueness of this kind of survey in the area in Brazil, we had to make some adjustments to conduct our research. We increased the sample diversity by using SNS such as Linkedin and Twitter instead of professionals who work for the Judiciary in Brazil. However, we acknowledge that convenience sampling reduces the study's generalization power and conveys potential biases.

5.4.2 Internal Validity

Confounding factors represent a major threat to the internal validity in empirical studies (WRIGHT; KIM; PERRY, 2010). Threats to internal validity influence the ability of researchers to draw accurate conclusions without errors due to bias or overlooking other elements, such as variables in an experimental study. For surveys, internal validity refers to the rigor of measurement: that the concepts one sets out to measure are actually measured (and completely) (WIERSMA, 2013).

We used a Likert scale score to measure each factor to ensure that factors experienced by experts when decision-making are relevant in this study. Using a quantitative metric scale, we will provide a way to measure these factors' relevancy and decision-making.

Randomization of the sample is a common technique to minimize selection bias and enhance internal validity. Since we had a limited sample size during this research, we used all respondent's data to conduct our analysis. In future work, we can use a randomization process or a control group to evaluate experts' perceptions of architectural decisions. Another possible approach is to conduct longitudinal research to check the consistency between the original and newer studies.

5.4.3 Construct Validity

According to FINK (2010) construct validity *"... is established experimentally to demonstrate that a survey distinguishes between people who do and do not have certain characteristics"*. This way, we want to distinguish practitioners according to their experience level in making architectural decisions. However, as a single researcher, efforts to mitigate subjectivity may be inherently limited (NORRIS, 1997).

In this study, the survey was designed to distinguish respondents according to their degree of experience in making architectural decisions. Those more confident in making these decisions correlate with the number of architectural solutions devised and the possibility of architectural revision. Therefore, we intend to demonstrate that similar items resonate, improving the convergent validity between research questions and the questionnaire.

The instrument used for this study was built through a literature review and a good survey design. Construct validity was strengthened by pilot testing survey questions to confirm they measured the intended constructs (RUEL; WAGNER; GILLESPIE, 2016). The survey fulfilled his purpose. Nevertheless, we perceived during the analysis that some demographic questions could enrich the discussion, such as the development process or the company's business domain. Since agile enterprises have an iterative and incremental process, architectural decisions and documentation should occur anytime during the project lifespan.

Another critical point to highlight is when we used the Likert scale to assess participants' responses to several topics in the survey. This method may have some caveats since it is vulnerable to acquiescence or social desirability bias. Still, it can provide a rational and systematic

method to express the acceptance or agreement degree of the respondents.

5.4.4 External Validity

Threats to external validity are conditions that limit our ability to generalize the results of our experiment to industrial practice. According to WIERSMA (2013), external validity refers to the validity of the survey beyond the study: its generalizability, both to the population and across contexts. Another issue pointed out by SHENTON (2004) is that the restricted geographic region and organization types surveyed constrain transferability.

The sample may not represent the population since we used a non-probabilistic sampling method in this study. The non-random sampling method limits the generalizability of findings beyond the respondents (PRICE; MURNAN, 2004). To mitigate this problem, we selected a sample of the population who deals directly with architectural decisions, excluding those who do not.

The generalizability of this study was impacted due to the sample size and the lack of determination of background of the respondents which it can generate an sampling bias. However, the selection of the characteristics related to the decision-making process and appropriateness of these characteristics with AD is consistent with the findings from previous works from WEINREICH; GROHER; MIESBAUER (2015) and TANG et al. (2006).

Nevertheless, no solid categorization of influence factors impairing the decision-making process exists. The first attempt was made by Tang et al. (2006), followed by Weinreich (2015), and since there is a relationship between these works, a general and summarized classification can be refined in later studies.

5.5 Summary of this chapter

This chapter presented the results obtained in the survey through tables and graphics, dividing data according to two main subjects: demographic and architectural decisions. The summary of the findings of this work and threats to results were analyzed, highlighting mitigation strategies to avoid bias and to provide transparency and a clear understanding of what has to be done to ensure a higher degree of validity and reliability.

As an alternative to guide IT's experts in choosing a suitable software architecture, the next chapter will present a framework to aid the architectural decision process called FAROL (FAROL).

6

FAROL: A LIGHTWEIGHT ARCHITECTURAL DECISION FRAMEWORK

This chapter presents a novel approach for supporting decision-making in software architecture called FAROL. When presenting a framework proposal, it is important to cover several key topics to provide a comprehensive understanding of the framework.

6.1 Introduction

This framework was conceived from the need to support IT experts when making AD. In Portuguese, the word FAROL means lighthouse, which seems adequate for its purpose. Anecdotal evidence indicates that a successful reasoning process to achieve an accurate system architecture comes from arduous experience and adequate technical knowledge.

With a clear understanding of the framework's purpose, it is essential to examine the complexities of architectural decision-making that FAROL seeks to mitigate. Using a housework analogy, this proposal of framework works like a game plan for home cleaning: which areas need to be cleaned, how this process is going to be, which aspects you should be aware of, and what lessons can be learned from all of this?

6.2 Problem Statement

Choosing a suitable software architecture in the early stages of a software project is considered daunting. Some experts even say that software architecture is overrated and clear and simple design is enough to achieve customers' needs (OROSZ, 2019).

Direction, when making those architectural choices, is often necessary. Some pitfalls are more evident than others. That is how this framework can be helpful. Shedding some light on those topics can provide valuable insights, reveal possible threats, and bring positive results beforehand.

Therefore, the problem lies in selecting a resilient architecture confidently. FAROL can provide a comprehensive understanding of this process. Recognizing the daunting nature of early-stage software architecture choices, FAROL sets out to achieve several key objectives that shed light on this intricate process.

6.3 Framework Objectives

These are the objectives that guide this framework:

- Comprehend the relevance of Decision Rationale in Decision-Making;
- Understand the critical aspects in choosing an architecture candidate;
- Pinpoint possible risks in selecting an unwanted architectural choice early on;
- Emphasize the importance of documentation and the rationale behind AD;
- Reason critically about influence factors and challenges involved in architectural decisions;

6.4 Theory-Building and Generalization in Software Engineering Research

Building theory is a proven process mature sciences use to accumulate and increase general knowledge. In software engineering research, some effort has been made to propose and test theories based on empirical evidence in software engineering (HANNAY; SJøBERG; DYBå, 2007).

One of the greatest arguments in favor of using theories is that they offer common conceptual frameworks that allow the organization and structuring of facts and knowledge in a concise and precise manner, thus facilitating the communication of ideas and knowledge. According to SJøBERG et al. (2008) theory can be defined as:

> *... the means through which one may generalize analytically (Shadish et al., 2002; Yin, 2003), thus enabling generalization from situations in which statistical generalization is not desirable or possible, such as from case studies (Yin, 2003), across populations (Lucas, 2003), and indeed, from experiments in the social and behavioural sciences (Shadish et al., 2002), with which experiments in empirical SE often share essential features.*

The most common process of generalization in science lies through statistical hypothesis testing, according to (JORGENSEN, 2004). This type of testing was particularly famous in empirical software engineering studies from 1996 to 2003 and reinforces the claim that statistical hypothesis testing is a frequently used ingredient in empirical software engineering studies.

However, JORGENSEN (2004) study highlights the need for more focus on research questions derived from theory and generalization across populations from sample, as opposed to isolated hypotheses and generalizations from sample to population through statistical hypothesis testing.

Therefore, FAROL's conception is well-grounded and theoretically sound since its structure is based on other theoretical constructs such as decision-making theories, systems of thinking, knowledge management, continuous improvement and learning theories, agile development methodology, and decision-making frameworks as depicted in Figure 6.1.

Figure 6.1: FAROL - Theory Building Elements

After understanding the empirical glue that ties this framework conception to empirical software engineering studies, we need to delve deeper into this framework's origins as explained in section 6.5.

6.5 Framework Origins

While developing a novel approach in the field of decision-making framework, it is necessary to gaze upon certain elements to build a resilient theoretical construct. Therefore, the following sections will describe the building blocks (aka, elements) to represent this framework.

When conceiving Strategic Planning and Decision-Making Framework (SPADE), RAJARAM (2020) delimited that its use was restricted for hard decisions. According to that author, this framework is not meant for every situation but for hard decisions that would have real consequences for a company or group since not all decisions are paramount. However, urgency and importance parameters can sometimes be perceived differently, impairing or even tipping the scales of evaluating these aspects.

On the other hand, Observe, Orient, Decide, Act (OODA) loop was created by US Air Force Colonel John Boyd. Initially, it was designed for tactical engagement, but later expanded the idea to incorporate broad strategic action (CHUN, 2019). When evaluating this framework, ENCK (2012) states that decision-making is filtered through culture, genetics, previous experience, new information, and the ability to analyze and synthesize. This highlights

the importance of identifying influence factors and the external environment before making terminal decisions.

Both approaches offer valuable insights into effective decision-making, customizing their application in complex environments. SPADEs five elements, OODA steps, Plan-Do-Check-Act (PDCA) cycle (aka. Deming Cycle) were valuable concepts in creating FAROL composition. This scenario, along with studies of TANG; KAZMAN (2021), VLIET; TANG (2016), GROHER; WEINREICH (2015), and WEINREICH; GROHER; MIESBAUER (2015) provided the necessary environment to come up with this approach.

Armed with a comprehensive set of objectives, FAROL takes shape by weaving together concepts from established decision-making methodologies, giving birth to a unique framework. Still, to leverage FAROL adherence to scientific knowledge, section 6.5.1 will explain its relationships with the evidence-based approach and its integration with existing knowledge.

6.5.1 Framework Theoretical Grounding

The FAROL aims to provide software architects with a robust yet flexible process for making complex design decisions. Grounding it in diverse theoretical foundations demonstrates that it is not an arbitrary methodology but systematically integrates relevant research and models into an evidence-based framework.

Building upon this groundwork, it is relevant to elaborate more about the decision-making theories that provide the intellectual foundation for FAROL 's innovative approach. These explicit linkages show that FAROL does not propose steps in a vacuum but purposefully interweaves knowledge across pertinent disciplines. It aims to contextualize and customize generalized concepts to the architectural decision realm. The integrated framework distills and tailors research ideas for practical usage by software teams.

Scholars emphasize the importance of grounding new models in previous theories to motivate their rationale and design (JABAREEN, 2009); (IMENDA, 2014). FAROL 's foundations provide initial credibility based on the track records of established decision-making, design, and learning theories. This credibility sets a direction for subsequent empirical testing and refinement of the untested new framework.

To leverage this framework's theoretical foundation, we will explain more about decision-making theories that strengthened its basis. The journey of constructing FAROL continues by grounding it in various theoretical frameworks, underscoring its systematic integration of diverse perspectives.

6.5.1.1 Framework Decision-Making Theories

FAROL draws inspiration from crucial decision-making theories to incorporate rational, naturalistic, intuitive, and data-driven thinking elements. Simon's theory of bounded rationality recognizes the inherent cognitive limitations that constrain human decision-making. Rather than

pursuing theoretically optimal solutions, individuals employ heuristics and satisfying strategies (SIMON, 1955). FAROL acknowledges rationality limits by providing practical guidelines and encouraging consideration of multiple options.

NDM examines how experts make decisions in real-world contexts (KLEIN, 2008). Unlike normative models, NDM recognizes the role of experience, intuition, and rational calculations. FAROL incorporates NDM aspects through its flexible documentation of rationale using simple templates rather than formal notations.

Dual-process theory differentiates two cognitive systems: fast, instinctive intuition and slower, conscious analytical reasoning (KAHNEMAN, 2011). System 1 intuition is rapid, automatic, and high-capacity, exploiting judgmental heuristics and shortcuts. System 2 thinking is deliberate, rule-governed, and requires effortful focus (THOMPSON; TURNER; PENNYCOOK, 2011), rather than strictly rational or naturalistic, FAROL combines intuitive and analytical processing. Its Execution phase applies an intuitive system of thinking to gather options and assumptions quickly. The Checking phase shifts to the analytical system two evaluation of alternatives against criteria.

DDDM advocates basing choices on empirical evidence, metrics, and statistical analysis (PROVOST; FAWCETT, 2013). Enabled by growing data volume and analytics capabilities, DDDM moves beyond intuition. FAROL incorporates data-driven aspects through steps to gather relevant architectural data points, establish measurable success metrics, and weigh factual information in evaluating options. However, it avoids over-reliance on data, using it to supplement (not supplant) expert judgment.

The mapping also identifies gaps between existing research and architectural needs that FAROL aims to address through synthesis. For instance, traditional stage-gated development may over-rely on early binding to defined requirements (RAMESH; CAO; BASKERVILLE, 2010). Lean methods provide greater flexibility but can undermine structured evaluations (BOEHM, 2002). FAROL balances these aspects in a tailored model for architecture decisions.

Transparently citing theoretical sources allows scholars and practitioners to inspect how FAROL was constructed from prior concepts. These sources enable critique about whether the right foundations have been integrated appropriately or if particular perspectives are underrepresented. Feedback can guide the strengthening of theoretical grounding in future framework iterations.

Overall, the diligent linkage of FAROL steps to precedents in decision-making, design, organizational science, and other relevant literature aims to demonstrate that it is not a speculative model. The framework distills, extends, and combines theories to address the distinct needs of architectural decisions. Tracing these evidentiary origins signals rigor and depth of research underlying FAROL 's development. This foundation better equips the new framework to elicit confidence from adopters for usage and refinement.

Building upon its theoretical foundation, FAROL draws inspiration from various decision-making theories, each contributing a unique facet to its comprehensive methodology. After

elucidating more about FAROL decision-making theories, it is possible to explore the core tenets of this framework: its structure.

6.6 Framework Structure

This lightweight framework is composed of two parts: four phases and thirteen steps. Each phase is based on the PDCA cycle, defined as Planning, Execution, Checking, and Feedback. This approach allows facing decision-making as a continuous improvement process, as shown in Figure 6.2. Each phase has several steps, as demonstrated in table 6.1. A more detailed explanation will be provided in section 6.6.1 about each of these phases.

FAROL - Phases and Steps	
Planning	Define Decision Context Reasoning Process Moment of Architectural Decision Architectural Decision Options
Execution	Architectural Decision Lifetime AD Documentation Influence Factors in AD Challenges Affecting AD
Checking	Tang and Kazman's Principles Decision Evaluation and Selection
Feedback	Communication and Collaboration Decision Governance Continuous Learning and Improvement

Table 6.1: FAROL Structure

6.6.1 FAROL Phases

FAROL leverages these strengths of PDCA in its four-phase structure. As illustrated in figure 6.2, The Planning phase encompasses problem definition, context analysis, requirements gathering, and alternative solution identification - similar to the Plan stage. The Execution phase involves prototyping potential architectures, testing assumptions, and assessing the feasibility of options - thereby enacting proposed plans. The Checking phase focuses on evaluating outcomes and results, benefiting from the objective analysis emphasis of Check. Finally, the Feedback phase consolidates learning, improves documentation, and updates the decision process through stakeholder communication. These channels key aspects of Act.

6.6.1.1 FAROL Planning Phase

The Planning phase in the FAROL aligns with early stages in design theory focused on problem framing, divergent solution ideation, and initial design synthesis. Using PDCA

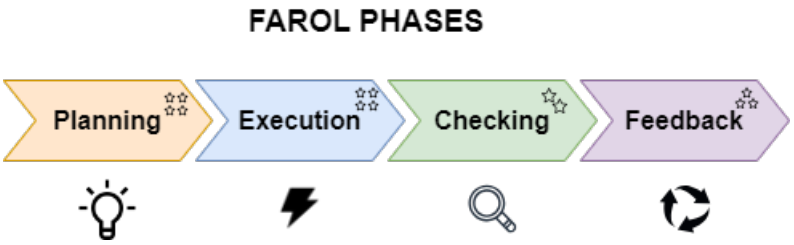


Figure 6.2: FAROL Phases

as references, this phase aligns with the "Plan" stage, where decision context is established, alternatives identified, and actions designed. Its composition is illustrated in figure 6.3.

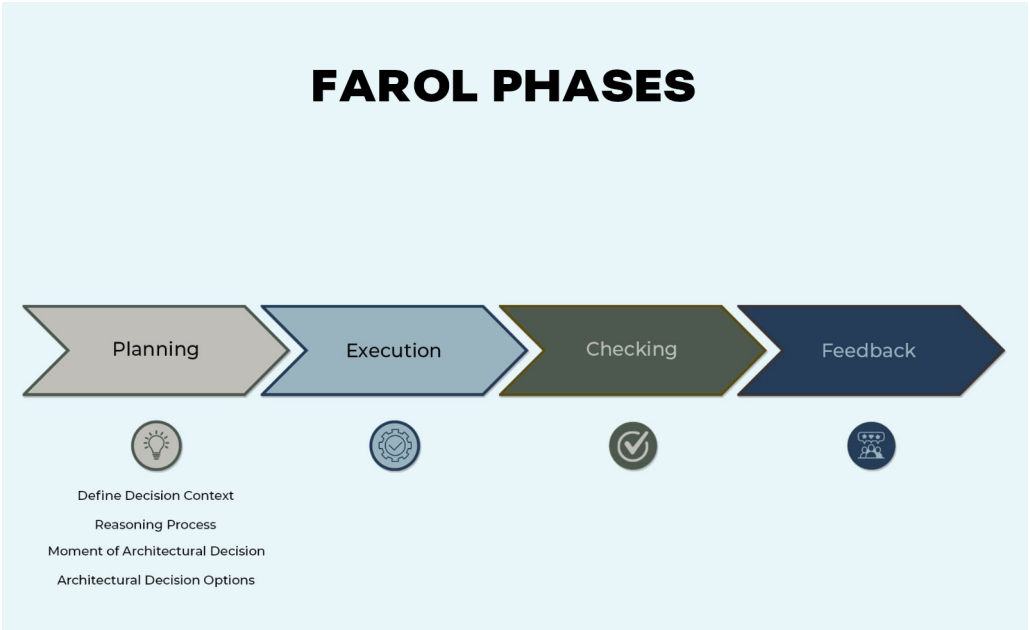


Figure 6.3: FAROL Planning Phase

Problem scoping and structuring a complex challenge into definable issues is recognized as a critical first step in design theories (CROSS, 1982). The Planning phase similarly emphasizes framing the architectural decision context and boundaries through activities like stakeholder needs analysis, gathering requirements, and scope constraints. This phase parallels the problem definition stage in the design thinking process (PLATTNER; MEINEL; LEIFER, 2016), establishing the starting point before ideating solutions.

Research shows problem scoping directly influences downstream solution development (CROSS, 2004). Defining architecture decision scope focuses on generating relevant options and prevents straying into unnecessary tangents. Scholars argue that problem framing is a creative act shaping solutions (KOLKO, 2010). In Planning, architects apply creativity in boundary setting just as in later design.

With the decision context defined, the Planning phase focuses on activities like brainstorming, market scanning, and research to generate multiple architecture solution ideas. This aligns with the divergent thinking stage in design theories, where the broadest solution set is

explored before funneling down selections (TIM, 2009). Open ideation defers evaluation to stimulate creative associations and possibilities (KELLEY; LITTMAN, 2001).

As a transition from broad ideation, the Planning phase also involves synthesizing promising options based on the decision drivers. This initial clustering and conceptualization of alternatives maps to the design synthesis phase where schemes are developed by combining ideas (CROSS, 2004). Early synthesis channels designers' attention towards more refined possibilities even as ideation continues.

By integrating problem scoping, solution ideation, and preliminary synthesis activities, the Planning phase aims to provide a smooth flow between divergent and convergent design thinking modes. This fluidity allows architectural knowledge to emerge gradually rather than suddenly synthesize after extensive ideation. Planning gives structure but encourages flexibility and reassessment of framing assumptions (DORST; CROSS, 2001).

Grounding the FAROL Planning phase in design theory foundations on problem framing, ideation, and synthesis provides rigor and evidence-based principles. By mapping design process elements into architecture decision contexts, FAROL leverages the extensive creativity research in design disciplines. This aims to stimulate expansive architectural thinking and prevent preconceived constraints from limiting options prematurely. At the same time, lightweight synthesis ties ideation back to the concrete problem, directing creative inquiry.

6.6.1.2 FAROL Execution Phase

The Execution phase maps to prototyping, testing, and iteration in lean/agile principles. This phase incorporates concepts of rapid prototyping, empirical testing, and iterative delivery from agile and lean software development movements.

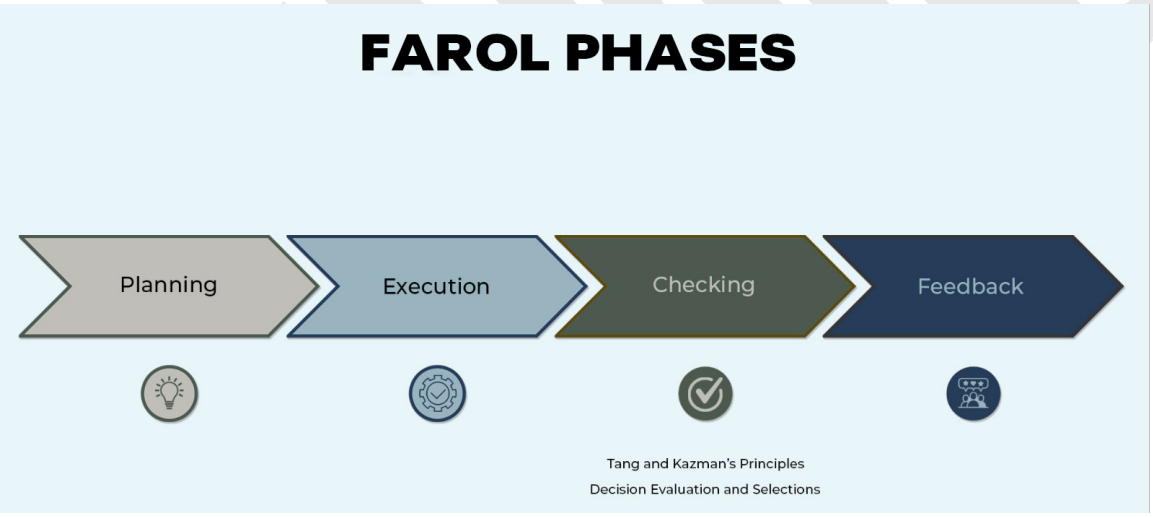


Figure 6.4: FAROL Execution Phase

A core lean principle is deciding as late as possible based on learning (REINERTSEN, 2009). Traditional stage-gated processes need more early commitment to assumed solutions

rather than evidence-based selections. The Execution phase enables architects to defer concrete decisions by first prototyping multiple architectures quickly (TIM, 2009). This parallels the lean startup's focus on minimum viable products to test assumptions (RIES, 2011).

Rapid prototyping avoids over-investment in architectural options before evaluating their effectiveness. Prototypes may range from low-fidelity sketches or simulations to working small-scale implementations. The intent validates or refutes architecture decisions through user feedback over polished productions (PLATTNER; MEINEL; LEIFER, 2016).

Short iterative cycles incorporate learning that informs subsequent solution refinement. Several scholars note that rapid iteration and feedback are instrumental for complexity management in software projects involving ambiguous user needs or technology uncertainty (MACCORMACK; VERGANTI; IANSITI (2001); RAMESH; CAO; BASKERVILLE (2010)). The Execution phase provides mechanisms to fail fast and incorporate lessons (DENNE; CLELAND-HUANG, 2003). This adaptive capability contrasts stage-gate processes optimized for predictable domains.

Testing assumptions behind architectural decisions is another agile-inspired aspect of the Execution phase. Literature highlights the risk of cognitive biases and flawed assumptions undermining decisions (JUDGMENT UNDER UNCERTAINTY: HEURISTICS AND BIASES, 1982). Explicitly testing assumptions helps prevent architecture choices from being built on sand. Early failure spotlights vulnerable areas to address rather than accruing issues downstream.

In line with agile values, FAROL emphasizes responding to evidence over rigidly following plans (BECK et al., 2001). The iterative cycles create touchpoints to reevaluate previous architecture choices regularly. This contrasts with making all decisions in a single upfront sequence isolated from validation feedback. Through empirical learning, teams can match the evolving architecture to users' needs rather than initial assumptions.

Some scholars warn of pitfalls in agile adoption, like abandoning essential Planning or skimping on non-functional requirements (BOEHM, 2002). FAROL incorporates agile aspects while providing structured decision guidelines to prevent such issues. Overall, the Execution phase leverages prototyping, testing, and iterative principles from agile models tailored for architectural decisions. This promotes evidence-based assessments and flexibility in architecture choices throughout a system's lifecycle.

Overall, the Execution phase leverages prototyping, testing, and iterative delivery strategies that emerged from lean and agile movements. This provides a mechanism to probe architectural decisions using empirical evidence over purely anticipation-based selection familiar to conventional process models. Avoiding big designs up-front reduces exposure from incorrect assumptions or outdated user needs (DYBå; DINGSøYR, 2008).

6.6.1.3 FAROL Checking Phase

The Checking phase aligns with the evaluation and analysis steps in decision-making models. This phase incorporates objective evaluation and analysis concepts from established decision-making theories and process models.

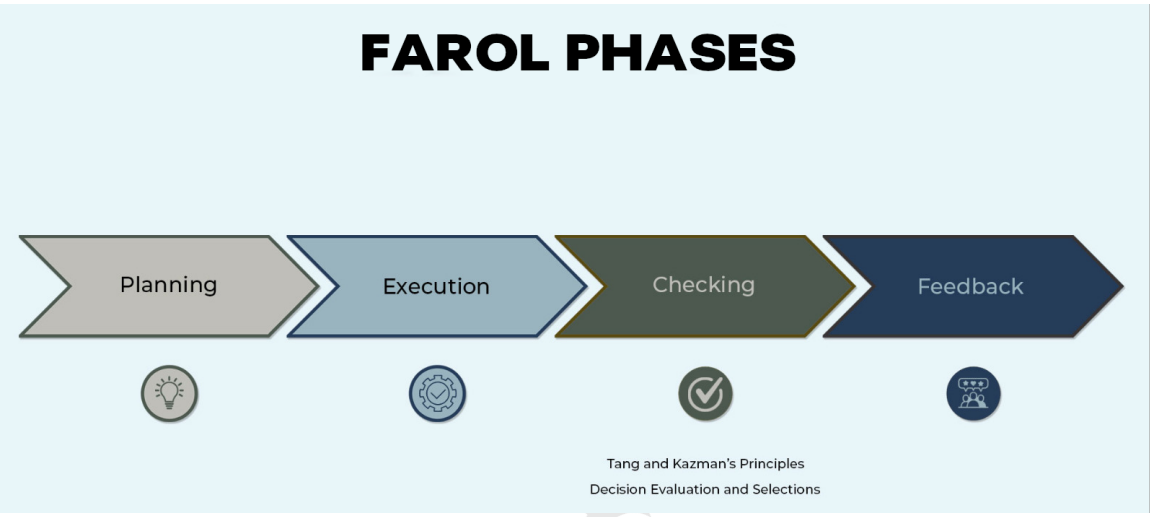


Figure 6.5: FAROL Checking Phase

Simon's theory of bounded rationality asserts that human judgment is limited when assessing complex architectures. People employ heuristics that may introduce bias or logical flaws (SIMON, 1955). The Checking phase aims to support sound evaluations using systematic analysis of alternatives against criteria.

This aligns with prescriptive decision models that structure evaluation to reduce subjective cognitive traps (HAMMOND; KEENEY; RAIFFA, 1998). Checklists prompt consideration of critical issues that architects may overlook under time pressure. Weighting techniques help prevent undue influence from personal preferences.

By delaying in-depth evaluation until this phase, FAROL avoids anchoring and confirmation bias that premature analysis can introduce in upfront planning (KAHNEMAN, 2011). The Execution phase develops multiple options in parallel to mitigate preconceptions. Checking evaluates these competing solutions neutrally using evidence-based selection criteria. Value-focused thinking techniques are incorporated to assess whether options address stakeholder needs and align with organizational objectives KEENEY (1996).

The Checking phase may apply modeling, simulations, proofs-of-concept, or experiments to evaluate options for complex architectural decisions under realistic conditions. Literature highlights the value of empirical testing over untested assumptions (KLEIN; MECKLING, 1958). These analytical evaluations aim to mitigate cognitive biases in human decision-making.

The Checking phase provides architects with a systematic decision-selection approach by supplementing intuitive judgment with structured, rational analysis methods. This reflects literature advocating integrated thinking leveraging the benefits of intuitive and analytical cogni-

tion (HAMMOND; KEENEY; RAIFFA, 1998). Checklists, weighting criteria, and experiments inject greater objectivity into architecture choices.

Overall, incorporating evaluation concepts and techniques from decision theory aims to reduce subjective limitations in assessing architecture options. The Checking phase deploys empirical and analytical methods to counterbalance bounded rationality constraints. This provides a rigorous yet flexible evaluation foundation for architecture decision-making before final selection.

6.6.1.4 FAROL Feedback Phase

The Feedback phase reflects learning, improvement, and adaptation research. This phase aims to enable continuous improvement by integrating architectural learning into the decision process. This maps to research on organizational learning and adaptive management.

Becoming a learning organization revolves around modifying behaviors based on new knowledge and insights (GARVIN, 1993). Senge's work highlights the role of "feedback loops" in surfacing deficiencies and strengthening systems thinking (SENGE, 2014). The Feedback phase provides mechanisms for architectural feedback through stakeholder reviews and decision audits.

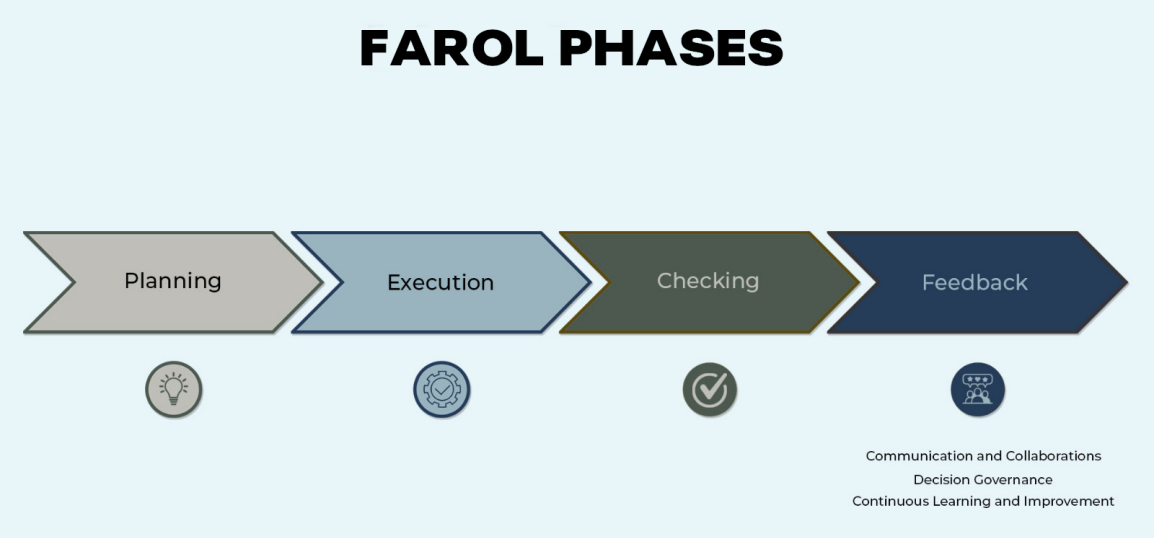


Figure 6.6: FAROL Feedback Phase

Evidence-based adaptation is another key concept from learning literature manifest in this phase. Findings on architecture effectiveness are translated into process changes rather than discarded after project completion (HEVNER; CHATTERJEE, 2010). Feedback channels help codify experiences into guidelines, checklists, and templates for reuse in future decisions.

The importance of feedback integration is underscored in the literature on managing uncertainty. Complex software projects require flexible adaptation as new issues emerge (THOMKE; REINERTSEN, 2012). The Feedback phase facilitates incremental adjustments in architecture planning and execution practices based on lessons from completed cycles.

Regular architecture decision reviews, as undertaken in this phase, reflect principles of continuous improvement models like PDCA that emphasize iteration MOEN; NORMAN (2009). Feedback flows enable incremental enhancement over time rather than disjointed upgrades. Review meetings provide touchpoints for stakeholders to discuss issues and improve mutual understanding.

Tracing the origins of each FAROL phase and activity to established theories lends credibility to its conceptual basis. For instance, the Planning phase ties to design thinking research on problem framing, solution ideation, and initial synthesis (KOLKO, 2010); (TIM, 2009). The Execution phase maps to lean startup principles of rapid prototyping, empirical testing, and iteration (RIES, 2011). Checking incorporates structured decision-making techniques to reduce cognitive limitations (HAMMOND; KEENEY; RAIFFA, 1998). Feedback channels leverage organizational learning models focused on continual improvement (SENGE, 2014).

6.6.1.5 FAROL Phases and PDCA Cycle

FAROL takes inspiration from the widely adopted PDCA continuous improvement cycle to inform its phased structure and iterative approach as illustrated in figure 6.7. First, conceptualized by management theorist W. Edwards Deming in the 1950s, the PDCA cycle is a systematic process for controlling and continuously improving processes, products, or services (MOEN; NORMAN, 2010). It has been applied across manufacturing, business, education, and healthcare sectors.

At its core, PDCA consists of four repetitive stages: Plan, Do, Check, and Act (PIETRZAK; PALISZKIEWICZ, 2015). In the Plan phase, objectives are established, processes analyzed, and success metrics defined. Potential solutions and courses of action are developed. Next, in the Do phase, proposed solutions are implemented on a small or experimental scale.

The Check phase involves measuring outcomes and results and then comparing these to expected goals. Gaps and deviations are identified and investigated. In the Act phase, learnings from the previous iterations are analyzed to determine necessary improvements and adjustments to objectives, plans, or implementation strategies. The revised plan kicks off another cycle of PDCA.

6.6.1.6 FAROL Phases and Design Thinking Principles

The FAROL aims to leverage key concepts from design thinking to promote human-centered, creative, and experimental perspectives in architectural decision-making.

Design thinking applies designer workflows and mindsets to problem-solving beyond traditional design domains (BROWN, 2008). It provides a theoretical grounding on creative ideation, rapid prototyping, bias mitigation, and evidence-based iteration. As LIEDTKA (2014) summarizes, core tenets of design thinking include focusing on human needs, "flipping" questions to reframe problems, ideating without judgment, seeking inspiration, rapid concept prototyping,

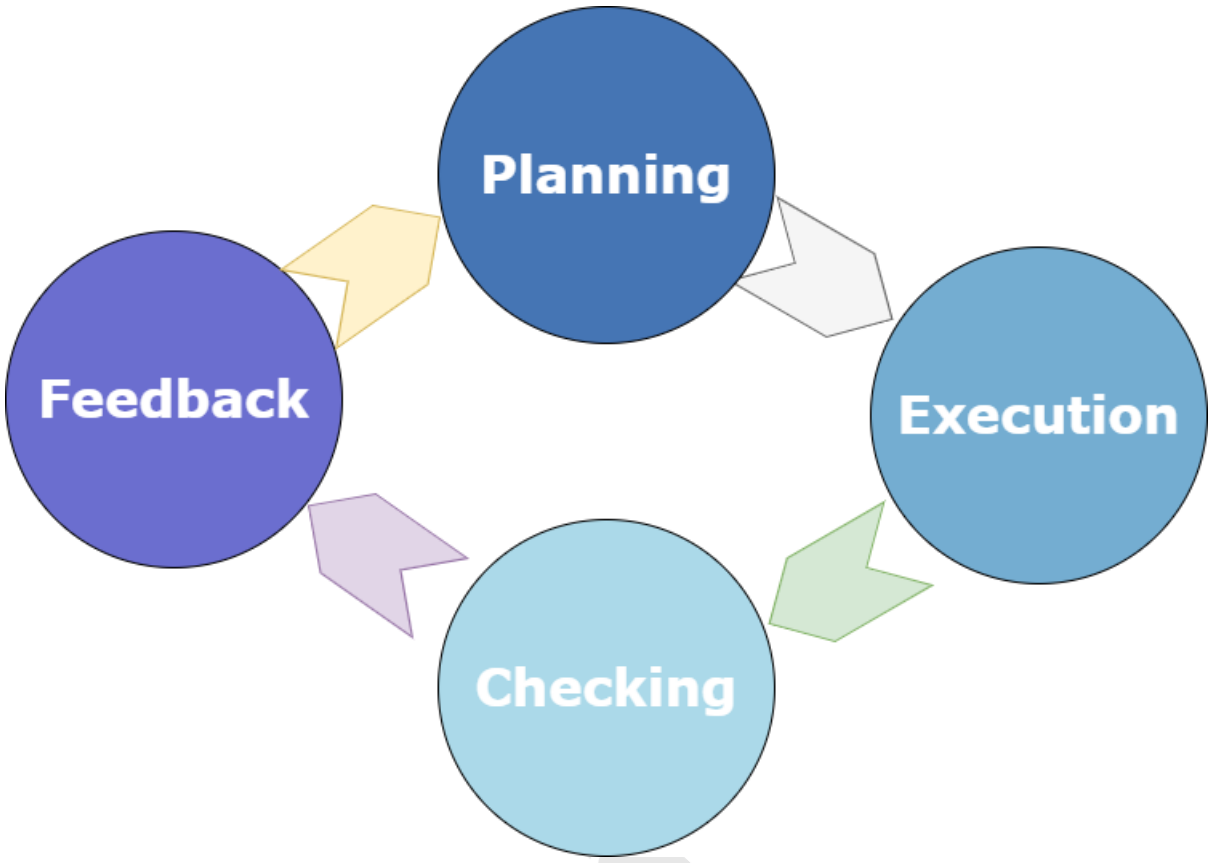


Figure 6.7: FAROL Phases - PDCA View

and iteratively evolving solutions.

Several scholars have synthesized models to encapsulate the design thinking process. For instance, the Hasso-Plattner model outlines six nonlinear steps: empathize, define, ideate, prototype, test, and implement (PLATTNER; MEINEL; LEIFER, 2016). Stanford's d.school promotes a similar five-stage framework: empathize, define, ideate, prototype, and test (DSCHOOL, 2023). Such models move from initial divergent exploration to convergent refinement.

FAROL incorporates aspects of these design thinking innovations customized for architectural decisions. The Planning phase draws on human-centered principles to define decision context and emphasize understanding stakeholders' needs that architecture must satisfy (PLATTNER; MEINEL; LEIFER, 2016). The Execution phase leverages design thinking's focus on rapid ideation and low-fidelity prototyping of multiple solutions before selection rather than deciding prematurely (BROWN, 2008). Lightweight ADR documentation templates enable this experimentation. The Checking and Feedback phases enact iterative refinement of options based on empirical testing and user feedback, mirroring design thinking test-and-learn loops (LIEDTKA, 2014).

FAROL counterbalances its creative aspects with systematic idea evaluation during the Checking phase. Others critique it lacks methodological rigor and consistency in application (CARLGREN; RAUTH; ELMQUIST, 2016). FAROL provides lightweight, customizable guidelines to promote rigor without over-engineering the human-centered process. Hence, it

preserves design thinking's adaptability to context.

Overall, FAROL leverages design thinking concepts, including human centricity, suspension of initial judgment, ideation velocity, early experimentation, and iterative evolution. It aims to infuse architectural decision practices with designer perspectives on problem framing, abductive reasoning, tolerance of ambiguity, creative expression, inquiry, and user empathy. As BROWN (2008) notes, design thinking "taps into capacities we all have but that are overlooked by more conventional problem-solving practices." FAROL aims to activate these capacities to enhance architectural decisions.

Integrating design thinking within FAROL 's broad theoretical grounding illustrates its applicability in addressing ambiguous, open-ended, and multivariate architecture challenges. FAROL incorporates related design perspectives like patterns of divergence and convergence (TIM, 2009), double diamond process models (COUNCIL, 2023), and focus on decision journey over solutions (KOLKO, 2010). However, it tailors these to the unique context of architecture, where technical and creativity requirements co-intersect. As FAROL is empirically tested and refined, design thinking elements enable human-centered, creative, yet analytically rigorous architectural decisions.

In the next section, we will dive into another essential construct for this framework called FAROL Steps.

6.6.2 FAROL Steps

FAROL steps are conceived to be a practical guide to experts during their process of choosing appropriate AD as defined in table 6.1. Using this table as a basis, we will explain how each step is designed and what concerns should be addressed.

Define Decision Context: This step allows experts to focus on the context. What is trying to be solved? What is trying to be built? Exploring project goals and objectives is mandatory to achieve success and manage stakeholders' requirements and expectations. This step is relevant to identifying key stakeholders and defining an appropriate communication channel.

Reasoning Process: Reflects on the reasoning process (intuitive or naturalistic). What form of rationale allows practitioners to feel more confident in their choices? Based on previous experiences or following a set of practices and standards defined by the enterprise? Individual or Group Decision?

Moment of Architectural Decision: Suggests an iterative approach. Instead of making all decisions at a single point, promote a continuous decision-making process throughout the project lifecycle.

Architectural Decision Options: Enables exploration, comparison, and evaluation of AD alternatives, leading to better-informed decisions. However, it is necessary to avoid selecting an excessive number of options.

Architectural Decision Lifetime: Involves revisiting architectural decisions at pre-

defined intervals to assess their effectiveness, relevance, and alignment with evolving project needs. Consider major milestones, project phases, or significant changes as triggers for decision reevaluation. Adapting decisions when necessary ensures their ongoing value.

AD Documentation: Suggests adopting a documentation framework that supports capturing essential information about architectural decisions without excessive bureaucracy. Concise templates or structured formats that capture decision context, rationale, alternatives considered, trade-offs, and implications are good examples of achieving this.

This documentation should be easily accessible, searchable, and maintainable. Experts argue that whiteboard meetings can effectively document AD (ALMEIDA; AHMED; HOEK, 2022). To increase the understanding of how architecture documentation methods influence FAROL framework's documentation step, section 6.6.2.1 will provide more information.

Influence factors in AD: Warn us about the influence factors across multiple dimensions, such as business, organizational, cultural, individual, technical, project-related, and other factors. Create awareness among decision-makers about these factors and their potential impact on architectural decisions. Encourage collaboration with relevant stakeholders who possess domain expertise.

Challenges Affecting in AD: Raise the awareness of the team and can be helpful to mitigate challenges such as lack of clarity, i.e., establishing precise requirements engineering practices to address lack of clarity in requirements. Identify and resolve conflicts in non-functional requirements through negotiation, prioritization, or trade-off analysis. Ensure realistic scheduling and resource allocation to address insufficient deadlines.

Facilitate effective stakeholder communication and conflict resolution mechanisms to tackle conflicts between stakeholders. Consider legal and contractual obligations during decision-making to avoid violations or conflicts. Promote knowledge-sharing and learning initiatives to address the need for more familiarity in the business domain.

Tang and Kazman's Principles: It is a holistic way to confirm that good practices for software architecture decisions have been used. The incorporation of these principles in the decision-making process can be helpful. For example, base decisions on factual information, empirical evidence, and data whenever possible.

Decision Evaluation and Selection: It helps us to establish decision evaluation criteria aligned with project goals, stakeholders' requirements, and quality attributes. Besides, decision evaluation techniques can be used to check the relationship between the results and the defined criteria.

Communication and Collaboration: It has two primary responsibilities. Facilitate effective communication channels: Establish channels for communication and collaboration among decision-makers and stakeholders. Utilize tools like online collaboration platforms, discussion forums, and regular meetings to share information, discuss decisions, address concerns, and gather feedback.

Encourage cross-disciplinary collaboration: Foster collaboration between architects,

developers, business analysts, and other stakeholders to ensure a comprehensive understanding of the decision context and to leverage diverse perspectives.

Decision Governance: It is composed of two activities. Definition of decision governance structure and Implementing decision review and approval process. First, it is necessary to establish a clear decision-making authority and accountability structure. Second, it is necessary to evaluate, review, approve, and document architectural decisions objectively.

Continuous Learning and Improvement: It has two stages. Promotion of a learning culture and Auto-Evolutionary form. The first stage has the role of encouraging continuous learning and improvement in architectural decision-making. The second stage is necessary to assess the effectiveness of the approach. By seeking feedback, the lightweight approach can be adapted based on lessons learned and emerging best practices in the field.

6.6.2.1 Architecture Documentation and AD Documentation Step

Adequate architecture decision documentation enables teams to capture design rationale and learnings for future reference. Lightweight, incremental documentation balanced with sufficient rigor is a critical enabler for continuous architectural improvement. As such, FAROL 's documentation steps build on concepts from seminal architecture documentation methods while customizing them for conciseness and flexibility.

Prominent among predecessors is TYREE; AKERMAN (2005) template comprising sections like Status, Context, Solutions, Assumptions, Implications, and Notes. It emphasizes succinctly documenting decision outcomes rather than a comprehensive analysis. This template inspired later methods prioritizing architecture knowledge retention over detailed documentation ceremony (ZIMMERMANN; LORENZ; OPPERMANN, 2007). For instance, the Architectural Decision (AD) model defines core elements like Decision, Status, Context, Solution, Rationale, and Implications (JANSEN; BOSCH, 2005).

A highly influential documentation method is the ADR introduced by Michael Nygard. ADRs capture architecture decisions using a lightweight template covering Context, Decision, Status, Consequences, and Compliance. Nygard positioned ADRs as an agile-friendly practice for open documentation favored over exhaustive models. ADRs spawned initiatives like MADR that provided developer-friendly Markdown versions (KOPP; ARMBRUSTER, 2019).

In FAROL, the Execution phase's AD Documentation step suggests adaptable templates for capturing architecture decisions, leveraging the concepts developed in prior methods (NYGARD, 2011) (JANSEN; BOSCH, 2005). FAROL promotes succinct documentation formats focusing on what architects need to know rather than formal specifications. Simple contextual descriptions are favored over comprehensive analytical models that can overburden stakeholders (TYREE; AKERMAN, 2005).

It also incorporates contextual documentation guidance, such as documenting architecture significant enough to cost at least one week of rework if changed (BOER et al., 2007).

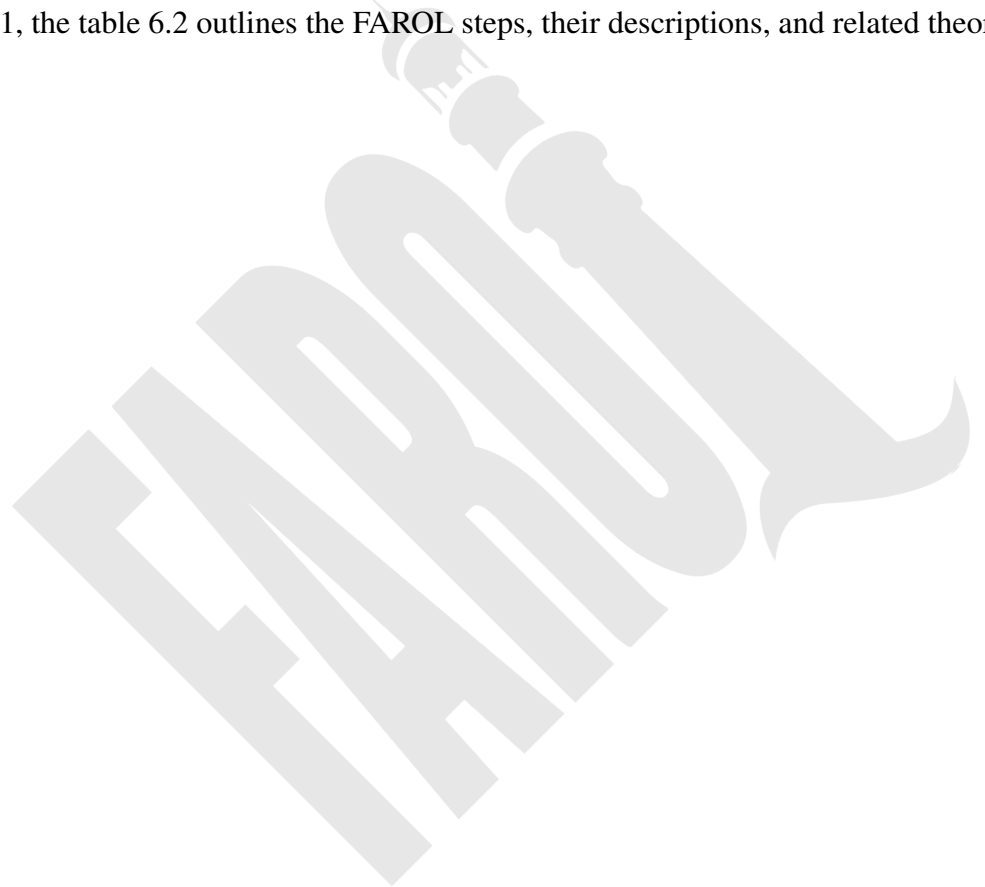
Usability research indicates architects favor simple templates that quickly create and annotate (LETHBRIDGE; SINGER; FORWARD, 2003). FAROL aims for minimal overhead, so benefits exceed the costs of documentation effort.

Overall, FAROL 's agile and lightweight philosophy towards architecture documentation balances conciseness and utility. It leverages previous research on architecture decision templates and rationale capture while customizing for flexibility over prescriptiveness. The focus is on pragmatic and outcome-oriented principles for documentation. FAROL aims to place outcomes before documenting the ceremony. It encourages teams to adapt documentation practices to their needs rather than be constrained by conventions.

The following section will explain FAROL steps and theoretical foundation.

6.6.2.2 FAROL Steps and Theoretical Foundation

The theoretical background provides the intellectual foundation for a novel decision-making framework. It helps establish the framework's validity, practicality, and potential for impacting both theory and practice in decision-making. Considering the layout provided by section 6.5.1, the table 6.2 outlines the FAROL steps, their descriptions, and related theoretical sources.



FAROL Step	Description	Theoretical Foundations
Define Decision Context	Frame architectural challenge, requirements, constraints	Problem scoping (Cross, 2004), Problem framing (Kolko, 2010)
Reasoning Process	Select intuitive vs analytical thinking approach	Dual process theory (Kahneman, 2011)
Moment of Decision	Determine continuous vs point-based decision cadence	Agile iterative development (Beck et al., 2001)
AD Options	Explore architecture patterns and alternatives	Design ideation (Brown, 2009), Conceptual solution generation (Hey et al., 2008)
Lifetime Definition	Establish intervals for decision revisit	Iterative reflective practice (Schön, 1983)
Document Decisions	Create lightweight templates to capture rationale	Agile documentation (Prenner, 2019), Design rationale capture (Chen et al., 2014)
Identify Influences	Recognize factors impacting architecture decisions	Architecture decision-making factors (Tang et al., 2006; Weinreich et al., 2015)
Mitigate Challenges	Develop strategies to address decision obstacles	Architecture decision-making challenges (Tang et al., 2006; Ali Babar et al., 2006)
Apply Principles	Leverage guidelines to improve reasoning	Software architecture decision principles (Tang & Kazman 2021)
Evaluate Options	Assess architecture alternatives against criteria	Structured decision analysis (Hammond et al., 1998), Cost-benefit analysis (Nord et al., 2007)
Communication	Facilitate stakeholder discussions and feedback	Collaborative rationality (Engelhardt & Edwards, 2019), PDCA (Deming, 2000)
Governance	Establish decision review and approval mechanisms	Architecture governance (Isozaki & Nakamura, 2005), Value-based governance (Lee & Miller, 2004)
Continuous Improvement	Incorporate learnings into future decisions	Organizational learning (Senge, 2014), Evidence-based practice (Hevner & Chatterjee, 2010)

Table 6.2: FAROL Steps and Theoretical Foundation Relationship

6.7 Comparison between FAROL and some decision-making frameworks

Table 6.3 briefly compares FAROL with existing decision-making frameworks, highlighting some key attributes. These attributes capture scope, methodology, usability, flexibility, resources needed, and knowledge retention capability. This comparison will display this novel approach's key attributes with some known frameworks.

We analyzed several existing decision-making frameworks in the context of software architecture. Each framework comes with its own set of key attributes that define its approach to guiding architectural decisions. This comparison aims to shed light on the unique characteristics of each framework and provide insights into its strengths and limitations.

In the next section, we will explore the use of this proposal through three famous architectures: Monolithic, Microservice Architecture (MSA), and Command Query Responsibility Segregation (CQRS).

Comparison	
Framework	**Key Attributes**
AHP	Pairwise comparisons and hierarchical weighting Subjective assessments Prone to inconsistencies Tedious pairwise comparisons Focused on ranking/prioritization
ADD	Formal quantitative approach Models quality attributes Not easily scalable Requires expertise in modeling Limited documentation features
ALMA	Analyzes modifiability by scenario profiling Limited tool support Does not cover all decisions Text-based scenarios Retrospective analysis
ATAM	Focuses on trade-off analysis and risk identification Rigorous but complex process Uses multiple evaluation methods Resource intensive Limited tool support
CBAM	Cost-benefit analysis of architecture candidates Quantitative financial models No guidelines for decision making Narrow focus on cost-benefit Difficult dependency modelling
Knowledge Architect	Knowledge-based using ontologies Automated reasoning and recommendations Requires extensive upfront knowledge capture Comprehensive ontology development Tight coupling to tools
FAROL	Lightweight and incremental Covers end-to-end decision process Practical guidelines based on theory Flexible documentation Facilitates stakeholder communication Customizable checklists Low tool overhead

Table 6.3: FAROL and Existing Decision-Making Frameworks

6.8 Examples

6.8.1 Monolithic Example

Context: A simple Create-Read-Update-Delete (CRUD) application to manage assets of company. The software must control where each asset is, who detains the property, and its current state.

Define Decision Context: Considering the system's simplicity, the requirements are easily perceived, and it is possible to use a monolithic approach.

Reasoning Process: Rational. Monolithic architecture is a tried and tested standard method; it is also considered more trustworthy than anything newer and, therefore, more untested.

Moment of Architectural Decision: Beginning of the project with PO, team members, and stakeholders to decide the project's scope and timeline.

Architectural Decision Options: Since requirements are relatively simple, it is possible to use a monolithic approach.

Architectural Decision Lifetime: The nature of the project allows us to use upfront decisions with a possible revaluation during later stages using stakeholders' feedback.

AD Documentation: UML diagrams and Word documents using the arc42 template can be used since these options provide a general view of what should be documented/communicated and how your architecture can be documented/communicated.

Influence Factors in AD: Project characteristics allow the use of a monolithic approach. Development speed, simplicity, and straightforwardness of creating an application based on one code are positive signals of its utilization.

Challenges Affecting AD: Scalability and maintainability are recognized as potential challenges of a Monolithic Architecture.

Tang and Kazman's Principles: As described in Table 6.4.

Decision Evaluation and Selection: Considering evaluating project progress with objective criteria. Like rapid development, faster time-to-market, and simplicity of deployment.

Communication and Collaboration: Emphasize clear communication among team members and stakeholders regarding the chosen Monolithic Architecture. Share the decision rationale and maintain open channels for discussions.

Decision Governance: Establish a clear decision-making structure for the Monolithic Architecture. This structure could involve project leads or team leaders who have the authority to ensure the architectural decision aligns with the project's goals.

Continuous Learning and Improvement: Encourage a culture of continuous improvement. Regularly assess the effectiveness of the Monolithic Architecture in meeting the immediate project goals and identify areas for improvement.

Principles	
Use facts	Monolithic Architecture is optimal for small applications because of rapid development, simplicity of testing and debugging, and cost. However, when the system grows, it can become an obstacle for business and should evolve into another form.
Check Assumptions and Explore Contexts	CRUD applications are easy to build, but validating their usability with users is necessary to avoid possible rework.
Weigh pros and cons	Simplicity (development, debugging, testing, and deployment) of monolithic are enticing. On the other hand, high coupling, testing, and performance issues are well-known disadvantages.
Design around Constraints	Considering the nature of the project, it is possible to use scaffolding techniques to accelerate the generation of the application's skeleton.
Generate Multiple Solution Options	Consider using more than one UI option for your CRUD if the app can be used on mobile or web.
Define the Time Horizon	Since the project size is small and requirements are straightforward, it is possible to define a safer timeline.
Assign Priorities	Focus on key aspects of the system and present an MVP to solve possible design doubts.
Anticipate Risk	Ensure active user participation and testing in the development phase to avoid late changes.

Table 6.4: Tang and Kazman Principles on Monolithic

6.8.2 Microservice Architecture Example

Context: A small startup of 5 members is being hired to develop a data mining and procedural data evaluation solution for a Court. Given the nature of the business, the microservices architecture was chosen, as the task segmentation could be done through two services: data extraction and processing and evaluation.

Define Decision Context: The analysis of procedural data involves mapping relevant aspects of this data. What classes will be needed? What relationships are relevant?

Reasoning Process: Intuitive. Considering that the team already has previous experience with MSA, it seems a reasonable choice.

Moment of Architectural Decision: Beginning of the project with Whiteboard Meetings at each iteration.

Architectural Decision Options: Considering a service-oriented approach, MSA, SOA, or Serverless could be used.

Architectural Decision Lifetime: Considering the usage of agile methodology, architectural decisions can be reevaluated at the start of each sprint.

AD Documentation: Whiteboards during meetings with Wiki.

Influence Factors in AD: Considering organizational aspects, company size, team expertise, and previous experiences, we can identify these factors that vouch for MSA choice.

Challenges Affecting AD: Here are some examples of these challenges:

- Lack of familiarity with procedural data can bring undesired results;
- Scalability and Cost can be conflicting NFRs;
- Legal Contractual Obligations, such as LGPD, can impair the viability of the solution;
- Hosting solution on-premise or on the cloud can incur conflict between Stakeholders;

Tang and Kazman's Principles: As described in Table 6.5.

Principles	
Use facts	Since the team has already used MSA, they know good prospects for this solution.
Check Assumptions and Explore Contexts	During the process of data analysis, it is essential to check data consistency and relationships.
Weigh pros and cons	Complexity, latency, and points of failure are examples of problems with using MSA. On the other hand, flexibility, scalability, and fault isolation are advantages of MSA.
Design around Constraints	Some of the procedural data comes from PDFs and multimedia. Specialized services can be designed to handle this type of data.
Generate Multiple Solution Options	parallel processing or data streaming are possible options for data handling.
Define the Time Horizon	A parameter of three weeks for each sprint can be used to assess architectural decisions.
Assign Priorities	Large Language Models (LLMs) can accelerate visualizing data relationships since data mapping is critical.
Anticipate Risk	Hosting on the cloud can be helpful, but latency and data protection (LGPD) could impact this process.

Table 6.5: Tang and Kazman Principles on MSA

Decision Evaluation and Selection: During each iteration of architectural decisions, is the MSA adoption reevaluated? This architecture overall has better performance than others.

Communication and Collaboration: For internal communication and collaboration, presential or online meetings can discuss hardships and share project evolution. The project repository can be used for this purpose as well. For external, e-mails and collaborative environments such as Microsoft Teams can be used.

Decision Governance: Are the segmentation of the designed solution in three different services informed to the client? The agreement for this model will be done using formal documents, or is the Team leader or Project Owner responsible?

Continuous Learning and Improvement: MSA's adoption can be reviewed at the end of the project. Which AD's were more important, and how were they managed?

6.8.3 Command Query Responsibility Segregation Example

Define Decision Context: The system requirements analysis indicates the need for a scalable and high-performance architecture that separates write operations from read operations.

Reasoning Process: Rational. Based on industry best practices and existing knowledge, CQRS is a reasonable choice considering the project's specific requirements.

Moment of Architectural Decision: Beginning of the project during the initial architectural design phase.

Architectural Decision Options: CQRS or a traditional monolithic architecture could be considered considering the need for separating write and read operations.

Architectural Decision Lifetime: Since CQRS introduces a significant change in the system's architecture, it is essential to consider its long-term viability. Regular evaluations and reevaluations should be conducted as the project progresses.

AD Documentation: UML diagrams or C4 Models can help convey CQRS complexity and isolate iterations between datastore and information systems.

Influence Factors in AD: Consider scalability requirements, complex domain model, high concurrency, and the need for improved performance. Evaluate how CQRS aligns with these factors and its potential impact on the project.

Challenges Affecting AD: Here are some examples of these challenges:

- Lack of familiarity with using CQRS can be an issue. It is necessary to check the familiarity of the team;
- The Complexity of maintaining Separate Read and Write Models can be conflicting NFRs;

Tang and Kazman's Principles: As described in Table 6.6.

Decision Evaluation and Selection: Regularly evaluate the adoption of CQRS by assessing its impact on performance, scalability, maintainability, and development productivity. Compare the actual benefits achieved with the expected advantages of CQRS.

Communication and Collaboration: Facilitate communication channels between developers, architects, and stakeholders to discuss the challenges, progress, and benefits of adopting CQRS. Utilize collaborative tools and platforms for efficient communication and knowledge sharing.

Principles	
Use facts	Consider empirical evidence, case studies, and success stories of organizations adopting CQRS to validate its benefits.
Check Assumptions and Explore Contexts	Analyze the system's requirements, data flows, and performance considerations to ensure that CQRS fits suitably.
Weigh pros and cons	Evaluate the advantages of improved performance, scalability, and separation of concerns against the potential Complexity and increased development effort.
Design around Constraints	Consider constraints such as existing system components, integration requirements, and the team's technical capabilities.
Generate Multiple Solution Options	Explore alternatives to CQRS, such as a monolithic architecture with performance optimizations.
Define the Time Horizon	Consider the project timeline and iterations to assess the feasibility and impact of implementing CQRS.
Assign Priorities	Determine the critical areas where CQRS can provide the most significant benefits, such as high-frequency write operations or complex reporting needs.
Anticipate Risk	Identify potential risks such as data synchronization issues, increased operational Complexity, and the learning curve for the development team.

Table 6.6: Tang and Kazman Principles on CQRS

Decision Governance: Establish clear decision-making authority and accountability for selecting and implementing CQRS. Define a decision review and approval process to ensure that architectural decisions align with project goals and standards.

Continuous Learning and Improvement: Encourage a learning culture within the team to continuously improve the implementation and utilization of CQRS. Share lessons learned, identify areas for improvement and adjust the approach based on feedback and emerging best practices.

6.9 Limitations

The limitations of FAROL can be summarized as follows:

Lack of empirical evaluation: Even though FAROL was conceived after an extensive theoretical background and supported by a survey designed on previous studies with architects, software engineers, senior developers, and other experts, it lacks evaluation on real projects and by peers. The framework would benefit from empirical and case studies to validate its

effectiveness and applicability in different contexts.

Limited peer review: While the framework has been developed with input from experts in software architecture, it has not undergone extensive peer review from the broader research community. Peer review is essential to ensure the framework's rigor and quality and identify potential biases or shortcomings.

Limited scalability: The framework may face challenges when applied to large-scale and complex software projects. As the Complexity and size of the project increase, it may become more challenging to apply all the phases and steps of the framework effectively. Further research and refinement are needed to address scalability concerns.

Subjectivity in decision-making: The decision-making process in software architecture is inherently subjective, influenced by individual experiences, preferences, and organizational contexts. While FAROL provides a structured approach, it does not eliminate the inherent subjectivity of decision-making. The framework should be used as a guide and complemented with expert judgment and critical thinking.

Limited coverage of specific architectural styles: The framework focuses on general decision-making aspects in software architecture and does not provide in-depth guidance for specific architectural styles or paradigms. Adapting and extending the framework to address the unique challenges and considerations of specific architectural styles may be necessary.

Dependence on practitioner expertise: The effectiveness of FAROL relies on the expertise and knowledge of the practitioners using the framework. It assumes that practitioners have a particular understanding and experience in software architecture. Novice practitioners may require additional support and training to apply the framework effectively.

Potential resistance to change: Introducing a new decision-making framework within an organization may face resistance from stakeholders accustomed to existing practices. The successful adoption of FAROL may require change management strategies and efforts to promote its benefits and gain organizational buy-in.

Addressing these limitations through further research, empirical evaluation, and refinement of the framework will strengthen the applicability and effectiveness of FAROL in supporting decision-making in software architecture.

6.10 Summary of this chapter

This chapter introduced the Lightweight Architectural Decision Framework (FAROL), a novel approach for supporting decision-making in software architecture. The framework was developed to address IT experts' challenges in making architectural decisions and provide a comprehensive understanding of the decision-making process.

The chapter highlighted the importance of decision rationale and technical knowledge in achieving accurate system architectures. It then presented the problem statement, emphasizing the need for a resilient and well-informed architectural decision-making process. The objectives

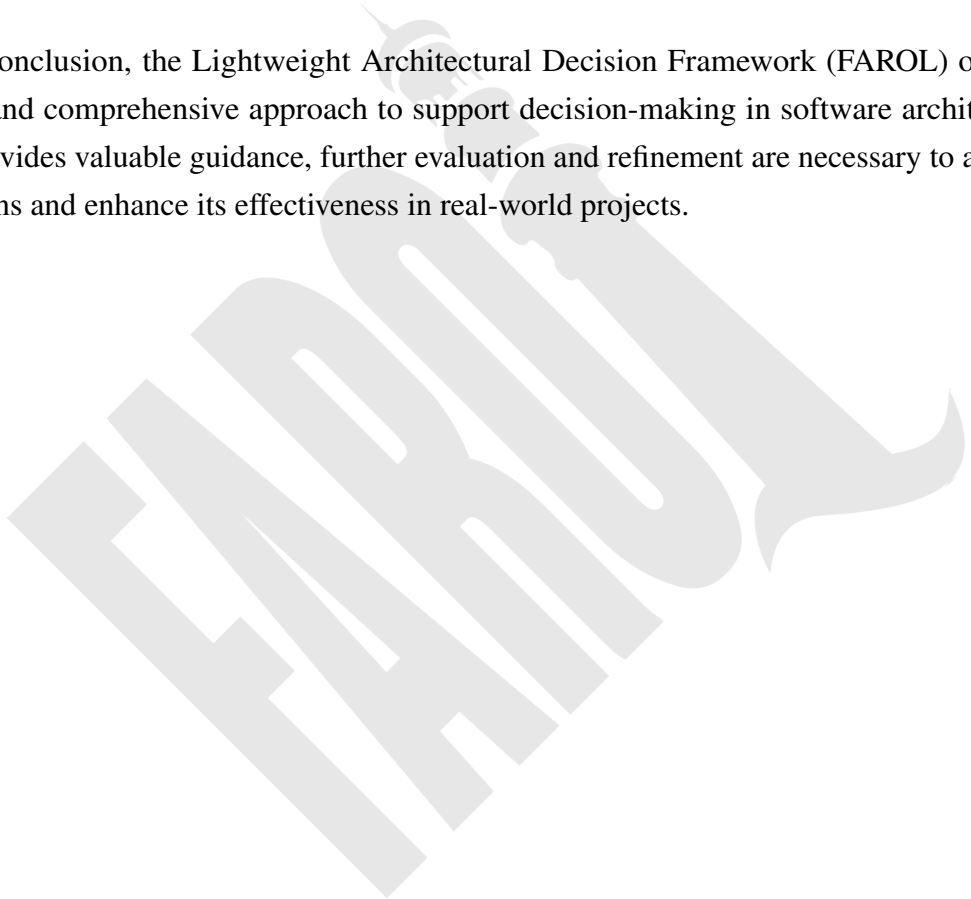
of FAROL were outlined, including comprehending decision rationale, understanding critical aspects in architecture choices, pinpointing risks, emphasizing documentation, and reasoning about influence factors and challenges.

The structure of FAROL consisted of four phases: Planning, Execution, Checking, and Feedback. Each phase was further divided into specific steps, forming a continuous improvement cycle. The chapter provided an overview of the steps and their significance in decision-making.

Two examples, Microservices Architecture (MSA) and Command Query Responsibility Segregation (CQRS), were presented to demonstrate the practical application of FAROL. These examples illustrated how the framework could be used to make informed architectural decisions, considering factors such as decision context, reasoning process, decision evaluation, communication, and continuous improvement.

Finally, the chapter discussed the limitations of FAROL, including the need for empirical evaluation, limited peer review, scalability challenges, subjectivity in decision-making, coverage of specific architectural styles, dependence on practitioner expertise, and potential resistance to change. These limitations highlighted the areas for further research and refinement of the framework.

In conclusion, the Lightweight Architectural Decision Framework (FAROL) offers a structured and comprehensive approach to support decision-making in software architecture. While it provides valuable guidance, further evaluation and refinement are necessary to address its limitations and enhance its effectiveness in real-world projects.



7

CONCLUSION

Making architectural decisions in software development is a multifaceted challenge that requires careful consideration of various factors, from technical requirements to team capabilities. Architects, software engineers, and other experts are pivotal in ensuring these decisions are well-informed, rational, and aligned with project goals. This chapter concludes our exploration by merging the insights from both perspectives into a comprehensive closing.

7.1 FAROL: A Lightweight Architectural Decision Framework

The FAROL is a milestone in the evolution of software architecture. It provides a structured approach to making architectural decisions, offering a systematic methodology for evaluating options, assessing risks, and effectively communicating findings. In the face of the inherent complexities and uncertainties of architectural decision-making, FAROL is a guiding light for architects and development teams.

Throughout this work, we've dissected the intricacies architects encounter during decision-making. The core objectives of FAROL—ensuring clear decision rationale, understanding architectural nuances, risk identification, documentation emphasis, and fostering continuous improvement—have been underscored. By unveiling its four-phase structure—Planning, Execution, Checking, and Feedback—, we've demonstrated how FAROL can be pragmatically employed in real-world scenarios through illustrative case studies.

7.2 Adaptability and Future Enhancements

Acknowledging that FAROL is not a rigid formula is crucial. Instead, it's a versatile framework that necessitates adaptability to suit specific project contexts. Each project has unique requirements, constraints, and variables, demanding tailored application of FAROL's principles and phases.

As we embrace the evolving landscape of software architecture, FAROL offers a valuable asset to seasoned architects and newcomers alike. Promoting a culture of thoughtful decision-making empowers teams to navigate the intricate landscape of architectural choices with confidence and clarity.

7.3 Contributions and Future Pathways

This research has contributed significantly to advancing the understanding and application of architectural decision-making. The creation of FAROL, with its structured approach and emphasis on rationale and documentation, is a testament to the commitment to enhancing software architecture practices.

However, every framework has limitations. FAROL acknowledges its boundaries while harnessing its strengths. In recognizing these aspects, this work lays the foundation for future research and refinement. Some avenues for further exploration include:

Empirical Validation: Empirical studies and real-world case applications are essential to validate FAROL's effectiveness and applicability in diverse contexts.

Peer Review: Engaging the broader research community through extensive peer review will enhance the framework's robustness and identify potential improvements.

Scalability Enhancement: Addressing concerns related to the scalability of FAROL for complex projects would expand its utility across a broader spectrum of scenarios.

Domain-Specific Guidance: Exploring the adaptation of FAROL to specific architectural styles or paradigms will provide targeted guidance for unique challenges.

Novice Practitioner Support: Developing resources and training for novice practitioners will enable them to utilize FAROL effectively.

Change Management Strategies: Strategies for managing resistance to adopting FAROL within organizations will facilitate smoother integration.

Web questionnaire enhancement: The web questionnaire needs to be evolved to validate this instrument through more robust statistical methods, such as exploratory factor analysis. The lack of proper mechanisms to evaluate decision-making in software architecture poses a considerable challenge to improving studies in this area.

Influence factors weight scale: Another area that requires further investigation, it's the weight of the influence factors in architectural decisions. A more robust classification of these factors can provide a better guideline for future researchers.

In conclusion, the Lightweight Architectural Decision Framework (FAROL) is a compass guiding architects through the intricate journey of software architecture. Its principles, phases, and steps pave the way for making informed, reasoned, and documented architectural decisions. As software architecture evolves, FAROL remains a cornerstone of effective decision-making. Through ongoing refinement, application, and future research, FAROL has the potential to shape the software landscape, fostering robust and resilient systems that stand the test of time.

References

ALLEN, I.; SEAMAN, C. Likert scales and data analyses. **Quality Progress**, [S.l.], v.40, p.64–65, 07 2007.

ALLEN, R. Formalism and informalism in software architectural style: a case study. In: FIRST INTERNATIONAL WORKSHOP ON ARCHITECTURES FOR SOFTWARE SYSTEMS. **Proceedings...** [S.l.: s.n.], 1995.

ALMEIDA, E. S. de; AHMED, I.; HOEK, A. van der. **Lets Go to the Whiteboard (Again)**: perceptions from software architects on whiteboard architecture meetings. 2022.

AMELLER, D.; FRANCH, X. Assisting software architects in architectural decision-making using Quark. **CLEI Electronic Journal**, [S.l.], v.17, n.3, 2014.

AMPATZOGLOU, A. et al. Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. **Information and software technology**, [S.l.], v.106, p.201–230, 2019.

ARROYO, P.; TOMMELEIN, I.; BALLARD, G. Deciding a sustainable alternative by 'choosing by advantages' in the AEC industry. In: CONF. OF THE INTERNATIONAL GROUP FOR LEAN CONSTRUCTION (IGLC), SAN DIEGO, CA, 20. **Proceedings...** [S.l.: s.n.], 2012. p.41–50.

ARROYO, P.; TOMMELEIN, I.; BALLARD, G. Comparing AHP and CBA as Decision Methods to Resolve the Choosing Problem in Detailed Design. **Journal of Construction Engineering and Management**, [S.l.], v.141, p.04014063, 10 2014.

AWS. **Using architectural decision records to streamline technical decision-making for a software development project**. 2022.

BABAR, M. A. et al. **Software Architecture Knowledge Management**: theory and practice. 1st.ed. [S.l.]: Springer Publishing Company, Incorporated, 2009.

BARAIS, O. et al. **Software Architecture Evolution**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. 233–262p.

BASS, L.; KAZMAN, R.; CLEMENTS, P. **Software Architecture in Practice**. 3.ed. Boston, MA: Addison-Wesley Educational, 2012.

BECK, K. et al. **Manifesto for Agile Software Development**. 2001.

BHAT, A. **Surveys**: what they are, characteristics & examples. 2018.

BHATTACHERJEE, A. **Social science research**: principles, methods, and practices. North Charleston, SC: Createspace Independent Publishing Platform, 2012.

BOEHM, B. Get ready for agile methods, with care. **Computer**, [S.l.], v.35, n.1, p.64–69, 2002.

BOER, R. C. de et al. Architectural Knowledge: getting to the core. In: SOFTWARE ARCHITECTURES, COMPONENTS, AND APPLICATIONS, Berlin, Heidelberg. **Anais...** Springer Berlin Heidelberg, 2007. p.197–214.

BONO, E. de. **Six thinking hats**: an essential approach to business management. New York, NY: Little, Brown Company, 1999.

BOYNTON, P. M.; GREENHALGH, T. Selecting, designing, and developing your questionnaire. **BMJ (Clinical research ed.)**, [S.l.], v.328, n.7451, p.1312–1315, 2004.

BROOKS, F. P. **No Silver Bullet -Essence and Accident in Software Engineering**. 1986.

BROWN, T. Design Thinking. **Harvard business review**, [S.l.], v.86, p.84–92, 141, 07 2008.

BRYMAN, A. **Social Research Methods**. 5.ed. London, England: Oxford University Press, 2015.

BUSCHMANN, F. et al. **Pattern-oriented software architecture**: a system of patterns. Nashville, TN: John Wiley & Sons, 1996.

CARLGREN, L.; RAUTH, I.; ELMQUIST, M. Framing Design Thinking: the concept in idea and enactment. **Creativity and Innovation Management**, [S.l.], v.25, p.38–57, 03 2016.

CARTER, J. **Time to market (TTM)**: 5 ways to reduce it and market quickly. 2023.

CHUN, C. **John Boyd and the "ooda" Loop (great strategists)**. [S.l.]: US Army War College, 2019.

COCKBURN, A. **Hexagonal architecture**. 2005.

CONTENT TEAM, L. **Software design vs. software architecture**. 2022.

COUNCIL, D. **Framework for innovation**. 2023.

CROSS, N. Designerly ways of knowing. **Design Studies**, [S.l.], v.3, n.4, p.221–227, 1982. Special Issue Design Education.

CROSS, N. Expertise in design: an overview. **Design Studies**, [S.l.], v.25, p.427–441, 09 2004.

CUMMINGS, D. B. **Reference Architecture Brief**: software observability. 2023.

DASANAYAKE, S. et al. Software Architecture Decision-Making Practices and Challenges: an industrial case study. In: AUSTRALASIAN SOFTWARE ENGINEERING CONFERENCE, 2015. **Anais...** [S.l.: s.n.], 2015. p.88–97.

DAWES, J. "Do data characteristics change according to the number of scale points used?". **Int. J. Mark. Res.**, [S.l.], v.50, p.61–77, 01 2007.

de Boer, R. C.; van Vliet, H. On the similarity between requirements and architecture. **Journal of Systems and Software**, [S.l.], v.82, n.3, p.544–550, 2009.

DENNE, M.; CLELAND-HUANG, J. **Business of Software Development**. Old Tappan, NJ: Prentice Hall, 2003.

DHADUK, H. **10 best software architecture patterns you must know about**. [S.l.]: Simform, 2020.

DILLMAN, D. A.; SMYTH, J. D.; CHRISTIAN, L. M. **Internet, phone, mail, and mixed mode surveys**: the tailored design method, 4th ed. [S.l.]: Somerset: Wiley, 2014. v.4.

DORST, K.; CROSS, N. Creativity in the design process: co-evolution of problem–solution. **Design Studies**, [S.l.], v.22, n.5, p.425–437, 2001.

DSCHOOL. [S.l.]: Hasso Plattner Institute of Design, 2023.

DUC, A. N.; ABRAHAMSSON, P. Minimum Viable Product or Multiple Facet Product? The Role of MVP in Software Startups. In: AGILE PROCESSES, IN SOFTWARE ENGINEERING, AND EXTREME PROGRAMMING, Cham. **Anais...** Springer International Publishing, 2016. p.118–130.

DUTOIT, A. H. et al. **Rationale Management in Software Engineering**. Berlin, Heidelberg: Springer-Verlag, 2006.

DYBå, T.; DINGSøYR, T. Empirical studies of agile software development: a systematic review. **Information and software technology**, [S.l.], v.50, n.9–10, p.833–859, 2008.

EDEN, A. H.; HIRSHFELD, Y.; KAZMAN, R. Abstraction classes in software design. **IEE proceedings**, [S.l.], v.153, n.4, p.163, 2006.

ENCK, R. E. The OODA loop. **Home health care management & practice**, [S.l.], v.24, n.3, p.123–124, 2012.

FALATIUK, H.; SHIROKOPETLEVA, M.; DUDAR, Z. Investigation of Architecture and Technology Stack for e-Archive System. In: IEEE INTERNATIONAL SCIENTIFIC-PRACTICAL CONFERENCE PROBLEMS OF INFOCOMMUNICATIONS, SCIENCE AND TECHNOLOGY (PIC S&T), 2019. **Anais...** [S.l.: s.n.], 2019. p.229–235.

FALESSI, D. et al. Decision-making techniques for software architecture design: a comparative survey. **ACM computing surveys**, [S.l.], v.43, n.4, p.1–28, 2011.

FARENHORST, R.; LAGO, P.; VLIET, H. van. Prerequisites for successful architectural knowledge sharing. In: AUSTRALIAN SOFTWARE ENGINEERING CONFERENCE (ASWEC'07), 2007. **Anais...** IEEE, 2007.

FELDHAUSEN, R. **THE SOFTWARE CRISIS**. 2020.

FELDT, R.; MAGAZINIUS, A. Validity threats in empirical software engineering research-an initial survey. In: SEKE. **Anais...** [S.l.: s.n.], 2010. p.374–379.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. 2000. Tese (Doutorado em Ciência da Computação) — UNIVERSITY OF CALIFORNIA, IRVINE.

FINK, A. **Survey Research Methods**. [S.l.: s.n.], 2010. p.152–160.

FOWLER, M. **Who needs an architect**. 2003.

GACEK, C. et al. On the definition of software system architecture. In: FIRST INTERNATIONAL WORKSHOP ON ARCHITECTURES FOR SOFTWARE SYSTEMS. **Proceedings...** [S.l.: s.n.], 1995. p.85–94.

GAMMA, E. et al. **Design patterns**: elements of reusable object-oriented software. Boston, MA: Addison Wesley, 1994.

GARLAN, D.; PERRY, D. E. Introduction to the special issue on software architecture. **IEEE transactions on software engineering**, [S.l.], v.21, n.4, p.269–274, 1995.

GARLAN, D.; SHAW, M. **AN INTRODUCTION TO SOFTWARE ARCHITECTURE**. [S.l.: s.n.], 1993. 1-39p.

GARVIN, D. A. Building a Learning Organization. **Harvard business review**, [S.l.], Jul 1993.

GLASS, R. L. **The Software Research Crisis**. 1994.

GROHER, I.; WEINREICH, R. A study on architectural decision-making in context. In: WORKING IEEE/IFIP CONFERENCE ON SOFTWARE ARCHITECTURE, 2015. **Anais...** IEEE, 2015.

GROVES, R. M. et al. **Survey Methodology**. 2.ed. Hoboken, NJ: Wiley-Blackwell, 2009.

HAMMOND, J.; KEENEY, R.; RAIFFA, H. The Hidden Traps in Decision Making. **Clinical laboratory management review : official publication of the Clinical Laboratory Management Association / CLMA**, [S.l.], v.13, p.39–47, 11 1998.

HANNAY, J.; SJøBERG, D.; DYBå, T. A Systematic Review of Theory Use in Software Engineering Experiments. **Software Engineering, IEEE Transactions on**, [S.l.], v.33, p.87 – 107, 03 2007.

HEESCH, U. van; AVGERIOU, P.; HILLIARD, R. A Documentation Framework for Architecture Decisions. **J. Syst. Softw.**, USA, v.85, n.4, p.795–820, apr 2012.

HERBSLEB, J.; MOCKUS, A. An empirical study of speed and communication in globally distributed software development. **IEEE Transactions on Software Engineering**, [S.l.], v.29, n.6, p.481–494, 2003.

HEVNER, A.; CHATTERJEE, S. **Design Research in Information Systems**: theory and practice. 1st.ed. [S.l.]: Springer Publishing Company, Incorporated, 2010.

HGRACA, P. b. **Architectural styles vs. Architectural patterns vs. Design patterns**. 2017.

HGRACA, P. b. **Documenting software architecture**. 2019.

HOTTOIS, J. W. The Spiral of Silence: public opinion—our social skin. by elisabeth noelle-neumann. (chicago: university of chicago press, 1984. pp. xi 184. $20.00.). **American Political Science Review**, [S.l.], v.79, n.3, p.919–920, 1985.

HWANG, C.-L.; YOON, K. **Multiple attribute decision making**: methods and applications a state-of-the-art survey. Berlin, Germany: Springer, 1981.

IMENDA, S. Is There a Conceptual Difference between Theoretical and Conceptual Frameworks? **Journal of Social Sciences**, [S.l.], v.38, p.185–195, 01 2014.

INMON, W. H.; LINSTEDT, D.; LEVINS, M. **Data architecture**: a primer for the data scientist: a primer for the data scientist. 2.ed. San Diego, CA: Academic Press, 2019.

International handbook of thinking and reasoning. 1.ed. London, England: Routledge, 2017.

JABAREEN, Y. Building a Conceptual Framework: philosophy, definitions, and procedure. **International Journal of Qualitative Methods**, [S.l.], v.8, n.4, p.49–62, 2009.

JANSEN, A.; AVGERIOU, P.; van der Ven, J. S. Enriching software architecture documentation. **Journal of Systems and Software**, [S.l.], v.82, n.8, p.1232–1248, 2009. SI: Architectural Decisions and Rationale.

JANSEN, A.; BOSCH, J. Software Architecture as a Set of Architectural Design Decisions. In: WORKING IEEE/IFIP CONFERENCE ON SOFTWARE ARCHITECTURE (WICSA'05), 5. **Anais...** IEEE, 2005.

JAYERATNAM, S. **The delicate balance of network security and performance**. 2022.

JORGENSEN, M. Generalization and theory-building in software engineering research. In: INTERNATION CONFERENCE ON EMPIRICAL ASSESSMENT IN SOFTWARE ENGINEERING (EASE 2004)" WORKSHOP - 26TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 8. **Anais...** IEE, 2004.

JOSHI, A. et al. Likert Scale: explored and explained. **British Journal of Applied Science and Technology**, [S.l.], v.7, p.396–403, 01 2015.

Judgment under uncertainty: heuristics and biases. Cambridge, England: Cambridge University Press, 1982.

KAHNEMAN, D. Maps of bounded rationality: psychology for behavioral economics. **American Economic Review**, [S.l.], v.93, n.5, p.1449–1475, 2003.

KAHNEMAN, D. **Thinking, fast and slow**. [S.l.]: Farrar Straus Giroux, 2011.

KAPOOR, R. **Onion architecture - expedia group technology - medium**. 2022.

KARAKHAN, A.; GAMBATESE, J.; RAJENDRAN, S. Application of choosing by advantages decision-making system to select fall-protection measures. In: ANNUAL CONFERENCE OF THE INTERNATIONAL GROUP FOR LEAN CONSTRUCTION, 24. **Anais...** [S.l.: s.n.], 2016.

KAZMAN, R.; WOODS, S.; CARRIERE, S. Requirements for integrating software architecture and reengineering models: corum ii. In: FIFTH WORKING CONFERENCE ON REVERSE ENGINEERING (CAT. NO.98TB100261). **Proceedings...** [S.l.: s.n.], 1998. p.154–163.

KEENEY, R. L. **Value-focused thinking**: a path to creative decisionmaking. London, England: Harvard University Press, 1996.

KELLEY, T.; LITTMAN, J. **The art of innovation**: lessons in creativity from ideo, america's leading design firm. New York, NY: Bantam Doubleday Dell Publishing Group, 2001.

KIM, C.-K. et al. A lightweight value-based software architecture evaluation. In: EIGHTH ACIS INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, ARTIFICIAL INTELLIGENCE, NETWORKING, AND PARALLEL/DISTRIBUTED COMPUTING (SNPD 2007). **Anais...** IEEE, 2007. v.2, p.646–649.

KITCHENHAM, B. A.; PFLEEGER, S. L. Principles of Survey Research Part 2: designing a survey. **SIGSOFT Softw. Eng. Notes**, New York, NY, USA, v.27, n.1, p.18–20, jan 2002.

KITCHENHAM, B. A.; PFLEEGER, S. L. Principles of Survey Research: part 3: constructing a survey instrument. **SIGSOFT Softw. Eng. Notes**, New York, NY, USA, v.27, n.2, p.20–24, mar 2002.

KITCHENHAM, B.; PFLEEGER, S. L. Principles of survey research part 4: questionnaire evaluation. **ACM SIGSOFT Software Engineering Notes**, [S.l.], v.27, n.3, p.20–23, 2002.

KITCHENHAM, B.; PFLEEGER, S. L. Principles of Survey Research: part 5: populations and samples. **SIGSOFT Softw. Eng. Notes**, New York, NY, USA, v.27, n.5, p.17–20, sep 2002.

KITCHENHAM, B.; PFLEEGER, S. L. Principles of Survey Research Part 6: data analysis. **SIGSOFT Softw. Eng. Notes**, New York, NY, USA, v.28, n.2, p.24–27, mar 2003.

KLEIN, B.; MECKLING, W. Application of operations research to development decisions. **Operations research**, [S.l.], v.6, n.3, p.352–363, 1958.

KLEIN, G. Naturalistic decision making. **Human factors**, [S.l.], v.50, n.3, p.456–460, 2008.

KLEIN, G. **Streetlights and shadows**: searching for the keys to adaptive decision making. [S.l.]: Mit Press, 2009.

KOLKO, J. Abductive Thinking and Sensemaking: the drivers of design synthesis. **Design Issues**, [S.l.], v.26, p.15–28, 12 2010.

KOPP, O.; ARMBRUSTER, A. Generalized Markdown Architectural Decision Records: capturing the essence of decisions (short paper). In: CENTRAL-EUROPEAN WORKSHOP ON SERVICES AND THEIR COMPOSITION. **Anais...** [S.l.: s.n.], 2019.

KOPP, O.; ARMBRUSTER, A.; ZIMMERMANN, O. **Markdown Architectural Decision Records**: format and tool support. [S.l.: s.n.], 2018.

KROSNICK, J. A. Response strategies for coping with the cognitive demands of attitude measures in surveys. **Applied Cognitive Psychology**, [S.l.], v.5, p.213–236, 1991.

KRUCHTEN, P.; NORD, R. L.; OZKAYA, I. Technical debt: from metaphor to theory and practice, software. **IEEE Computer Society**, [S.l.], v.29, n.6, p.18–21, 2012.

LAVRAKAS, P. **Encyclopedia of survey research methods**. 2455 Teller Road, Thousand Oaks California 91320 United States of America: Sage Publications, Inc., 2008.

LEE, L.; KRUCHTEN, P. Capturing Software Architectural Design Decisions. In: CANADIAN CONFERENCE ON ELECTRICAL AND COMPUTER ENGINEERING, 2007. **Anais...** [S.l.: s.n.], 2007. p.686–689.

LEHMAN, M. M.; STENNING, V.; TURSKI, W. M. Another look at software design methodology. **Software engineering notes**, [S.l.], v.9, n.2, p.38–53, 1984.

LETHBRIDGE, T.; SINGER, J.; FORWARD, A. How software engineers use documentation: the state of the practice. **IEEE Software**, [S.l.], v.20, n.6, p.35–39, 2003.

LI, B. et al. Enjoy your observability: an industrial survey of microservice tracing and analysis. **Empirical Software Engineer**, [S.l.], v.27, n.1, 2022.

LIEDTKA, J. Innovative ways companies are using design thinking. **Strategy and leadership**, [S.l.], v.42, n.2, p.40–45, 2014.

LIU, H. H. **Software performance and scalability**: a quantitative approach. [S.l.]: John Wiley & Sons, 2011.

MACCORMACK, A.; VERGANTI, R.; IANSITI, M. Developing products on “Internet time”: the anatomy of a flexible development process. **Management science**, [S.l.], v.47, n.1, p.133–150, 2001.

MALAN, R.; BREDEMEYER, D. **Less is More with Minimalist Architecture**. 2002.

MALLAWAARACHCHI, V. **10 Common Software Architectural Patterns in a nutshell**. 2017.

MARCHAU, V. A. W. J. et al. **Decision making under deep uncertainty**: from theory to practice. 1.ed. Cham, Switzerland: Springer Nature, 2019.

MATZLER, K.; BAILOM, F.; MOORADIAN, T. A. Intuitive decision making. **MIT Sloan Management Review**, [S.l.], v.49, n.1, p.13, 2007.

MEHTA, N. R.; MEDVIDOVIC, N. Composing architectural styles from architectural primitives. **Software engineering notes**, [S.l.], v.28, n.5, p.347, 2003.

MEI, H. A Complementary Approach to Requirements Engineering—software Architecture Orientation. **SIGSOFT Softw. Eng. Notes**, New York, NY, USA, v.25, n.2, p.40–45, mar 2000.

MERRIAM, S. B.; TISDELL, E. J. **Qualitative research**: a guide to design and implementation. 4.ed. [S.l.]: Standards Information Network, 2015.

MERTERNS, D. M.; WILSON, A. T. **Program evaluation theory and practice, second edition**: a comprehensive guide. 2.ed. New York, NY: Guilford Publications, 2018.

MOE, N. B.; AURUM, A.; DYBA, T. Challenges of shared decision-making: a multiple case study of agile software development. **Information and software technology**, [S.l.], v.54, n.8, p.853–865, 2012.

MOEN, R. D.; NORMAN, C. Clearing up myths about the Deming cycle and seeing how it keeps evolving. **Quality Progress**, [S.l.], v.43, p.22–28, 2010.

MOEN, R.; NORMAN, C. **Evolution of the PDCA cycle**. 2009.

MONROE, R. T. et al. Architectural styles, design patterns, and objects. **IEEE software**, [S.l.], v.14, n.1, p.43–52, 1997.

MYERS, S. C. Determinants of corporate borrowing. **Journal of Financial Economics**, [S.l.], v.5, n.2, p.147–175, 1977.

NEDERHOF, A. J. Methods of coping with social desirability bias: a review. **European journal of social psychology**, [S.l.], v.15, n.3, p.263–280, 1985.

NORRIS, N. Error, Bias and Validity in Qualitative Research. **Educational Action Research**, [S.l.], v.5, p.172–176, 03 1997.

NUSEIBEH, B. Weaving together requirements and architectures. **Computer**, [S.l.], v.34, n.3, p.115–119, 2001.

NYGARD, M. **Documenting architecture decisions**. 2011.

ORLOV, S.; VISHNYAKOV, A. Decision Making for the Software Architecture Structure Based on the Criteria Importance Theory. **Procedia Computer Science**, [S.l.], v.104, p.27–34, 2017. ICTE 2016, Riga Technical University, Latvia.

OROSZ, G. **Software Architecture is Overrated, Clear and Simple Design is Underrated**. 2019.

PAULA, R. J. d.; FALVOJR, V. **Architectural Patterns and Styles**. 2016.

PERRY, D. E.; WOLF, A. L. Foundations for the study of software architecture. **Software engineering notes**, [S.l.], v.17, n.4, p.40–52, 1992.

PFLEEGER, S.; KITCHENHAM, B. Principles of survey research: part 1: turning lemons into lemonade. **ACM SIGSOFT Software Engineering Notes**, [S.l.], v.26, p.16–18, 01 2001.

PHILLIPS, W. J. et al. Thinking styles and decision making: a meta-analysis. **Psychological bulletin**, [S.l.], v.142, n.3, p.260–290, 2016.

PIETRZAK, M.; PALISZKIEWICZ, J. Framework of Strategic Learning: pdca cycle. **Management**, [S.l.], v.10, p.149–161, 01 2015.

PLANTROU, G. **Microservices are becoming the default application**. 2022.

PLATTNER, H.; MEINEL, C.; LEIFER, L. **Design thinking research**: making design thinking foundational. 1.ed. [S.l.]: Springer International Publishing, 2016.

PRETORIUS, C. Beyond Reason: uniting intuition and rationality in software architecture decision making. In: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ARCHITECTURE COMPANION (ICSA-C), 2019. **Anais...** [S.l.: s.n.], 2019. p.275–282.

PRETORIUS, C. et al. Towards a dual processing perspective of software architecture decision making. In: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ARCHITECTURE COMPANION (ICSA-C), 2018. **Anais...** IEEE, 2018. p.48–51.

PRICE, J.; MURNAN, J. Research Limitations and the Necessity of Reporting Them. **American Journal of Health Education**, [S.l.], v.35, p.66–67, 04 2004.

PROVOST, F.; FAWCETT, T. Data Science and Its Relationship to Big Data and Data-Driven Decision Making. **Big Data**, [S.l.], v.1, 03 2013.

RAJARAM, G. **Gokul's S.p.a.d.e. toolkit**. 2020.

RALPH, P.; WAND, Y. A Proposal for a Formal Definition of the Design Concept. In: DESIGN REQUIREMENTS ENGINEERING: A TEN-YEAR PERSPECTIVE, Berlin, Heidelberg. **Anais...** Springer Berlin Heidelberg, 2009. p.103–136.

RAMESH, B.; CAO, L.; BASKERVILLE, R. Agile requirements engineering practices and challenges: an empirical study. **Inf. Syst. J.**, [S.l.], v.20, p.449–480, 09 2010.

REA, L. M.; PARKER, R. A. **Designing and conducting survey research**: a comprehensive guide. 4.ed. London, England: Jossey-Bass, 2014.

REINERTSEN, D. G. **The principles of product development flow**: second generation lean product development. [S.l.]: Celeritas Publishing, 2009.

REKHAV, V. S.; MUCCINI, H. A study on group decision-making in software architecture. In: IEEE/IFIP CONFERENCE ON SOFTWARE ARCHITECTURE, 2014. **Anais...** [S.l.: s.n.], 2014. p.185–194.

REVILLA, M. et al. Do online access panels need to adapt surveys for mobile devices? **Internet Research**, [S.l.], v.26, p.1209–1227, 10 2016.

RIES, E. (Ed.). **The Lean Startup**: how today's entrepreneurs use continuous innovation to create radically successful businesses. [S.l.]: Crown Currency, 2011.

RUEL, E.; WAGNER, W.; GILLESPIE, B. **The Practice of Survey Research**: theory and applications. [S.l.: s.n.], 2016.

SAM, D. **Twitter's tough architectural decision**. 2022.

SANTOS, L. et al. An architectural style for internet of things systems. In: ANNUAL ACM SYMPOSIUM ON APPLIED COMPUTING, 35., New York, NY, USA. **Proceedings...** ACM, 2020.

SCHRIEK, C. et al. Software Architecture Design Reasoning: a card game to help novice designers. In: SOFTWARE ARCHITECTURE, Cham. **Anais...** Springer International Publishing, 2016. p.22–38.

SEI. **What is your definition of software architecture**. [S.l.]: Software Engineering Institute, 2010.

SENECKI, A.; GOIK, R. **Hexagonal architecture – is it for me? A no-nonsense overview**. 2023.

SENGE, P. M. **The fifth discipline fieldbook**: strategies and tools for building a learning organization. [S.l.]: Crown Business, 2014.

SHARMA, A.; KUMAR, M.; AGARWAL, S. A complete survey on software architectural styles and patterns. **Procedia computer science**, [S.l.], v.70, p.16–28, 2015.

SHAW, M.; GARLAN, D. **Software Architecture**: perspectives on an emerging discipline. Upper Saddle River, NJ: Pearson, 1996.

SHENTON, A. Strategies for Ensuring Trustworthiness in Qualitative Research Projects. **Education for Information**, [S.l.], v.22, p.63–75, 07 2004.

SIMON, H. A. A Behavioral Model of Rational Choice. **The Quarterly Journal of Economics**, [S.l.], v.69, n.1, p.99–118, 1955.

EATWELL, J.; MILGATE, M.; NEWMAN, P. (Ed.). **Bounded Rationality**. London: Palgrave Macmillan UK, 1990. 15–18p.

SIMON, H. **The Sciences of the Artificial**. [S.l.]: The MIT Press, 1996.

SJøBERG, D. I. K. et al. **Building Theories in Software Engineering**. London: Springer London, 2008. 312–336p.

SLOVIC, P.; LICHTENSTEIN, S.; FISCHHOFF, B. Decision making. In: **Steven's Handbook of Experimental Psychology**. [S.l.]: Wiley, 1988.

SMITH, C. U.; WILLIAMS, L. G. Performance and scalability of distributed software architectures: an spe approach. **Parallel and Distributed Computing Practices**, [S.l.], v.3, n.4, p.74106–0700, 2002.

SMYTH, J. et al. Comparing Check-All and Forced-Choice Question Formats in Web Surveys. **Public Opinion Quarterly**, [S.l.], v.70, 03 2006.

SOC, D. **Y-statements**. 2020.

STEWART, R.; BENEPE, O.; MITCHELL, A. Formal Planning: the staff planner's role at start up (no. 250). **California: Stanford Research Institute**, [S.l.], 1965.

STOBIERSKI, T. **The advantages of data-driven decision-making**. 2019.

SUHR, J. Basic principles of sound decisionmaking. **BIOGRAPHY**, [S.l.], v.801, p.782–6168, 2000.

TANG, A. et al. A survey of architecture design rationale. **The Journal of systems and software**, [S.l.], v.79, n.12, p.1792–1804, 2006.

TANG, A. et al. Human aspects in software architecture decision making: a literature review. In: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ARCHITECTURE (ICSA), 2017. **Anais...** IEEE, 2017. p.107–116.

TANG, A.; KAZMAN, R. Decision-Making Principles for Better Software Design Decisions. **IEEE Software**, [S.l.], v.38, n.6, p.98–102, 2021.

TANG, A.; LIANG, P.; VLIET, H. v. Software Architecture Documentation: the road ahead. In: NINTH WORKING IEEE/IFIP CONFERENCE ON SOFTWARE ARCHITECTURE, 2011. **Anais...** [S.l.: s.n.], 2011. p.252–255.

TAYLOR, R. N.; MEDVIDOVIC, N.; DASHOFY, E. **Software architecture**: foundations, theory, and practice. Chichester, England: John Wiley & Sons, 2008.

TEAM, I. E. **What are decision-making techniques and how do they work?** 2022.

TECH, C. **10 Architecture Patterns Used In Enterprise Software Development Today**. 2021.

THOMKE, S.; REINERTSEN, D. Six myths of product development. **Harvard business review**, [S.l.], May 2012.

THOMPSON, V. A.; TURNER, J. A. P.; PENNYCOOK, G. Intuition, reason, and metacognition. **Cognitive Psychology**, [S.l.], v.63, p.107–140, 2011.

TIM, B. **Change by Design**: how design thinking transforms organizations and inspires innovation. [S.l.]: HarperBusiness, 2009.

TRAN, H. et al. **Sustainable architectural design decisions**. 2014.

TRIANTAPHYLLOU, E.; BAIG, K. The impact of aggregating benefit and cost criteria in four MCDA methods. **IEEE Transactions on Engineering Management**, [S.l.], v.52, n.2, p.213–226, 2005.

TYREE, J.; AKERMAN, A. Architecture decisions: demystifying architecture. **IEEE software**, [S.l.], v.22, n.2, p.19–27, 2005.

VALIPOUR, M. H. et al. A brief survey of software architecture concepts and service oriented architecture. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE AND INFORMATION TECHNOLOGY, 2009. **Anais...** IEEE, 2009.

VASSILEV, V.; GENOVA, K.; VASSILEVA, M. A brief survey of multicriteria decision making methods and software systems. **Cybernetics and information technologies**, [S.l.], v.5, n.1, p.3–13, 2005.

VLIET, H. van; TANG, A. Decision making in software architecture. **The Journal of systems and software**, [S.l.], v.117, p.638–644, 2016.

VON NEUMANN, J.; MORGENSTERN, O. **Theory of games and economic behavior**: 60th anniversary commemorative edition. Princeton, NJ: Princeton University Press, 2007.

WALKER, V. **14 software architecture design patterns to know**. [S.l.]: Red Hat, Inc., 2022.

WEINREICH, R.; GROHER, I.; MIESBAUER, C. An expert survey on kinds, influence factors and documentation of design decisions in practice. **Future generations computer systems: FGCS**, [S.l.], v.47, p.145–160, 2015.

WIERSMA, W. The validity of surveys: online and offline. **Oxf. Internet Inst**, [S.l.], v.18, n.3, p.321–340, 2013.

WILLIAMS, O. **Fundamental software architectural patterns**. 2022.

WOHLIN, C. et al. **Experimentation in software engineering**. 2012.ed. Berlin, Germany: Springer, 2012.

WRIGHT, H. K.; KIM, M.; PERRY, D. E. Validity concerns in software engineering research. In: FSE/SDP WORKSHOP ON FUTURE OF SOFTWARE ENGINEERING RESEARCH - FOSER '10, New York, New York, USA. **Proceedings...** ACM Press, 2010.

XU, L. et al. An architectural pattern for non-functional dependability requirements. **The Journal of systems and software**, [S.l.], v.79, n.10, p.1370–1378, 2006.

ZANNIER, C.; CHIASSON, M.; MAURER, F. A model of design decision making based on empirical results of interviews with software designers. **Information and software technology**, [S.l.], v.49, n.6, p.637–653, 2007.

ZHU, H. **Software design methodology**: from principles to architectural styles. Oxford, England: Butterworth-Heinemann, 2005.

ZIMMERMANN, A.; LORENZ, A.; OPPERMANN, R. An Operational Definition of Context. In: MODELING AND USING CONTEXT. **Anais...** [S.l.: s.n.], 2007. v.4635, p.558–571.

ZIMMERMANN, O. **Architectural Decisions — The Making Of**. 2020.

ZIMMERMANN, O.; MIKSOVIC, C.; KüSTER, J. **Reference architecture, metamodel, and modeling principles for architectural knowledge management in information technology services**. [S.l.: s.n.], 2012. v.85, p.2014–2033.

Appendix

A

Questionnaire Initial Version

Tomada de Decisão Arquitetural

Caracterização da Questionário

O processo de tomada de decisão arquitetural é algo inerente a atividade de desenvolvimento de software. É através dele na qual o software é organizado, descrevendo os componentes internos, protocolos utilizados, interfaces disponíveis e formato de resposta esperado.

O objetivo dessa pesquisa é entender como os praticantes (engenheiros de software, arquitetos, desenvolvedores senior, dentre outros) fazem decisões arquiteturais, quais os princípios/motivos subjacentes as escolhas feitas e quais fatores influenciam positivamente/negativamente nesse processo.

Os resultados dessa pesquisa são importantes para melhorar o entendimento sobre o processo de tomada de decisão arquitetural. Portanto a colaboração de cada participante é crucial nesse sentido.

O tempo médio para responder esse questionário é de 4 a 5 minutos.

Essa pesquisa está sendo realizada no contexto da Justiça Federal com base nos profissionais que atuam ou prestam serviço diretamente aos órgãos relacionados (TRF's). Ela será utilizada como instrumento de pesquisa para elaboração da dissertação de mestrado do programa de Pós-Graduação do Centro de Informática da Universidade Federal de Pernambuco (CIn-UFPE) em Ciências da Computação.

O que preciso fazer?

Concorde com sua participação marcando a caixa de seleção “Concordo” abaixo para iniciar o questionário online. Somente os questionários enviados ao clicar no botão "Concluir" na página final serão salvos.

Sobre a privacidade dos dados

Sua participação no estudo será confidencial. Este questionário não coleta informações de identificação pessoal. Os dados do estudo serão totalmente anonimizados no ponto de coleta. Isso significa que não conectamos nenhum nome ou endereço de email às respostas que você enviar. Elas receberão apenas um número de identificação (por exemplo, ID 21). Portanto, não será possível identificar você pelo nome a partir de qualquer aspecto da documentação ou relatório para este estudo de pesquisa. Ao final do estudo, seus dados serão “Dados Abertos”. Isso significa que eles serão armazenados em um banco de dados online e disponibilizados ao público.

Contato

Em caso de qualquer dificuldade durante o processo de preenchimento, favor em entrar em contato através desse e-mail.

*Obrigatório

Confirme que você concorda com os termos e condições acima para a pesquisa. *

- ◯ Concordo.
- ◯ Não concordo.

Dados demográficos

Formulário de coleta de dados demográficos dos participantes da pesquisa.

Qual sua idade?

- ◯ Menos de 20 anos
- ◯ 20 a 30 anos
- ◯ 31 a 40 anos
- ◯ 41 a 50 anos
- ◯ Mais de 50 anos

Há quantos anos você trabalha na área de tecnologia da informação/TI?

- ◯ Menos de 1 ano
- ◯ 1 a 5 anos
- ◯ 6 a 10 anos
- ◯ 11 a 15 anos
- ◯ Mais de 15 anos

Qual sua função na empresa onde vc trabalha? *

- ○ Engenheiro(a) de Software
- ○ Arquiteto(a) de Software
- ○ Desenvolvedor(a) Senior
- ○ Líder de time
- ○ Gerente de Projeto
- ○ Outro:

Qual o tamanho da sua equipe de trabalho?

- ○ 1 - 10
- ○ 11 - 25
- ○ 26 - 50
- ○ 50 - 100
- ○ Mais de 100

Você é responsável pelas decisões arquiteturais no seu local de trabalho? *

- ○ Sim
- ○ Não

Tomada de Decisão Aquitetural

Seção relativa ao processo de tomada de decisão arquitetural.

Há quantos anos você trabalha fazendo decisões arquiteturais? *

Sua resposta

Em qual fase do projeto são tomadas as decisões arquiteturais? *

- ○ Fase inicial do projeto
- ○ Durante as iterações ou sprints
- ○ Qualquer momento
- ○ Outro:

Como as decisões arquiteturais são tomadas? *

- ○ Individualmente
- ○ Decisão em grupo
- ○ Outro

Descreva como são tomadas as decisões arquiteturais onde você trabalha.

Caso você tenha marcado, a opção "Outro" na pergunta anterior descreva como são tomadas as decisões arquiteturais onde você trabalha.

Sua resposta

O quão confiante você é ao tomar uma decisão arquitetural? *

Nem um pouco confiante

1 ○

2 ○

3 ○

4 ○

5 ○

Extremamente confiante

Ao tomar uma decisão arquitetural, você procura eleger mais uma solução na sua mente? *

Nunca

1 ○

2 ○

3 ○

4 ○

5 ○

Sempre

Você utiliza alguma ferramenta de apoio a tomada de decisão arquitetural? Se sim, qual? *

Sua resposta

Como são documentadas as decisões arquiteturais tomadas? *

- [ ] Wiki
- [ ] Any/Architectural Decision Record (ADR)
- [ ] Y's Statements
- [ ] Documento de Texto (Word, Text, Google Docs)
- [ ] Código-fonte (Gitlab, Github, DCVS, CVS, etc.)
- [ ] Open Decision Framework (ODF)
- [ ] Outro:

Na sua opinião, quais os potenciais fatores que influenciam o processo de tomada de decisão arquitetural? *

Descreva os fatores mais importantes a serem considerados durante o processo de tomada de decisão arquitetural.

Sua resposta

As decisões arquiteturais tomadas são revisitadas durante o tempo de vida do projeto? *

Nunca

- 1 ○
- 2 ○
- 3 ○
- 4 ○
- 5 ○

Sempre

Quais são as dificuldades encontradas durante o processo de tomada de decisão arquitetural? *

Descreva os elementos dificultadores do processo de tomada de decisão arquitetural.

Sua resposta

B

Questionnaire Final Version

Tomada de Decisão Arquitetural

Caracterização da Questionário

O processo de tomada de decisão arquitetural é algo inerente a atividade de desenvolvimento de software. É através dele na qual o software é organizado, descrevendo os componentes internos, protocolos utilizados, interfaces disponíveis e formato de resposta esperado.

O objetivo dessa pesquisa é entender como os praticantes (engenheiros de software, arquitetos, desenvolvedores senior, dentre outros) tomam e documentam as decisões arquiteturais, quais os princípios/motivos subjacentes as escolhas feitas e quais fatores influenciam positivamente/negativamente nesse processo.

Os resultados dessa pesquisa são importantes para melhorar o entendimento sobre o processo de tomada de decisão arquitetural. Portanto a colaboração de cada participante é crucial nesse sentido.

O tempo médio para responder esse questionário é de 4 a 5 minutos.

Essa pesquisa está sendo realizada no contexto brasileiro com base nos profissionais que trabalham ou atuam com desenvolvimento de software seja em empresas privadas ou públicas. Ela será utilizada como instrumento de pesquisa para elaboração da dissertação de mestrado do programa de Pós-Graduação do Centro de Informática da Universidade Federal de Pernambuco (CIn-UFPE) em Ciências da Computação.

O que preciso fazer?

Concorde com sua participação marcando a caixa de seleção “Concordo” abaixo para iniciar o questionário online. Somente os questionários enviados ao clicar no botão "Concluir" na página final serão salvos.

Sobre a privacidade dos dados

Sua participação no estudo será confidencial. Este questionário não coleta informações de identificação pessoal. Os dados do estudo serão totalmente anonimizados no ponto de coleta. Isso significa que não conectamos nenhum nome ou endereço de email às respostas que você enviar. Elas receberão apenas um número de identificação (por exemplo, ID 21). Portanto, não será possível identificar você pelo nome a partir de qualquer aspecto da documentação ou relatório para este estudo de pesquisa. Ao final do estudo, os dados anonimizados serão “Dados Abertos”. Isso significa que eles serão armazenados em um banco de dados online e disponibilizados ao público.

Contato

Em caso de qualquer dificuldade durante o processo de preenchimento, favor em entrar em contato através desse e-mail.

* Indica uma pergunta obrigatória

1. Confirme que você concorda com os termos e condições acima para a pesquisa. *

   Marcar apenas uma oval.

   - ◯ Concordo. *Pular para a pergunta 2*
   - ◯ Não concordo.

Pular para a pergunta 2

Dados demográficos

Formulário de coleta de dados demográficos dos participantes da pesquisa.

2. Qual sua idade?

Marcar apenas uma oval.

- ◯ Menos de 20 anos
- ◯ 20 a 30 anos
- ◯ 31 a 40 anos
- ◯ 41 a 50 anos
- ◯ Mais de 50 anos

3. Há quantos anos você trabalha na área de tecnologia da informação/TI?

Marcar apenas uma oval.

- ◯ Menos de 1 ano
- ◯ 1 a 5 anos
- ◯ 6 a 10 anos
- ◯ 11 a 15 anos
- ◯ Mais de 15 anos

4. Qual sua função na empresa onde vc trabalha? *

Marcar apenas uma oval.

- ◯ Engenheiro(a) de Software
- ◯ Arquiteto(a) de Software
- ◯ Desenvolvedor(a) Senior
- ◯ Líder de time
- ◯ Gerente de Projeto
- ◯ Outro: _____

5. Qual o tamanho da sua equipe de trabalho?

Marcar apenas uma oval.

- ◯ 1 - 10
- ◯ 11 - 25
- ◯ 26 - 50
- ◯ 50 - 100
- ◯ Mais de 100

6. Qual seu grau de educação formal?

Marcar apenas uma oval.

- ◯ Estudante de Graduação
- ◯ Graduação
- ◯ Pós-Graduação / MBA
- ◯ Mestrado
- ◯ Doutorado / PhD
- ◯ Outro: _____

7. Você é responsável pelas decisões arquiteturais no seu local de trabalho? *

Marcar apenas uma oval.

- ◯ Sim *Pular para a pergunta 8*
- ◯ Não

Pular para a pergunta 8

Tomada de Decisão Aquitetural

Seção relativa ao processo de tomada de decisão arquitetural.

8. Há quantos anos você trabalha tomando e/ou documentando decisões arquiteturais? *

Marcar apenas uma oval.

- ◯ 1 a 2 anos
- ◯ 3 a 4 anos
- ◯ 5 a 10 anos
- ◯ 10 a 20 anos
- ◯ Mais de 20 anos

9. Em qual fase do projeto são tomadas as decisões arquiteturais? *

Marcar apenas uma oval.

- ◯ Fase inicial do projeto
- ◯ Durante as iterações ou sprints
- ◯ Qualquer momento
- ◯ Outro: _____

10. Como as decisões arquiteturais são tomadas? *

Marcar apenas uma oval.

- ◯ Individualmente
- ◯ Decisão em grupo
- ◯ Outro: _____

11. Descreva como são tomadas as decisões arquiteturais onde você trabalha.

Caso você tenha marcado, a opção "Outro" na pergunta anterior descreva como são tomadas as decisões arquiteturais onde você trabalha.

_____

_____

_____

_____

_____

12. O quão confiante você é ao tomar uma decisão arquitetural? *

Marcar apenas uma oval.

Nem um pouco confiante

1 ◯
2 ◯
3 ◯
4 ◯
5 ◯

Extremamente confiante

13. Ao tomar uma decisão arquitetural, você procura eleger mais uma solução na sua mente? *

Marcar apenas uma oval.

Nunca

1 ◯
2 ◯
3 ◯
4 ◯
5 ◯

Sempre

14. Você utiliza alguma ferramenta de apoio a tomada de decisão arquitetural? *

Marcar apenas uma oval.

◯ Não
◯ Sim

15. Como são documentadas as decisões arquiteturais tomadas? *

Marque todas que se aplicam.

- [ ] Wiki
- [ ] Any/Architectural Decision Record (ADR)
- [ ] Y's Statements
- [ ] Documento de Texto (Word, Text, Google Docs)
- [ ] Código-fonte (Gitlab, Github, DCVS, CVS, etc.)
- [ ] Open Decision Framework (ODF)
- [ ] E-mails
- [ ] Sistema de Gerenciamento de Problemas (Redmine, JIRA, Backlog, etc.)
- [ ] Outro: _____

16. Quão importante você considera documentar decisões arquiteturais tomadas? *

Marcar apenas uma oval.

Nem um pouco importante

1 ◯
2 ◯
3 ◯
4 ◯
5 ◯

Extremamente importante

17. As decisões arquiteturais tomadas são revisitadas durante o tempo de vida do projeto? *

Marcar apenas uma oval.

Nunca

1 ◯
2 ◯
3 ◯
4 ◯
5 ◯

Sempre

18. Indique o quão importante são os fatores listados abaixo para o processo de tomada de decisão arquitetural. *

Marcar apenas uma oval por linha.

	Nenhum Pouco Importante	Pouco Importante	Moderadamente Importante	Muito Importante	Extremamente Importante	Não se aplica
Tamanho da Empresa	◯	◯	◯	◯	◯	◯
Fatores Negociais (Domínio, Modelo, Custo/Risco/Tempo, Estratégia de Negócio)	◯	◯	◯	◯	◯	◯
Fatores Organizacionais (Tamanho do Time, Organização do Time, Processos e Práticas, Padrões e Restrições)	◯	◯	◯	◯	◯	◯
Fatores Técnicos (Princípios e Padrões, Restrições)	◯	◯	◯	◯	◯	◯
Fatores Culturais	◯	◯	◯	◯	◯	◯
Fatores Individuais	◯	◯	◯	◯	◯	◯
Fatores de Projeto (Tipo e Duração do Projeto)	◯	◯	◯	◯	◯	◯
Escopo da Decisão	◯	◯	◯	◯	◯	◯
Atributos de Qualidade	◯	◯	◯	◯	◯	◯
Requisitos do Usuário	◯	◯	◯	◯	◯	◯
Literatura existente	◯	◯	◯	◯	◯	◯
Ferramentas e Tecnologia Disponíveis	◯	◯	◯	◯	◯	◯

19. Indique o quão impactante são as dificuldades listadas durante o processo de tomada de decisão arquitetural. *

Marcar apenas uma oval por linha.

	Nenhum impacto	Pouco impacto	Impacto moderado	Impacto elevado	Impacto muito elevado	Não se aplica
Falta de clareza nos requisitos	◯	◯	◯	◯	◯	◯
Requisitos Não Funcionais Conflitantes (ex.: Desempenho x Segurança)	◯	◯	◯	◯	◯	◯
Prazo de entrega insuficiente	◯	◯	◯	◯	◯	◯
Conflito entre Stakeholders	◯	◯	◯	◯	◯	◯
Obrigações Contratuais Legais (ex.: LGPD)	◯	◯	◯	◯	◯	◯
Pouca Familiariadade com Domínio de Negócio	◯	◯	◯	◯	◯	◯

20. Indique o quão importante são os princípios abaixo para o processo de tomada de decisão arquitetural. *

Marcar apenas uma oval por linha.

	Nenhum Pouco Importante	Pouco Importante	Moderamente Importante	Muito Importante	Extremamente Importante	Não se aplica
Utilizar fatos ao invés de suposições	◯	◯	◯	◯	◯	◯
Verificar adequação da solução com o problema	◯	◯	◯	◯	◯	◯
Checar prós e contras da solução selecionada	◯	◯	◯	◯	◯	◯
Escolher a solução de acordo com as limitações do sistema	◯	◯	◯	◯	◯	◯
Pensar em mais de uma solução arquitetural para o problema	◯	◯	◯	◯	◯	◯
Decidir respeitando o tempo de entrega do sistema	◯	◯	◯	◯	◯	◯
Definir prioridades do sistema	◯	◯	◯	◯	◯	◯
Antecipar possíveis riscos na escolha da solução	◯	◯	◯	◯	◯	◯