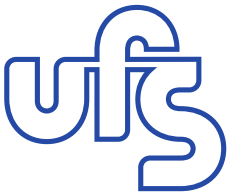

FEDERAL UNIVERSITY SERGIPE

CENTER FOR EXACT SCIENCES AND TECHNOLOGY

POSTGRADUATE PROGRAM IN COMPUTER SCIENCE

ISAA: An Iterative Framework for Software Architecture Assessment

Master's Dissertation

Gustavo dos Santos Melo

São Cristóvão – Sergipe

2026

FEDERAL UNIVERSITY SERGIPE

CENTER FOR EXACT SCIENCES AND TECHNOLOGY

POSTGRADUATE PROGRAM IN COMPUTER SCIENCE

Gustavo dos Santos Melo

ISAA: An Iterative Framework for Software Architecture Assessment

Master's Dissertation submitted to the Graduate Program in Computer Science at the Federal University of Sergipe as a partial requirement for obtaining the degree of master in Computer Science.

Advisor: Michel dos Santos Soares

São Cristóvão – Sergipe

2026

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL
UNIVERSIDADE FEDERAL DE SERGIPE

Melo, Gustavo dos Santos
M528i ISAA: an iterative framework for software architecture assessment / Gustavo dos Santos Melo ; orientador Michel dos Santos Soares. - São Cristóvão, 2026.
77 f.; il.

Dissertação (mestrado em Ciência da Computação) – Universidade Federal de Sergipe, 2026.

1. Computação. 2. Arquitetura de software. I. Soares, Michel dos Santos orient. II. Título.

CDU 004.273

Abstract

Software architecture plays a decisive role for success of software projects, influencing critical aspects such as scalability, performance, reliability, and maintainability. Systematic assessment of software architectures allows development teams to identify risks early, validate quality attributes, and ensure alignment with business objectives throughout the software development lifecycle. Over the years, different approaches have emerged to guide this process, notably the Architectural Tradeoff Analysis Method (ATAM) and the ISO/IEC/IEEE 42020:2019 standard. While ATAM offers a structured methodology for identifying and balancing tradeoffs between architectural decisions, the ISO/IEC/IEEE 42020:2019 standard introduces a comprehensive governance model that encompasses the management, assessment, and conceptualization of architectures in various contexts. This dissertation presents a comparative analysis between ATAM and ISO/IEC/IEEE 42020:2019, highlighting their evolution, complementarities, and the new architectural concerns that have emerged over the past two decades. The comparison reveals that ISO/IEC/IEEE 42020:2019 expands traditional assessment practices by incorporating processes for continuous architectural governance and lifecycle integration, reflecting contemporary demands in Software Engineering. The broader scope of the standard does not diminish the relevance of specialized methods like ATAM; rather, it points to a complementary relationship in which targeted assessment techniques can be incorporated into more comprehensive frameworks for greater applicability in industry and academia. Based on the comparative study of ATAM and ISO/IEC/IEEE:42020:2019, the Iterative Software Architecture Assessment (ISAA) framework is proposed as an iterative and hybrid methodology, based on clause 9 of the ISO/IEC/IEEE 42020:2019 standard. ISAA aims to operationalize architectural assessments by providing structured and repeatable procedures that facilitate documentation and communication among stakeholders, including architects, developers, product owners, and managers, across all development phases. The framework's design is grounded in a literature review, an analysis of established methods, and empirical validation with software industry professionals. The results suggest that ISAA's effectiveness depends heavily on organizational maturity, contextual adaptability, and achieving an appropriate balance between methodological rigor and agility.

Keywords: ATAM. ISO/IEC/IEEE 42020:2019. Software Architecture Evaluation. Survey.

UNIVERSIDADE FEDERAL DE SERGIPE
PRÓ-REITORIA DE PÓS-GRADUAÇÃO E PESQUISA
COORDENAÇÃO DE PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Ata da Sessão Solene de Defesa da Dissertação do Curso de Mestrado em Ciência da Computação-UFS.
Candidato: GUSTAVO DOS SANTOS MELO

Em 30 dias do mês de janeiro do ano de dois mil e vinte seis, com início às 09:00hs, realizou-se na Sala de Seminários do PROCC da Universidade Federal de Sergipe, na Cidade Universitária Prof. José Aloísio de Campos, a Sessão Pública de Defesa de Dissertação de Mestrado do candidato **GUSTAVO DOS SANTOS MELO** que desenvolveu o trabalho intitulado: **"ISAA: An Iterative Framework for Software Architecture Assessment",** sob a orientação do Prof. Dr. **Michel dos Santos Soares.** A Sessão foi presidida pelo Prof. Dr. **Michel dos Santos Soares** (PROCC/UFS), que após a apresentação da dissertação passou a palavra aos outros membros da Banca Examinadora, a Dra. **Joyce Meire da Silva França (IFMG)** e, em seguida, Dr. **Fábio Gomes Rocha** (PROCC/UFS). Após as discussões, a Banca Examinadora reuniu-se e considerou o mestrando (a) **APROVADO** "*(aprovado/reprovado)*". Atendidas as exigências da Instrução Normativa 05/2019/PROCC, do Regimento Interno do PROCC (Resolução 67/2014/CONEPE), e da Resolução nº 04/2021/CONEPE que regulamentam a Apresentação e Defesa de Dissertação, e nada mais havendo a tratar, a Banca Examinadora elaborou esta Ata que será assinada pelos seus membros e pelo mestrando.

Cidade Universitária "*Prof. José Aloísio de Campos",* 27 de janeiro de 2026.

Documento assinado digitalmente
gov.br MICHEL DOS SANTOS SOARES
Data: 19/02/2026 14:26:05-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Michel dos Santos Soares
(PROCC/UFS)
Presidente

Documento assinado digitalmente
gov.br FABIO GOMES ROCHA
Data: 25/02/2026 09:01:11-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Fábio Gomes Rocha
(PROCC/UFS)
Examinador Interno

Documento assinado digitalmente
gov.br JOYCE MEIRE DA SILVA FRANCA
Data: 20/02/2026 02:50:33-0300
Verifique em https://validar.iti.gov.br

Prof. Dra. **Joyce Meire da Silva França**
(IFMG)
Examinador Externo ao programa

Documento assinado digitalmente
gov.br GUSTAVO DOS SANTOS MELO
Data: 25/02/2026 12:13:15-0300
Verifique em https://validar.iti.gov.br

Gustavo dos Santos Melo
Candidato

List of Figures

List of Tables

Contents

1

Introduction

The evolution of Software Engineering in recent decades has brought with it increasingly complex challenges related to the design, maintenance, and evolution of computer systems. The growing demand for robust, secure, and scalable technological solutions, driven by digital transformation and global competitiveness, has placed Software Architecture as a central element in the development lifecycle. The concept of time-to-market, widely discussed since the early 2000s, remains current and relevant, as the success of a software product is directly linked to its ability to respond quickly and efficiently to industry demands without compromising its internal quality. In this context, the definition and evaluation of Software Architecture emerge as fundamental practices for achieving technical and strategic excellence in software development.

Software Architecture can be understood as the fundamental organization of a software system, encompassing one or more structures that describe its components, their externally visible properties, and the relationships among them (BASS; CLEMENTS; KAZMAN, 2003). Structural abstractions derived from this organization enable engineers to comprehend, communicate, and control the complexity inherent in contemporary software development. Beyond a static representation of software structure, Software Architecture incorporates strategic design decisions that influence essential quality attributes, including performance, scalability, maintainability, security, and usability. As a result, poor or uninformed architectural decisions may critically undermine the system's quality, sustainability, and long-term viability (van Vliet; TANG, 2016; COSTA; SOARES, 2023a).

Software Architecture assessment plays a crucial role in enabling developers to verify the alignment of architectural decisions with both functional and non-functional requirements, as well as with organizational and business goals. Continuous assessment of Software Architectures throughout the software lifecycle facilitates early identification of risks, validation of quality requirements, and prevention of defects that may escalate into critical issues if left unresolved (KAUR; KHURANA; MANISHA, 2021; SILVA et al., 2023). Such an iterative process effectively

mitigates technical risks while simultaneously strengthening strategic planning and governance within Software Engineering.

Furthermore, architectural assessment should not be viewed as an isolated or merely technical process. On the contrary, it involves collaboration between multiple stakeholders, including architects, developers, quality analysts, product managers, and organizational stakeholders. Previous studies (FRANÇA; LIMA; SOARES, 2017; SANTOS; MISRA; SOARES, 2020) highlight that effective communication between participants is a critical factor for the success of the assessment. Documenting decisions, representing layers and components, and recording patterns and justifications are practices that promote transparency, traceability, and collective learning indispensable aspects for the maintenance and evolution of complex software.

Specialized literature presents methods and international standards developed to support architectural assessment activities. Among the most relevant are the ATAM (Architecture Tradeoff Analysis Method), created by Kazman et al., and the ISO/IEC/IEEE 42020:2019 standard, which establishes modern principles and guidelines for Systems and Software Architecture processes. The former stands out for proposing a systematic approach for analyzing tradeoffs between quality attributes, while the latter presents a broader normative framework focused on integrating the architectural process into the organizational lifecycle. Both are significantly relevant and widely used in industrial and academic settings, although each has limitations and specific application contexts (KAZMAN et al., 2000).

In this sense, understanding the similarities, differences, and relationships between methods and patterns is essential for the advancement of contemporary architectural practices. The need to combine theoretical approaches with practical and adaptable methodologies has motivated research that seeks to propose new models, frameworks, and tools that make the evaluation process more dynamic, continuous, and accessible to different maturity levels of development teams. Therefore, this dissertation begins with a broad reflection on the importance of architectural evaluation and is organized around three main axes, which correspond to the articles developed throughout the research.

The ISAA (Iterative Software Architecture Assessment) framework is presented in detail, describing its phases, functions, artifacts, and analysis criteria. The framework is validated by experienced software industry professionals, who provided valuable insights into its applicability, clarity, and efficiency. The results indicate that ISAA promotes collaboration among stakeholders, improves decision traceability, and enhances the quality of architectural documentation, consolidating its position as a relevant tool in both academic and industrial settings.

Beyond its practical contribution, ISAA seeks to fill a gap identified in the literature, marked by the scarcity of studies and concrete implementations of the ISO/IEC/IEEE 42020:2019 standard, whose adoption remains limited in the current Software Engineering landscape (MARTIN, 2018; COSTA; SOARES, 2023b; MELO; SOARES, 2025).

Despite the recognized relevance of both ATAM and ISO/IEC/IEEE 42020:2019, practitioners still face a recurring challenge: choosing, tailoring, and operationalizing architectural assessment in a way that is both systematic and compatible with contemporary development constraints (e.g., iterative delivery, continuous evolution, and heterogeneous stakeholder participation). In particular, while ATAM offers a well-established assessment approach focused on quality-attribute tradeoffs, the ISO/IEC/IEEE 42020:2019 standard provides a broader organizational and lifecycle perspective, but its assessment guidance is less frequently translated into concrete, repeatable assessment routines in practice (KAZMAN et al., 2000; MARTIN, 2018; COSTA; SOARES, 2023b; MELO; SOARES, 2025). This dissertation addresses this gap by (i) structuring a comparative perspective between these approaches in Chapter 9 and (ii) proposing an assessment framework designed to be iterative, actionable, and traceable in real-world settings in Chapter 9.

Accordingly, the main objective of this dissertation is to advance the state of practice and research in Software Architecture assessment by connecting method-level guidance (ATAM) with standard-level guidance (ISO/IEC/IEEE 42020:2019), and by consolidating this connection into an iterative framework (ISAA) that supports systematic assessment, stakeholder communication, and architectural decision traceability throughout the software lifecycle (KAZMAN et al., 2000; MARTIN, 2018; COSTA; SOARES, 2023b; MELO; SOARES, 2025).

From this objective, the dissertation delivers three primary contributions. First, it provides a structured comparison between ATAM and the assessment guidance established in Chapter 9 of ISO/IEC/IEEE 42020:2019, highlighting complementarities, overlaps, and key differences that influence applicability and expected outcomes (KAZMAN et al., 2000; MELO; SOARES, 2025). Second, it proposes the ISAA (Iterative Software Architecture Assessment) framework, detailing its phases, activities, artifacts, and evaluation criteria to enable repeatable assessments aligned with organizational goals and quality requirements (MELO; SOARES, 2025). Third, it reports empirical evidence from the validation of ISAA with experienced industry professionals, synthesizing feedback about clarity, feasibility, and expected benefits such as collaboration support and improved architectural documentation and traceability (KAUR; KHURANA; MANISHA, 2021; SILVA et al., 2023).

In addition, the dissertation is structured around three main axes that reflect the articles developed throughout the research, ensuring a progressive buildup from comparative foundations to an actionable framework and its empirical assessment.

To guide the reader and make explicit what follows, this dissertation is organized as follows. Chapter 2 introduces the theoretical foundations, covering key concepts of Software Architecture and assessment, as well as the role of architects and recurring challenges in architectural practice. Chapter 3 details the processes established by ISO/IEC/IEEE 42020:2019, with emphasis on the standard's lifecycle and governance-oriented perspective. Chapter 4 presents a systematic process to compare ATAM with the assessment guidance defined in Chapter 9 of ISO/IEC/IEEE

42020:2019, reporting similarities, differences, and practical implications. Chapter 5 introduces the ISAA framework, describing its iterative structure, activities, and artifacts, and reports its validation with industry professionals through interviews and synthesized results. Chapter 6 discusses future trends that impact Software Architecture and its assessment (including emerging technological and organizational drivers). Finally, Chapter 7 summarizes the findings, revisits contributions, and outlines limitations and future work.

2

Theoretical Reference

Drawing upon the literature review presented in (ABRAÃO; INSFRAN, 2017; BANIJAMALI et al., 2019a; AHMADI et al., 2019; SILVA et al., 2023), the analyzed studies provided insights into approaches for comparing ATAM with the guidelines established in Chapter 9 of the ISO/IEC/IEEE 42020:2019 standard. The synthesis of these insights enabled the development of a process and a comparative framework for evaluating criteria related to the construction of Software Architectures from their initial design phases. Such classification supports the prediction of the sustainability of Software Architectures designed using these evaluation methods (KOZIOLEK, 2011).

2.1 Software Architecture

At its core, Software Architecture represents the fundamental organization of a software system, encompassing its structural elements, their relationships, and the principles that guide its design and evolution (ISO/IEC/IEEE, 2011). Software Architecture can be seen as the blueprint of a software, similar to the architectural design of a building, defining how components interact, how responsibilities are distributed, and how the software achieves its functional and non-functional goals. Formally, Shaw and Garlan define Software Architecture as “the set of structures necessary for reasoning about the system, comprising the software elements, their externally visible properties, and the relationships between them.” (MARY; DAVID, 1996; BASS; CLEMENTS; KAZMAN, 2021).

More than a structural description, Software Architecture serves as a strategic instrument for managing complexity and ensuring alignment of technical decisions with business objectives. A shared understanding among stakeholders including developers, architects, project managers, and customers facilitates clarity on how the system will evolve and how quality attributes such as performance, security, scalability, and maintainability will be achieved (BASS; CLEMENTS;

KAZMAN, 2021; ISO/IEC/IEEE, 2019). Bass et al. emphasize that Software Architecture constitutes the first tangible artifact enabling early assessment of a capacity to meet its requirements. Such alignment fundamentally guides technical implementation and acts as a foundation for organizational agility, operational efficiency, and long-term value creation. By bridging technical and business perspectives, Software Architecture empowers enterprises to adapt to changing market demands while sustaining competitive advantage in dynamic environments.

The role of Software Architecture extends throughout the entire software life cycle. During the requirements engineering phase, architectural drivers, such as quality attributes, constraints, and business objectives, are identified. The architecture guides design and implementation, providing a framework for decomposition into modules, communication mechanisms, and technology choices. During testing and deployment, the architecture influences testing strategies and system integration, while during maintenance and evolution, it serves as a reference model for managing change and technical debt (MARTIN, 2017; ISO/IEC/IEEE, 2019). Modern approaches, such as continuous architecture and architecture as code, emphasize the need to evolve the architecture iteratively, along with agile and DevOps practices (ERDER; PUREUR; WOODS, 2021; KOTHAPALLI et al., 2024; HUMBLE; FARLEY, 2010).

The responsibilities for defining and maintaining architecture are typically assigned to software architects or technical leads, but the process is inherently collaborative. Successful architectural decisions emerge from negotiation among technical and non-technical stakeholders, balancing tradeoffs between cost, performance, time-to-market, and long-term sustainability (RICHARDS; FORD, 2020; ISO/IEC/IEEE, 2019). Without this architectural governance, software projects risk loss of coherence, increased coupling, poor scalability, and unsustainable maintenance costs.

Distinguishing Software Architecture from Software Design is necessary to understand the responsibilities of each. Architecture focuses on high-level structural and strategic decisions, such as division of responsibilities, communication between components, and the technology stack, while Design deals with more refined aspects, such as algorithms, data structures, and implementation patterns. As Ralph Johnson notes, architectural decisions are those that are difficult to change later, making them essential to a software longevity and adaptability (RICHARDS, 2015; BASS; CLEMENTS; KAZMAN, 2021).

In summary, Software Architecture is not a static artifact but a living process connecting engineering and strategy. Visibility is provided, uncertainty is reduced, and continuous alignment between technological evolution and business value is supported. Recent studies highlight the essential role of treating architecture as an evolving asset rather than a one-time design to achieve agility, resilience, and sustainability in modern software systems (VERDECCHIA et al., 2021; KOTHAPALLI et al., 2024).

2.1.1 Historical Evolution of Software Architecture

Concern about software architecture did not arise suddenly. Since the late 1960s, work by renowned researchers on program structuring and modular decomposition introduced concepts that are now associated with Software Architecture. Dijkstra's work on structured programming and the THE multiprogramming system highlighted the importance of controlling complexity through clear separation of responsibilities and well-defined layers, anticipating architectural reasoning about systems as a whole (GARLAN, 1995).

In parallel, Parnas' article on criteria for decomposing systems into modules formalized ideas such as information hiding and interface stability. Although these authors did not yet use the term Software Architecture in the sense currently adopted, their work already treated the overall structure of the software and the organization of its modules as first-class design decisions (PARNAS, 1977; GARLAN, 1995).

During the 1970s and 1980s, the ideas of researchers like Parnas and Dijkstra evolved into operating systems, real-time and embedded systems, and large information systems. Layered structures, "pipe and filter" type organizations, and other recurring forms of composition began to be recognized, and the first notations of architectural description were proposed. This period consolidated the view that Software Architecture encompasses not only modularization but also interaction patterns, allocation of responsibilities to components, and support for quality attributes such as performance, reliability, and scalability. Upon this conceptual basis, subsequent architectural styles and paradigms were built (MARY; DAVID, 1996).

Software Architecture has evolved significantly over the decades, keeping pace with the increasing complexity of software and changing business demands. The following highlights the main milestones in this evolution:

1. **Traditional Monoliths (1990s)**: During the 1990s, software systems were predominantly developed as monolithic applications with large unified code bases in which user interface, business logic and data access were tightly integrated. The architectural focus centered on achieving logical separation of layers within a single process to improve maintainability and internal organization.

2. **Service-Oriented Architecture - SOA (2000s)**: With the rise of the Internet, software needed to communicate and reuse functionality. SOA emerged to break the monolith into larger, independent, and loosely coupled services that communicated using standardized protocols (such as SOAP or XML). The focus was on reuse and enterprise integration (ERL, 2005).

3. **Microservices and Cloud (2010s onward)**: Driven by cloud computing, the microservices architecture has become dominant. By fragmenting software into even smaller services, each running in its own process, often managing its own database, and communicating via

APIs, this evolution enables massive scalability and independent deployment, known as CI/CD (HUMBLE; FARLEY, 2010; NEWMAN, 2015; BALALAIE; HEYDARNOORI; JAMSHIDI, 2016).

2.1.2 The Role of the Software Architect

The software architect is the professional responsible for defining, documenting, and developing the architecture of complex software. The Software Architect role goes beyond creating diagrams: it consists of a strategic technical decision-maker, ensuring that the software is robust, scalable, secure, and aligned with business objectives (BASS; CLEMENTS; KAZMAN, 2021; ISO/IEC/IEEE, 2019).

According to Bass et al. (BASS; CLEMENTS; KAZMAN, 2021), the software architect must be able to transform functional and non-functional requirements into a framework of components and connections that support the software throughout its lifecycle.

Key responsibilities include:

- **Make critical architectural decisions**: Choose technologies, architectural styles, and standards appropriate to the context and requirements (BASS; CLEMENTS; KAZMAN, 2021; ISO/IEC/IEEE, 2019).
- **Ensure adherence to non-functional requirements**: The architectural design must address key quality attributes, including performance, scalability, security, availability, maintainability, and reliability (BASS; CLEMENTS; KAZMAN, 2021).
- **Communicate and document the Software Architecture:** Use diagrams and lightweight documentation mechanisms (e.g., ADRs - Architectural Decision Records) to keep architectural decisions explicit, traceable, and aligned across teams (BASS; CLEMENTS; KAZMAN, 2021; AHMETI B., 2024).
- **Assess technical risks and propose mitigations:** Anticipate bottlenecks, potential failures, dependencies, and scalability challenges, and propose architectural mitigations early in the lifecycle (BASS; CLEMENTS; KAZMAN, 2021; ISO/IEC/IEEE, 2019).
- **Mentor and align technical teams:** Act as a technical leader, guiding developers, reviewing implementations, and promoting good engineering practices (RICHARDS; FORD, 2020).
- **Continuously adapt the Software Architecture:** Architecture is not an immutable artifact; the architect must be able to adapt decisions based on changing context, requirements, or constraints (ISO/IEC/IEEE, 2019).

2.1.3 Software Architecture Challenges

Software Architecture seeks to solve the problem of creating an organized, efficient, and sustainable framework for the development of complex software. Aiming to define the structure, behavior and organization of software components to ensure that they meet functional and non-functional requirements, such as performance, security, maintainability and scalability, Software Architecture establishes standards, principles, and guidelines that facilitate the construction, evolution, and maintenance of software, ensuring quality, reliability, and alignment with business objectives (ISO/IEC/IEEE, 2019; BASS; CLEMENTS; KAZMAN, 2021).

Among the recurring challenges addressed by Software Architecture are modularization to facilitate reuse and maintenance, performance assurance through resource management, security to protect against threats, scalability to handle growth and fluctuating demand, and reliability during continuous delivery and operation in production environments (BASS; CLEMENTS; KAZMAN, 2021; HUMBLE; FARLEY, 2010; BEYER et al., 2016). Architecture also supports communication between teams and clear documentation, which is essential for minimizing risks and enabling adaptations throughout the software lifecycle (ISO/IEC/IEEE, 2019).

Some important points about the challenges that Software Architecture aims to solve are shown in Table 1.

Table 1 – Challenges of Software Architecture (synthesized from (BASS; CLEMENTS; KAZMAN, 2021; ISO/IEC/IEEE, 2019; BEYER et al., 2016))

Challenge	**Description**
Scalability	Ensure that the software can handle an exponential increase in users and data without compromising performance.
Security	Protect the software against cyber threats. This requires integrating security policies and authentication/authorization mechanisms into each layer of the Software Architecture.
Maintainability and Evolution	Designing the software so that it can be easily modified, corrected, and adapted to new business requirements without requiring a complete rewrite.
Latency and Performance	Ensuring that critical operations are executed in a time acceptable to the end user, especially in distributed Software Architectures.
Distributed Complexity Management	In microservices, the challenge is managing communication between dozens or hundreds of independent services and ensuring that transactions involving multiple services are consistent.
Observability	The ability to understand the internal state of a software system by analyzing its external outputs, such as logs, metrics, and transaction traces.

2.2 ATAM

ATAM (Architecture Tradeoff Analysis Method) is a structured method designed specifically for analyzing and evaluating architectural tradeoffs. The central point of the ATAM approach is to identify risks and make informed decisions throughout the software development lifecycle (KAZMAN et al., 2000; MÜLLER, 2020; ISO/IEC/IEEE, 2015). The idea is to refine the multiple

attributes and architectural decisions according to the needs of the project, to prevent hasty observations from jeopardizing the development of the product.

ATAM has its roots in three distinct areas: the fundamental principles of Software Architecture, the communities dedicated to analyzing quality attributes, and its direct predecessor, SAAM (Software Architecture Analysis Method). Although it shares similarities with ATAM, SAAM's main objective is to determine which Software Architectures best meet quality requirements, taking into account the specific context of software development (KAZMAN et al., 1994; MÜLLER, 2020).

There are risk assessment models that focus on unitary requirements, for example, performance, availability, and modifiability, which can skew the software product since a study focused on a single requirement can cause other requirements to be left out or go unnoticed. Using ATAM, the analysis is based on multiple requirements, making it clear which areas have tradeoffs and which areas need attention for a better balance (KAZMAN et al., 2000; MÜLLER, 2020).

By applying ATAM, organizations can explore in depth how architectural decisions have an impact on achieving the quality attributes that are essential to the success of the final product. ATAM contains phases defined as preparation, evaluation and consolidation of results, promoting a detailed evaluation of Software Architectures at different stages of development, using techniques such as scenarios, allowing potential risks to be identified and mitigated before they become concrete problems (SALDANA et al., 2019).

ATAM consists of nine steps:

1. **Present the ATAM**: The person responsible for the evaluation describes the evaluation method to the assembled participants, defines their expectations in line with the project's objectives, and answers the participants' questions.

2. **Present business drivers**: A project spokesperson describes what business goals are motivating the development efforts and hence what will be the primary architectural drivers.

3. **Present Software Architecture**: The architect will describe the proposed Software Architecture through the level of detail collected during the previous two steps, focusing on how it addresses the business drivers.

4. **Identify architectural approaches**: Identification of the best architectural approaches by the architect and the team for the software to be developed.

5. **Generate a quality attribute utility tree**: The quality factors that comprise software "utility" (e.g., availability, security, modifiability, usability) are elicited, specified to the level of scenarios, annotated with stimuli and responses, and prioritized.

6. **Analyze architectural approaches**: Based on high-priority factors identified in Step 5, the architectural approaches that address those factors are elicited and analyzed. For this step, architectural risks, sensitivity points, and tradeoff points are identified.

7. **Brainstorm and prioritize scenarios**: A larger set of scenarios is elicited from the entire group of stakeholders. This set of scenarios is prioritized through a voting process involving the entire stakeholder group.

8. **Analyze architectural approaches**: This step reiterates the activities of Step 6 but uses the highly ranked scenarios from Step 7. Scenarios are considered test cases to confirm the analysis performed so far. This analysis may uncover additional architectural approaches, risks, sensitivity, and tradeoff points, which are then documented.

9. **Present results**: Based on the information collected in ATAM (which may include approaches, scenarios, attribute-specific questions, risks, the utility tree, sensitivity points, and tradeoffs), the ATAM team presents the findings to stakeholders.

2.3 ISO/IEC/IEEE 42020:2019

The ISO/IEC/IEEE 42020:2019 standard complements processes related to ISO/IEEE 15288:2015 in the field of Systems Engineering, offering a more comprehensive approach to Software Architecture practices. In addition, the ISO/IEC/IEEE 42020:2019 standard improves and complements the procedures established in ISO/IEC/IEEE 12207:2017 and ISO/IEC 15704:2019, effectively integrating them (ISO/IEC/IEEE, 2015; ISO/IEC/IEEE, 2019).

The ISO/IEC/IEEE 42020:2019 standard is intrinsically linked to ISO/IEC/IEEE 42010:2011 and ISO/IEC/IEEE 42030:2019, which are, respectively, standards for Architecture Description and Architecture Evaluation (MARTIN, 2018; COSTA; SOARES, 2023b). Related patterns contribute to a consistent understanding and practice in the field of Software Architecture.

2.3.1 ISO/IEC/IEEE 42020:2019 Processes

ISO/IEC/IEEE 42020:2019 is articulated through six main processes: Governance, Management, Conceptualization, Evaluation, Elaboration, and Enablement. Each of these processes is designed to strengthen Software Architecture practices and promote effective implementation. Together, they form a robust framework that not only improves architectural practices but also ensures successful project development by providing a solid foundation for governance, strategy, and efficient execution.

Chapter 9 of ISO/IEC/IEEE 42020:2019 provides general guidelines for managing Software Architecture assessments, without delving into a specific methodology. Establishing

Table 2 – Architecture Process ISO/IEC/IEEE 42020:2019

Process	Description
Governance	Provides direction and guidance on the Software Architectures developed by the company and are usually decisions that come from the business level.
Management	Responsible for implementing the directives coming from governance is a process that requires continuous communication with governance, transmitting the defined strategies and progress towards achieving the objectives.
Conceptualization	Determines the appropriate solutions that must meet the needs arising from governance.
Evaluation	Determines the extent to which one or more of the Software Architectures developed meet the needs of the stakeholders, guaranteeing linear development in line with the defined objectives.
Elaboration	Responsible for documenting the Software Architecture(s) in a sufficiently complete manner for its intended use.
Enablement	Provides the necessary facilitators to carry out activities efficiently and intelligently.

principles and processes that can be applied in different organizational contexts and software projects. Rather than prescribing a step-by-step approach, ISO/IEC/IEEE 42020:2019 emphasizes the importance of adapting to specific project needs, collaborating with stakeholders, and adequately documenting assessment results.

Table 1 shows the functions of each process.

2.3.2 Detailing chapter 9 of the ISO/IEC/IEEE 42020:2019 standard

This section addresses each of the six main processes of ISO/IEC/IEEE 42020:2019 (Governance, Management, Conceptualization, Evaluation, Development, and Enablement) in detail individually.

For each process, the following aspects are addressed:

- **Purpose and Core Responsibility**: The fundamental objective of the process in the context of Software Architecture.
- **Key Activities**: The specific actions and tasks that the process encompasses (as established in Chapters 6 to 11 of the standard).
- **Essential Contributions**: How the process positively impacts software architecture practices and project success.

This breakdown aims to provide an in-depth understanding of how each element of the framework contributes to strengthening software architecture practices within the organization,

ensuring that the defined guidelines are executed efficiently and in line with strategic objectives.

2.3.3 Chapter 9 - ISO/IEC/IEEE 42020:2019 - Evaluation

Chapter 9 of ISO/IEC/IEEE 42020:2019, titled “Architecture Assessment Process”, provides general guidelines for managing Software Architecture assessments, without delving into a specific methodology. Establishing principles and processes that can be applied to different organizational contexts and software projects.

To validate the completeness of the topics covered in the Software Architecture evaluation process, six questions must be answered cohesively:

9.1 Is the Software Architecture suitable for the intended uses and operational situations?

9.2 Is the Software Architecture flexible and extensible enough to meet future needs?

9.3 Is the quality of the Software Architecture acceptable?

9.4 Does the Software Architecture address the concerns of stakeholders?

9.5 Does the Software Architecture achieve its stated objectives?

9.6 Can the Software Architecture be implemented successfully?

Section 9.4.1: Focuses on defining the objectives of the Software Architecture assessment, as proposed by the stakeholders, and includes a full review if necessary. The use, development or adoption of specific techniques and tools can be beneficial for the evaluation of Software Architectures. Klinaku et. al., used modelling and simulations to evaluate alternative styles and configurations for cloud sizing policies (KLINAKU; HAKAMIAN; BECKER, 2021).

Section 9.4.2: Focuses on the evaluation, monitoring and control of the activities carried out during the Software Architecture assessment. Phase is important for verifying that the definitions and objectives defined by Software Architecture governance and management are being met, including tracing the results back to the source material and implementing corrections to the Software Architecture when necessary.

Section 9.4.3: Establishes the evaluation objectives and criteria. The criteria include the “value assessment”, which involves conditions and tests to be met as determined by the stakeholders, and the “architectural assessment”, which analyses the concepts used in the design of the Software Architecture to check their correlation with the defined objectives.

Section 9.4.4: Determining the evaluation methods that integrate the aforementioned objectives and criteria. The selection or development of value and Software Architecture evaluation methods must be aligned with these objectives. Although the ideal is to carry out several evaluations of the selected methods based on the software to be developed, this process

can become costly and unfeasible (BASS; NORD, 2012). Therefore, it is necessary to periodically review the defined objectives and maintain continuous communication with stakeholders, as efficient communication is correlated with increased productivity and the success of activities (KALOGIANNIDIS, 2020).

Section 9.4.5: Focuses on establishing appropriate measurement techniques, methods and tools. Considering the variety of possible approaches, it is not always necessary to implement a multi-layered assessment or to use weights to illustrate the tradeoffs related to the assessment objectives. Key measures include using scales and weights only when necessary, identifying measures that correspond to the desired objectives, defining the relationship between metrics, objectives and evaluation criteria and, finally, estimating the degree of risk involved.

Section 9.4.6: Stands out for carrying out a review and gathering essential information to evaluate the software. During this phase, the architect, together with the development team, analyses in detail all the relevant aspects of the Software Architecture, the associated quality attributes, the main decisions made and the risks identified.

Section 9.4.7: Establishes which teams are responsible for the evaluation review and analyse the data collected to identify relevant patterns, trends and insights. During consolidation that detailed assessment reports are drawn up, documenting the results of the assessment, including strengths, weaknesses, risks identified and recommendations for improvement.

Section 9.4.8: One or more Software Architectures under evaluation are analysed to identify and characterise the points of tradeoffs associated with quality attributes, stakeholder concerns, costs and risks. The aim is to determine the extent to which an Software Architecture meets the established objectives, offering a detailed view of its strengths and weaknesses regarding requirements and expectations.

Section 9.4.9: Involves validating the conclusions and recommendations with experts and stakeholders. The aim is to confirm that the conclusions and recommendations obtained are consistent and relevant to the evaluation and evolution of other Software Architectures.

Section 9.4.10: Communicating results to stakeholders involved in the project. During this task, the main conclusions and recommendations that were identified during the evaluation of the Software Architecture are selected, generating a report that will be presented and distributed to the interested parties.

3

A Process to Compare ATAM and Chapter 9 of ISO/IEC/IEEE 42020:2019

Comparing ATAM to ISO/IEC/IEEE 42020:2019 requires a consistent set of criteria that allows for a fair evaluation of both approaches. To this end, the criteria were not defined arbitrarily, but rather derived from a literature review, which ensures methodological rigor and minimizes bias (MELO; SOARES, 2025). Literature review allowed to identify which aspects are considered relevant for evaluation of Software Architecture methods and standards, such as:

1. Coverage of architectural perspectives (functional, quality, organizational);
2. Emphasis on quality attributes (performance, security, maintainability, scalability);
3. Formalization and level of detail of the information provided;
4. Decision-making support (such as prioritization of scenarios or requirements);
5. Practical applicability (processes, tools, roles involved);

3.1 Comparative Tables

This section presents the results of an information collection process conducted to identify and organize key comparison points between the Architecture Tradeoff Analysis Method (ATAM) and the guidelines established in Chapter 9 of the ISO/IEC/IEEE 42020:2019 standard. The gathered data facilitated the identification of overlapping areas addressed by both approaches, as well as unique aspects specific to each, providing a structured basis for evaluating their coverage, depth, and applicability. By systematically classifying these points, a comprehensive view emerges that supports the assessment of how each method contributes to the construction, evaluation, and sustainability of Software Architectures.

Eliciting functional and non-functional requirements is the first process of Requirements Engineering. Through it, it is possible to understand and identify the needs of interested parties

Table 3 – Comparison of Requirements & Design between ATAM and ISO/IEC/IEEE 42020:2019

Criteria	ATAM	ISO 42020
Functional Requirements	✓	✓
Non-Functional Requirements	✓	✓
Technical Selection	✓	✓
Architecture Selection	✓	✓
Architecture Style	✓	✓
Design Patterns	✓	✓
Possible Scenarios	✓	✓
Scenario Details	✓	

(AKRAM; AHMAD; SADIQ, 2024). The software architect is responsible for determining the technologies that should be used to build the software, selecting Software Architectures appropriate to the context in which they are applied, and choosing architectural styles.

Architectural styles provide general patterns for organizing software components' and defining how they interact with each other. Choosing the right architectural style facilitates software maintenance, scalability, and adaptability, ensuring that it meets specified requirements efficiently and effectively (POWER; WIRFS-BROCK, 2019; MARZOONI; MOTAMENI; EBRAHIMNEJAD, 2021). Crucial points such as requirements elicitation, technology selection, choice of appropriate Software Architectures, and definition of architectural styles are addressed in both ATAM and ISO/IEC/IEEE 42020:2019, highlighting the importance of addressing these aspects to ensure the development of Software Architectures that are robust and effective.

Design patterns are reusable conceptual solutions to common software design problems throughout the development cycle, providing a construction pattern that developers can follow. The application of a design pattern varies depending on the problem to be solved, making some patterns more or less suitable for certain cases (ULUDAG; MATTHES, 2020; EIGLER; HUBER; HAGEL, 2023), a point addressed by both.

The discussion and detailing of scenarios are important points covered in different ways by ATAM and ISO/IEC/IEEE 42020:2019. Both approaches involve discussing possible scenarios with stakeholders, and refining the main ideas that should be incorporated into the project. However, the detailing of scenarios is an exclusive feature of ATAM, as it is a scenario-based methodology, while ISO/IEC/IEEE 42020:2019 addresses this issue in a more generic way.

Table 4 – Comparison of Development Tools between ATAM and ISO/IEC/IEEE 42020:2019

Criteria	ATAM	ISO 42020
CI/CD Tools		✓
Logging Tools		✓
Versioning Tools		✓
Dependency Manager		✓
IDE Tools		✓

The ATAM column is intentionally left empty. Proposed in the late 1990s, ATAM focuses on evaluating architectural decisions and trade-offs between quality attributes, having been

formulated before practices such as CI/CD, DevOps tools, and systematic log management became common. Therefore, the method does not explicitly prescribe or classify these development tools.

Adopting CI/CD (Continuous Integration and Continuous Delivery) practices plays a crucial role in new software development standards, allowing developers to reduce product release cycles and detect failures more quickly. However, despite its significant contributions, using CI/CD in practice can be laborious and generate new challenges for development teams (ZAMPETTI et al., 2023). Monitoring and management of logs is a process that is intrinsically linked to risk mitigation. Monitoring and recording software behavior are essential activities within a Software Architecture, as they facilitate error detection, performance analysis, and maintenance (CâNDIDO; ANICHE; DEURSEN, 2021). The selection of development tools, to be used in conjunction with the supporting technologies defined for the architectural design, is a concern explicitly addressed by the ISO/IEC/IEEE 42020:2019 standard.

Table 5 – Comparison of Infrastructure & Platform between ATAM and ISO/IEC/IEEE 42020:2019

Criteria	**ATAM**	**ISO 42020**
Infra. Management	✓	✓
Platform Services	✓	✓

Two topics addressed by both ATAM and ISO/IEC/IEEE 42020:2019 relate to infrastructure and platform choices fundamental aspects of software development that directly influence efficiency, scalability, and maintainability. Appropriate infrastructure selection facilitates implementation of robust architectural patterns and ensures adaptability of software to technological changes and evolving business needs. Decisions regarding infrastructure impact not only immediate software performance but also its long-term evolution and capacity for integration with other technological solutions.

Mapping security strategies is fundamental to the success and reliability of software. Security processes, based on CIA principles (Confidentiality, Integrity, Availability), need to be discussed from the initial stages of Software Architecture design (ZAROUR; ALENEZI; ALSARAYRAH, 2020). Software scalability is an irrefutable and necessary feature of Software Architectures. As the number of users of a piece of software grows, it becomes more likely that the software will be overloaded.

If the hardware used in conjunction with the software developed does not support this exponential growth, then the software could fail, revealing flaws in the design of the Software Architecture. Therefore, software scalability must be considered from the outset of its design to avoid future problems, maintenance costs, and even the loss of customers due to not being able to serve them (AMORIM; ALMEIDA; MCGREGOR, 2014).

Recovery and fault tolerance strategies, addressed by both ATAM and ISO/IEC/IEEE 42020:2019, represent essential concerns in modern software development models. High-

Table 6 – Comparison of Strategic Issues between ATAM and ISO/IEC/IEEE 42020:2019

Criteria	**ATAM**	**ISO 42020**
Flexibility	✓	✓
Stakeholder Involvement	✓	✓
Continuous Evaluation	✓	✓
Architecture Alternatives	✓	✓
Recovery & Fault Tolerance	✓	✓
Maintenance & Support		✓
Monitoring		✓
Alignment Meetings	✓	✓
Testing Strategies	✓	✓
Communication	✓	✓
Type	Specific method	General guidelines
Evaluation Scope	Quality attributes	Lifecycle management
Quality Attributes	Tree of QA	Architecture documentation
Number of Tasks	9	10
Preliminary Architecture Sketch	✓	
General Structure	Prepare, evaluate & consolidate	Documentation & collaboration
Risk Focus	Priority	No specific focus

performance systems are designed to minimize the occurrence of errors, given the intense competition within the software industry. Even small failures can result in significant financial and reputational losses (PARCHMAN et al., 2016; TADI, 2022). Maintenance and support strategies are fundamental in the development of a Software Architecture, as they guarantee the longevity, efficiency, and adaptability of systems. Maintenance allows software to evolve as user needs and market conditions change, fixing bugs, improving performance, and implementing new features. Support, in turn, ensures that users receive the help they need to solve problems and use the software effectively. Together, these strategies aim to achieve customer satisfaction, reduce costs in the long term and prevent failures that could compromise software execution (HINRICHS; PRIFTI, 2022).

Testing strategies are steps in software development, by enabling exhaustive testing to identify bugs, flaws, and incorrectly implemented processes. Tests ensure software quality and reliability, increasing end-user satisfaction, and reducing costs associated with post-deployment fixes (COSTA; TEIXEIRA, 2018; DUARTE et al., 2024). Effective communication and coordination strategies are necessary in software development, especially in distributed or multidisciplinary teams.

Clear and consistent communication facilitates the exchange of information, rapid problem resolution, and informed decision making, ensuring that all team members are aligned with project objectives and deadlines. Project management tools, regular meetings and detailed documentation are essential for coordinating activities, distributing responsibilities and monitoring progress. Good cooperation also helps to minimize rework and conflicts, improving efficiency and productivity (KALOGIANNIDIS, 2020; MULLER et al., 2019). Both are concerns explained by ATAM and ISO/IEC/IEEE 42020:2019.

3.2 Similarities between ATAM and ISO/IEC/IEEE 42020:2019

By correlating ATAM and ISO/IEC/IEEE 42020:2019, several similarities can be highlighted. The first is the flexibility in the order of the stages. Although numbered and sequential, both approaches allow the steps to be carried out as the Software Architecture team and stakeholders decide to proceed. Constant communication between stakeholders and business drivers is essential for both approaches. From this communication and initial planning, the most important functional requirements, technical, economic, management constraints, key stakeholders, and business objectives are defined.

Stakeholder involvement is a feature present from the outset in both ATAM and the ISO/IEC/IEEE 42020:2019 guidelines. Stakeholders must maintain continuous communication and be involved during refinements throughout the development of the project, ensuring that their objectives are aligned and can be achieved. Both approaches emphasise the importance of continually evaluating the Software Architecture to ensure that it meets the quality requirements and project objectives.

Both ATAM and ISO/IEC/IEEE 42020:2019 emphasise the importance of proper documentation that contains all the objectives, requirements and possible scenarios, allowing stakeholders to have a complete understanding of the project.

ATAM and the ISO/IEC/IEEE 42020:2019 guidelines identify possible architectural approaches to be followed. In ATAM, the identification is performed in an initial phase, followed by the generation of the quality attribute tree and a thorough evaluation to correlate which Software Architectures meet the most requirements with the best quality, according to prioritisation. ISO/IEC/IEEE 42020:2019 guidelines establish techniques, methods, and tools to assist in the investigation of architectural alternatives, followed by an analysis of the architectural properties that best meet the requirements, highlighting the favourite choices.

Finally, both approaches highlight the importance of adapting to the project, with alignment meetings from the beginning of the product's conception to ensure that the main objectives are accepted by the majority of stakeholders.

3.3 Differences between ATAM and ISO/IEC/IEEE 42020:2019

ATAM is a specific method for analysing and evaluating architectural tradeoffs, focused on identifying risks, sensitivity points, and making informed decisions during the software development process. In contrast, Chapter 9 of ISO/IEC/ IEEE 42020:2019 provides general guidelines for managing Software Architecture assessments, without delving into a specific methodology.

The ATAM evaluation scope focuses on evaluating a software Software Architecture in terms of quality attributes and identifying possible points of sensitivity between these attributes.

ISO/IEC/IEEE 42020:2019 addresses the lifecycle management of Software Architecture evaluations, including the definition of evaluation criteria, the selection of appropriate evaluation methods, and the integration of evaluation results into the development process.

Although ATAM and ISO/IEC/IEEE 42020:2019 address quality attributes as an essential pillar, they do so in different ways. ATAM is highly effective in creating a comprehensive tree of quality attributes by systematically evaluating architectural decisions against multiple attributes such as performance, security, modifiability, and usability. This helps stakeholders understand the tradeoffs involved in different architectural choices and their impact on software qualities. For example, ATAM can be used to determine the balance between performance and modifiability, highlighting architectural risks and their priorities. In contrast, ISO/IEC/IEEE 42020:2019 provides general guidelines for defining and managing the Software Architecture of software-intensive systems, including guidance on architectural documentation, architectural description management, and business-oriented Software Architecture planning.

When comparing the tasks of each method, there are some significant differences. ATAM consists of 9 tasks, while ISO/IEC/IEEE 42020:2019 defines 10 tasks. One notable difference is that ATAM's first task involves presenting the method, setting expectations, and clarifying any doubts the meeting may have. In contrast, ISO/IEC/IEEE 42020:2019 begins by noting the level of effort required for the Software Architecture assessment, without involving a detailed initial discussion with stakeholders.

The approach to the existing Software Architecture is another point of difference. In ATAM, after defining the business drivers, the responsible team drafts the Software Architecture to be followed, which will be discussed and adjusted as necessary. ISO/IEC/IEEE 42020:2019, on the other hand, works directly with the identification of possible Software Architectures that can meet the demand, without the need to write a preliminary architectural sketch.

Another distinguishing factor is brainstorming and prioritisation of scenarios, which in ATAM involves bringing together all stakeholders to discuss possible scenarios that could occur. Scenarios are classified as use case scenarios and change scenarios, which can be subdivided into growth and exploratory scenarios. Growth scenarios help to identify strengths and weaknesses, while exploratory scenarios identify points of sensitivity in the software. ISO/IEC/IEEE 42020:2019 does not specify this level of detail for prioritisation of scenarios.

ATAM structure is divided into three phases: preparation, evaluation, and consolidation, using techniques focused on quality attributes, scenarios and sensitivity points to conduct the evaluation. In contrast, Chapter 9 of ISO/IEC/IEEE 42020:2019 takes a more generic approach, focusing on documentation of the assessment and stakeholder collaboration throughout development.

Regarding the focus on risks, ATAM highlights risk mapping as a priority, helping the team to identify and mitigate these risks from the earliest stages of the project. ISO/IEC/IEEE

42020:2019, even though it recognises the importance of risks, does not specifically focus on them during the Software Architecture assessment.

4

ISAA Framework

The ISAA (Iterative Software Architecture Assessment) framework is a comprehensive approach to Software Architecture assessment based on the ISO/IEC/IEEE 42020:2019 standard and enriched with elements from other relevant frameworks, such as MS-QuAAF, SAF Toolkit, and SQME (SEDAGHATBAF; AZGOMI, 2019; KADRI; AOUAG; HEDJAZI, 2021; LAGO et al., 2024). ISAA is composed of 5 activities that contain subactivities:

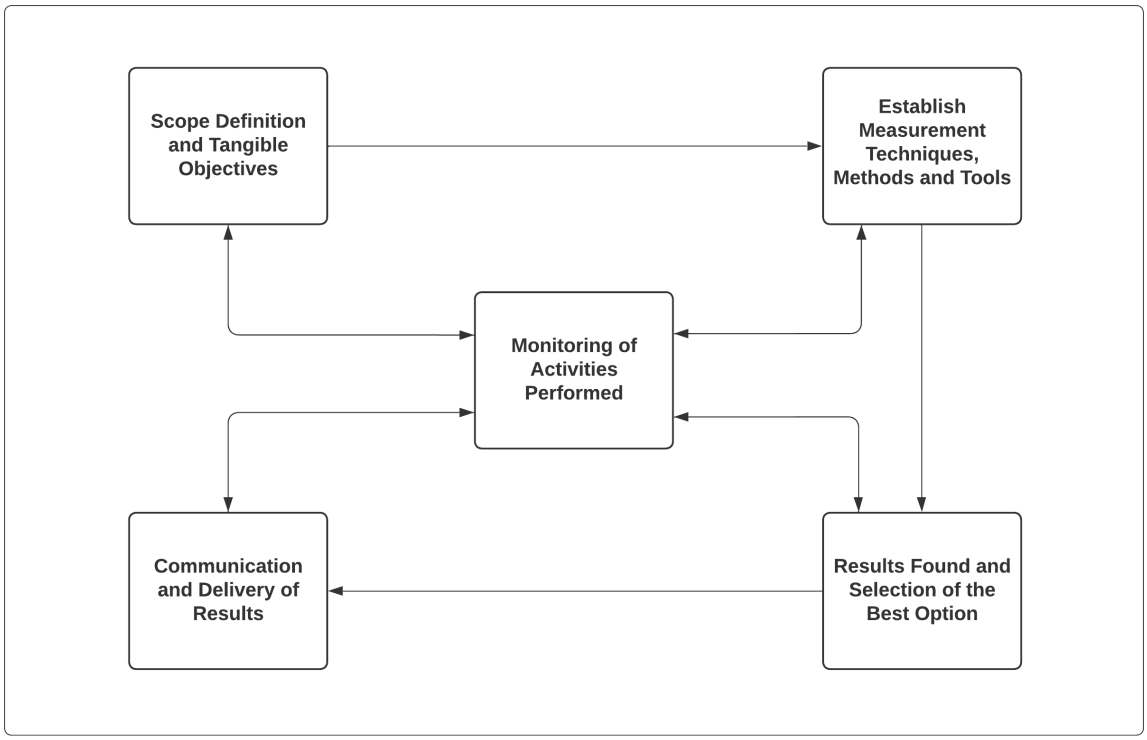

Figure 1 – Illustration from ISAA - Iterative Software Architecture Assessment

Figure 1 presents the visual construction of the framework, illustrating the activities in

the architectural assessment process and the defined sequence from activity 1 to activity 5.

Activity 2 stands out as responsible for the continuous monitoring of the activities performed. The activity is iterative and can be reused whenever there are changes in the evaluation process. Furthermore, all activities are connected to the central documentation, emphasizing that any action taken during the architectural evaluation must be recorded and continuously updated.

1. **Scope Definition and Tangible Objectives**: A fundamental activity in the architectural process involves collaboration with stakeholders to establish clear expectations for the proposed Software Architecture. The activity encompasses defining the core scope, identifying concrete objectives, selecting suitable approaches, and performing preliminary tests to evaluate the architecture's capacity for evolution.

    a) Identification and engagement of all relevant stakeholders
    b) Conducting structured discussions to gather expectations and requirements
    c) Clear definition of the main scope of the architectural project
    d) Establishment of tangible and measurable objectives
    e) Research and selection of architectural approaches aligned with the scope and objectives
    f) Prioritization and selection of up to 3 architectural approaches for evaluation
    g) Development of evaluation criteria for the selected approaches
    h) Conducting preliminary evaluation tests for each approach
    i) Analysis of the potential for evolution of each architectural approach
    j) Documentation of the results and insights obtained during the process

2. **Monitoring of Activities Performed**: The continuous monitoring activity of the ISAA framework involves the supervision and validation of all stages of the architectural assessment process. The goal is to ensure that each activity performed is aligned with the scope and objectives initially defined, ensuring the coherence and effectiveness of the process as a whole.

    a) Performing regular checks of alignment with scope and objectives
    b) Conducting review meetings after completion of each major activity
    c) Documenting deviations or adjustments needed during the process
    d) Updating the project plan based on monitoring results
    e) Proactively identifying potential risks or issues
    f) Implementing corrective actions when necessary

3. **Establish Measurement Techniques, Methods and Tools**: Focuses on defining and implementing a comprehensive set of techniques, methods, and tools to measure and evaluate multiple aspects of Software Architecture.

    a) Identifying the quality attributes relevant to the project

    b) Selecting appropriate metrics for each quality attribute

    c) Defining data collection techniques for each metric

    d) Researching and selecting appropriate measurement tools

    e) Training staff in the selected techniques and tools

    f) Conducting pilot tests to validate the techniques and tools

    g) Detailed documentation of the established measurement process

4. **Results Found and Selection of the Best Option**: Involves analysing the results obtained from the different evaluation methods and selecting the most suitable architectural option based on the overall findings.

    a) Compilation of all results of the evaluations performed

    b) Comparative analysis of the results between the different architectural options

    c) Identification of strengths and weaknesses of each option

    d) Assessment of the alignment of each option with the project objectives

    e) Consideration of factors such as cost, time and resources required

    f) Consultation with stakeholders to obtain feedback on the results

    g) Weighing the tradeoffs between the different options

    h) Selection of the best architectural option based on the overall analysis

    i) Detailed documentation of the decision and its justification

    j) Preparation of an implementation plan for the chosen Software Architecture

5. **Communication and Delivery of Results**: The final phase of the ISAA framework constitutes a comprehensive activity that involves documenting and effectively disseminating all relevant information related to the architectural project. The objective is to provide stakeholders with a complete overview of the process, including all stages, decisions, and outcomes obtained throughout the application of the framework.

    a) Compilation of all information and details of the architectural project

    b) Structured organization of the documentation, covering all stages of the framework

    c) Details of the scope and final objectives of the project

d) Description of the chosen Software Architecture and justification for the selection

e) Documentation of all activities carried out during the process

f) Detailed record of the time spent in each phase of the framework

g) Preparation of executive and technical reports

h) Review and validation of the documentation with the main stakeholders

i) Controlled distribution of the final documents to the relevant stakeholders

ISAA allows non-linearity in the process, offering flexibility to fit contemporary software projects. Stakeholders leading the architectural assessment may choose which activity to start from, provided that objectives and constraints remain aligned. Each subactivity represents an action required to complete an activity. ISAA also supports the evolution of the set of activities: new subactivities can be introduced and existing ones removed or refactored according to the project's needs, keeping the structure complete and adaptable.

4.1 Definition of ISAA Activities

Activities within the framework were derived from the ISO/IEC/IEEE 42020:2019 standard and further refined through an extensive literature review. Analysis of multiple studies and the recurring themes identified in them led to the definition of the five activities that constitute ISAA. The integrated approach enabled the development of a robust framework grounded in internationally recognized practices and enriched by insights from contemporary academic research.

4.1.1 Defining Scope and Tangible Objectives

Defining scope and tangible objectives is an aspect in sustainable software development and effective project management, as it establishes the foundations for the entire development process and determines the final success of the product. Authors of paper (LAGO, 2019) propose the use of "decision maps" as an innovative tool to frame architectural design concerns around four dimensions of sustainability: technical, economic, social and environmental. Maps offer a holistic and visual approach, allowing development teams to comprehensively explore the design space and characterise sustainability concerns before making specific decisions.

However, other work (VERDECCHIA et al., 2017) emphasises the importance of estimating the energy impact of software releases, highlighting how design decisions can significantly affect environmental sustainability. The proposed approach recognises that software energy consumption is a critical factor in the overall environmental footprint of software systems. The authors present methods to quantify and analyse energy consumption across different software releases, providing valuable insights into how to optimize design to improve energy

efficiency. Such an approach not only contributes to environmental sustainability but also leads to operational cost reductions and improvements in software performance.

In another paper (VERA; PEROVICH; OCHOA, 2021), the authors present an innovative tool named the "product scope canvas" to define product scope in the critical pre-sale phase. The canvas is designed to facilitate effective collaboration between customers and suppliers by creating a structured space to identify and prioritize the main functional and non-functional requirements of the product. By providing a visual and interactive representation of the product scope, the canvas allows all stakeholders to contribute their perspectives and reach a shared understanding of the project objectives. The tool aims to make the scope definition process more efficient and effective, reducing misunderstandings and rework in later phases of development.

4.1.2 Monitoring of Performed Activities

Activity monitoring is a priority in the framework, being responsible for supervising all other activities in development. Continuous monitoring validates and confirms the architectural health of the project, allowing early identification and resolution of potential problems (MIRAKHORLI; CLELAND-HUANG, 2016; WAN et al., 2023). Another study (WAN et al., 2023), involving interviews with 32 professionals responsible for implementing Software Architectures, revealed that software development without continuous monitoring often results in problems being detected only at the end of the development cycle. In such cases, the late discovery of architectural flaws makes it impossible to perform corrections due to the high cost in time and resources, frequently leading to legacy software that is difficult to maintain.

4.1.3 Establishing Measurement Techniques, Methods, and Tools

When establishing techniques, methods, and tools for measuring a Software Architecture, software developers aim to maintain technical quality and ensure that stakeholder objectives are being achieved. Such practice is essential for delivering reliable and sustainable software systems aligned with business expectations. Developing software is a highly complex activity, often accompanied by technical debt (TD) (CUNNINGHAM, 1992), a term introduced to describe suboptimal design or implementation solutions that provide short-term benefits but, if neglected, result in long-term costs.

Software commonly contains some degree of technical debt (LENARDUZZI et al., 2021), and it is the responsibility of developers and architects to prevent these suboptimal solutions from accumulating, as they can compromise software maintenance and evolution. Continuous evaluation of the Software Architecture, code refactoring, and artifact reviews therefore become crucial. The literature identifies four key strategies for prioritising technical debt:

1. **Internal Software Quality**: Focus on technical aspects of the code, such as readability, simplicity, and maintainability, allowing future changes to be less costly.

2. **Software Productivity**: Assessment of the impact of technical debt on developer efficiency, measuring whether technical debt is hindering the delivery of new features.

3. **Software Correction**: Prioritisation based on the resolution of critical defects that directly affect the functionality of the software, such as bugs and failures that compromise the user experience.

4. **Cost-Benefit Analysis**: Consideration of tradeoffs between refactoring costs and expected benefits, including financial and operational factors.

Strategies allow a systematic approach to mitigating technical debt, promoting sustainable software evolution and ensuring that the Software Architecture continues to meet technical and business objectives over time.

The use of specific tools for Software Architecture assessment and technical debt management has proven essential to ensure the sustainability and evolution of complex systems. Tools offer automated support to identify, analyze and measure different aspects related to software performance, maintenance and quality, in addition to assisting in prioritising corrective activities.

Among the widely used tools, the following stand out:

1. **SonarQube**: A popular open-source platform that performs static code analysis, identifying violations of quality standards, potential bugs and vulnerabilities. In addition, it provides metrics on the internal quality of the code and reports for continuous monitoring (HOYO-GABALDON et al., 2024).

2. **CAST Highlight**: A tool focused on software portfolio assessment that allows identifying and prioritising technical debt, quickly analyzing large volumes of code and suggesting corrective actions based on strategic criteria (DAS et al., 2022).

3. **Kiuwan**: A global platform for software quality analysis and management, supporting several languages. Offering functionalities for security analysis, technical debt measurement and detailed insights into code maintainability and efficiency (LÓPEZ et al., 2021).

Tools like these facilitate the implementation of strategies for mitigating technical debt and continuous Software Architecture evaluation, allowing stakeholders to make more informed and proactive decisions about software evolution.

4.1.4 Results Found and Selection of the Best Option

A series of tests (MAGABALEH et al., 2024) were conducted using different multicriteria decision making (MCDM) methods applied to Software Engineering processes and obtained

several interesting results. Tests involved the application of techniques such as AHP (Analytic Hierarchy Process), TOPSIS (Technique for Order of Preference by Similarity to Ideal Solution) and FAHP (Fuzzy Analytic Hierarchy Process) in different phases of the software development life cycle, from initial planning to maintenance. In addition, the authors also explored other MCDM methods such as DEMATEL (Decision Making Trial and Evaluation Laboratory), ANP (Analytic Network Process) and PROMETHEE (Preference Ranking Organization Method for Enrichment Evaluations).

The analysis of these results revealed that each MCDM method has its advantages and limitations when applied to specific Software Engineering problems. For example, AHP has proven particularly effective for prioritising software requirements, while TOPSIS has shown good results for selecting Software Architectures.

Carefully selecting the best outcome from the many options generated is crucial for several reasons:

1. **Project impact**: The decision made based on the best outcome will directly influence the success of the software project.
2. **Resource optimisation**: Choosing the most appropriate outcome allows for more efficient resource allocation.
3. **Software quality**: The correct selection can lead to a higher quality end product that is more aligned with user needs.
4. **Risk reduction**: Choosing the best outcome minimises the risks associated with suboptimal decisions in the development process.
5. **Continuous learning and improvement**: Analysing why a specific result is considered the best provides valuable insights for future projects.

4.1.5 Communication and Delivery of Results

One study (BOMSTRöM et al., 2023) emphasises the importance of visually representing information to catalyse discussions among stakeholders. The authors found that visualisation facilitates dialogue and collaboration between different stakeholder groups involved in software development. Such an approach is particularly relevant in the context of Software Architecture assessments, where communicating complex results clearly and accessibly to all parties is essential. The study also highlights the need to adapt the presentation of information to different stakeholder groups. According to (BOMSTRöM et al., 2023), two levels of information are essential: a high level, represented by a software product roadmap that provides an overview of the development direction, and a more detailed level that presents development milestones and the practical status of activities. The two-tier approach enables each stakeholder group to access

the information most relevant to its specific needs, enhancing the overall effectiveness of result communication.

Furthermore, it is essential to minimise the amount of information presented in day-to-day interactions with stakeholders. Although visual representation is a powerful tool for levelling knowledge among those involved, excess data can make visualisation complex and exhausting, making it difficult to fully understand the information presented and generating doubts both at the time and later (BOMSTRöM et al., 2023; ERAY; HAAS; RAYSIDE, 2021). Careful selection of shared information contributes to clear and objective communication, increasing the efficiency of discussions and decisions.

4.2 Survey Results and Participant Information

The survey was conducted over 24 consecutive days, contacting individuals with solid experience in Software Architecture. A total of 15 participants agreed to participate in the study, which was structured in two stages: the first involved reading the explanatory section about ISAA and the second consisted of completing a questionnaire.

From the 15 individuals who expressed willingness to participate, a total of 11 complete responses were obtained within the 24-day survey period. The collected feedback proved essential for refining the framework and provided valuable insights that underscored ISAA's potential as an effective tool for Software Architecture assessment.

After reading the ISAA documentation, participants were asked to provide demographic and academic information so that they could be classified according to their academic background, postgraduate studies, professional experience and other relevant data.

The majority of participants have degrees in Computer Science or Information Systems, which together represent eight participants in the sample. The remaining interviewees have degrees in Information Technology and one in Mechatronics Engineering.

Regarding postgraduate studies, the responses reveal a predominance of participants with a master's degree. Worth noting that some individuals have more than one course/certification, including several specialisations. Among these areas, areas such as Business Management, Competitive Intelligence, Systems Engineering, Project Management, Cybersecurity Management, COBIT, ITIL, Secure Development, Data Science and Big Data stand out, as well as MBAs in Cybersecurity and Robotics.

As illustrated in Figure 2, more than half of the 11 respondents reported having between 7 and 10 years of experience in software development. According to one study (PEITEK et al., 2022), developer proficiency is best evaluated through the ability to apply effective strategies, reduce cognitive load, and deliver correct outcomes in complex activities traits typically associated with senior-level professionals.

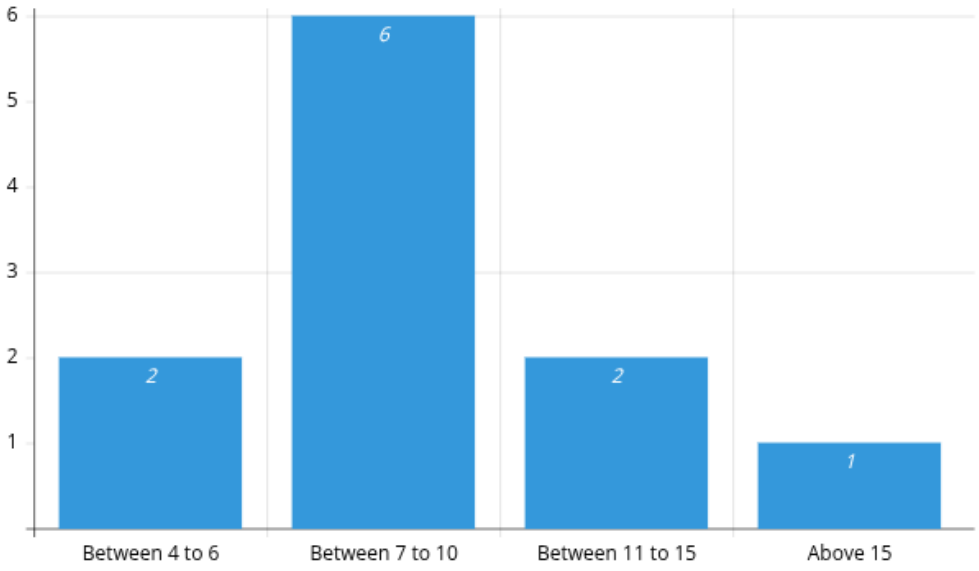

Figure 2 – Experience in Development

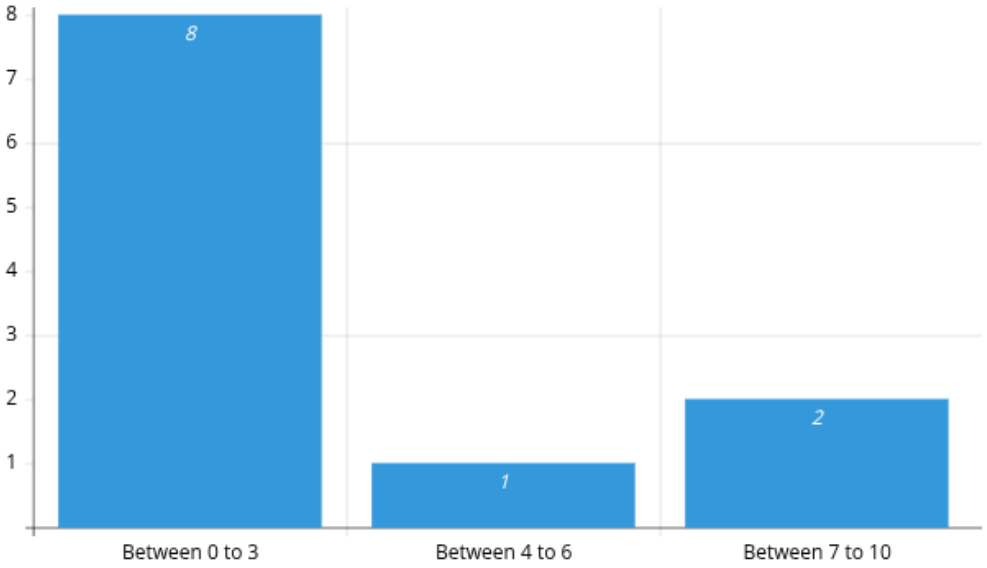

Figure 3 – Experience in Software Architecture

When asked about their experience specifically in the role of software architect, a decrease in the number of years is observed, as shown in Figure 3. The average experience, considering the upper bound of each classification, was approximately 4.5 years.

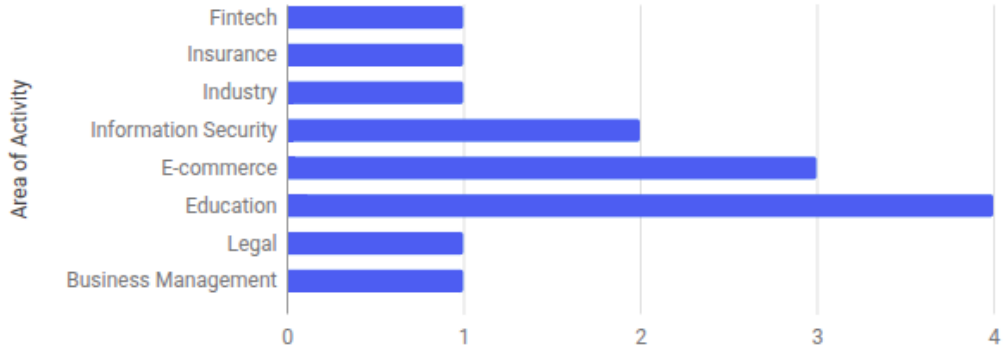

Figure 4 – Area of Activity

Figure 4 presents the distribution of the participants' professional domains. The Education sector stands out as the most represented, with 4 participants, followed by E-commerce with 3. Information Security also appears as a relevant domain, with 2 participants.

Other sectors, such as Fintech, Insurance, Industry, Legal, and Business Management, were each represented by one participant. Such diversity in professional backgrounds contributes to a broader range of perspectives and enriches the insights generated in this study.

4.3 Participant Responses

The ISAA questionnaire consists of 15 questions divided into three categories. The first two questions are intended to assess whether the participant is familiar with the ISO/IEC/IEEE 42020:2019 standard and whether they have ever used a Software Architecture assessment framework. Following that, there are nine Likert-scale questions designed to evaluate key aspects of ISAA. The questions include optional comment fields, triggered when participants assign a score of 3 or less, which corresponds to a "Neutral" or less favorable evaluation. In this case, participants are asked to justify their rating and describe why ISAA was not considered satisfactory in that aspect.

Lastly, the questionnaire includes four open-ended, subjective questions, aimed at eliciting insights based on the participants' professional experiences. Questions explore how ISAA compares to other frameworks, which aspects participants perceive as distinct, which areas require improvement, and what additional functionalities they would like to see incorporated into ISAA.

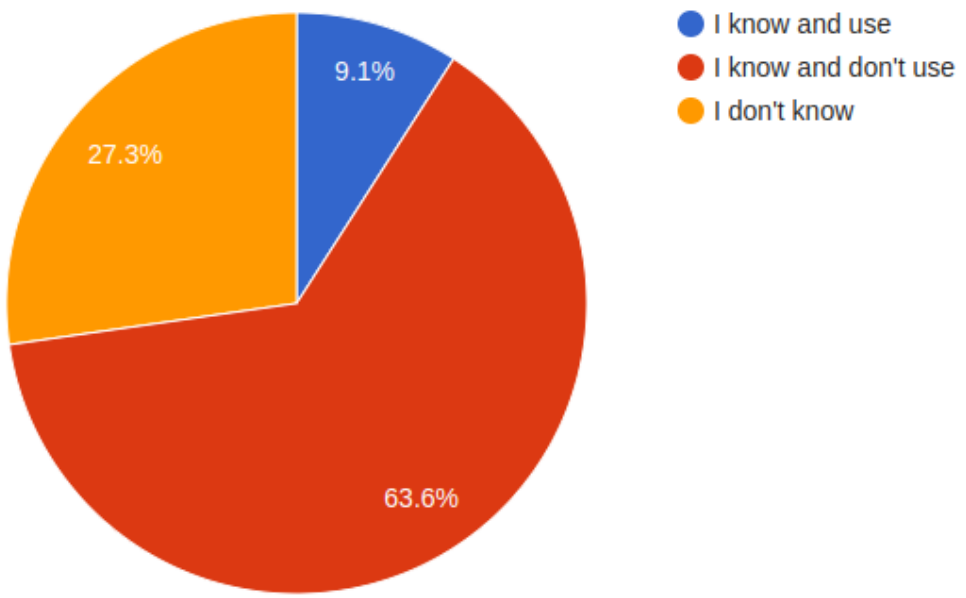

Figure 5 – Knowledge About ISO 42020 Standard

Among the 11 participants who completed the questionnaire, Question Q1 assessed their familiarity with the ISO/IEC/IEEE 42020:2019 standard. As shown in Figure 5, seven respondents indicated familiarity with the standard, but only one participant reported having applied it in practice. The result suggests that, while professionals demonstrate awareness of the standard, its practical adoption remains limited and its potential benefits appear to be underexplored.

Question Q2 investigated the use of Software Architecture assessment frameworks. All respondents stated that they do not employ any specific framework, with only one participant mentioning the use of general Software Architecture references provided by Len Bass (BASS; CLEMENTS; KAZMAN, 2003). The result, supported by insights from subsequent responses, indicates that incorporating structured frameworks into agile development processes remains challenging, particularly in environments where speed and adaptability are paramount. Furthermore, the adoption of such frameworks appears to be more feasible and beneficial in large-scale projects, where architectural rigour plays a more decisive role.

Question Q3, answered in Figure 6, investigates the extent to which ISAA's activities are appropriate for conducting Software Architecture assessments.The results indicate that ten participants agreed that ISAA includes relevant activities that support the identification of architectural risks and facilitate assessment activities. Only one participant selected the neutral option, reinforcing the general consensus about the framework's relevance in this area.

Question Q4, illustrated in Figure 7, examined whether ISAA effectively meets the needs of Software Architecture assessment. Responses were more diverse, with four respondents selecting the "Neutral" option. The result suggests that some respondents may require more

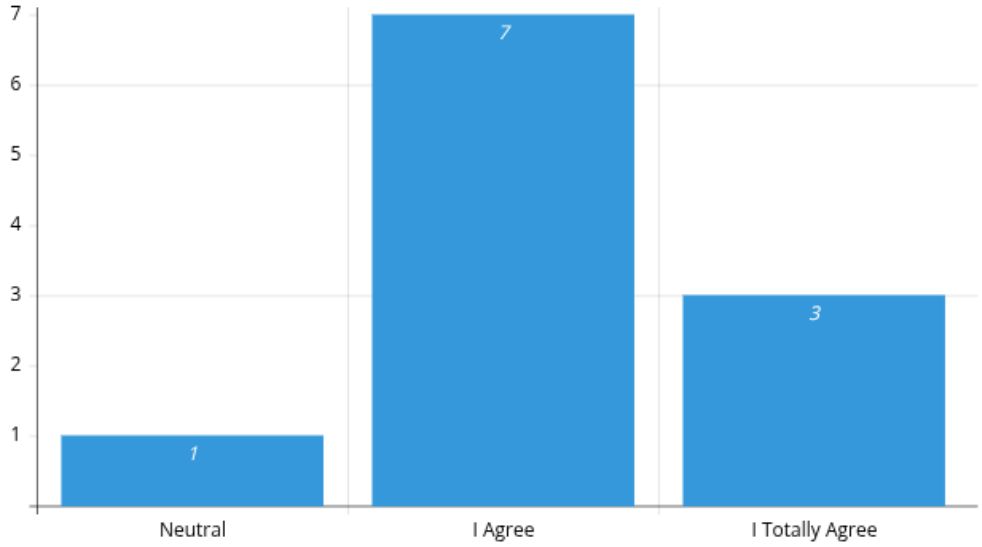

Figure 6 – Adaptation of Activities

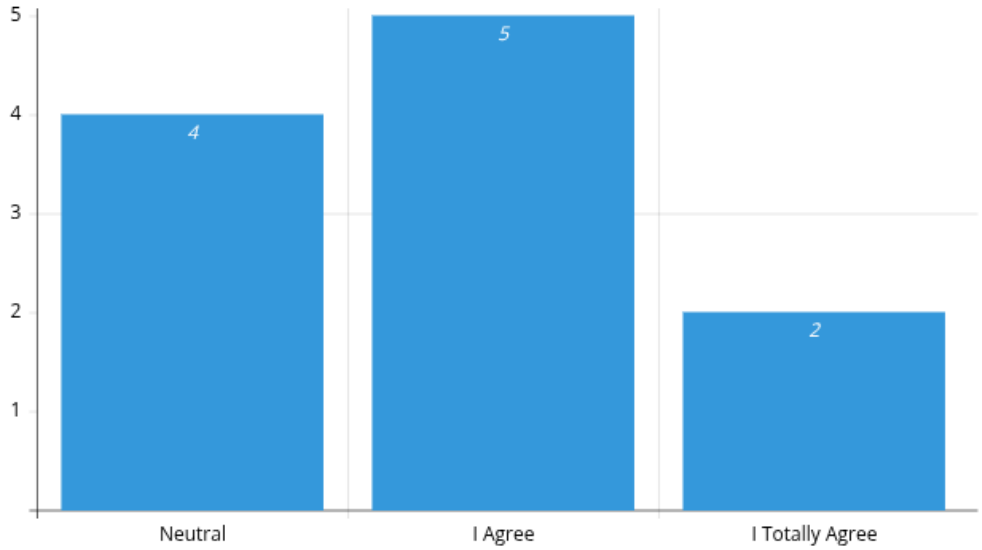

Figure 7 – Meeting the Needs of Architecture

comprehensive explanations, practical validation, or real-world application scenarios before forming a conclusive opinion about the framework's effectiveness.

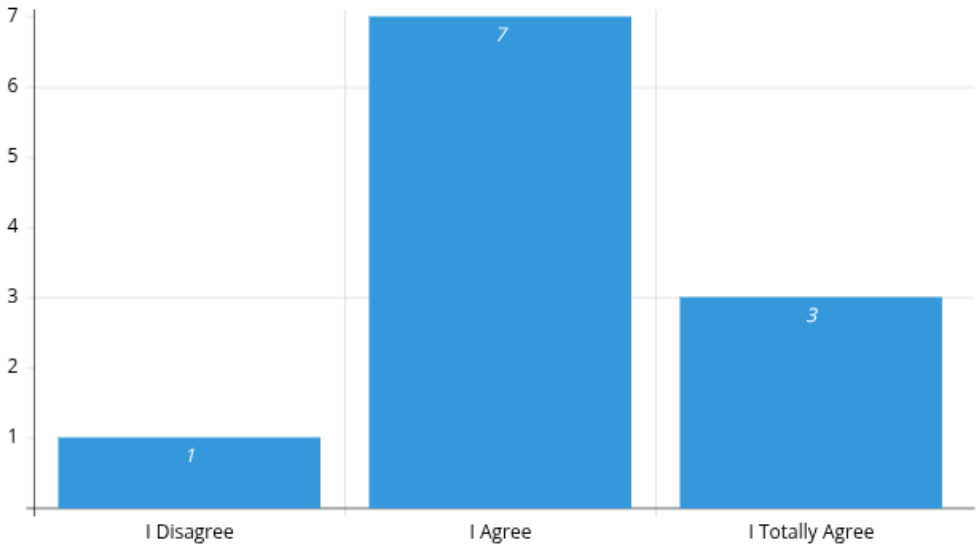

Figure 8 – Ease of Understanding ISAA

Question Q5, illustrated in Figure 8, evaluates the clarity and comprehensibility of the ISAA framework. Ten of the eleven respondents stated that they found the ISAA easy to understand. However, only one respondent indicated that the framework is still perceived as too generic. Participants recommended the inclusion of practical examples and a detailed step-by-step guide to improve accessibility and facilitate implementation.

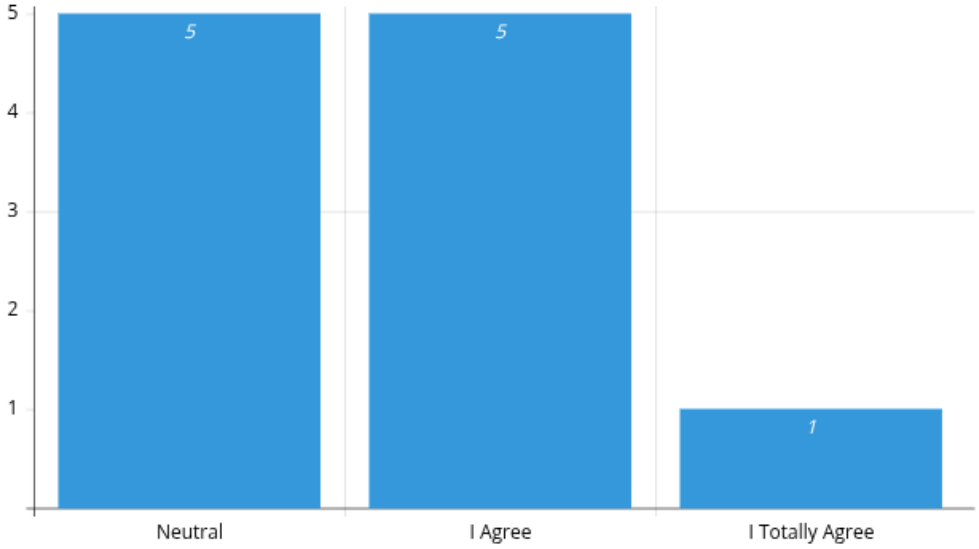

Figure 9 – ISAA Ease of Use

Question Q6, answered in Figure 9, examined the ease of use of ISAA. Responses revealed a more balanced distribution: six participants agreed or strongly agreed that the framework is easy

to use, while five selected the neutral option. The result suggests that the framework's usability is still under evaluation and might benefit from clearer implementation guidelines, additional user support, or simplification of certain procedures.

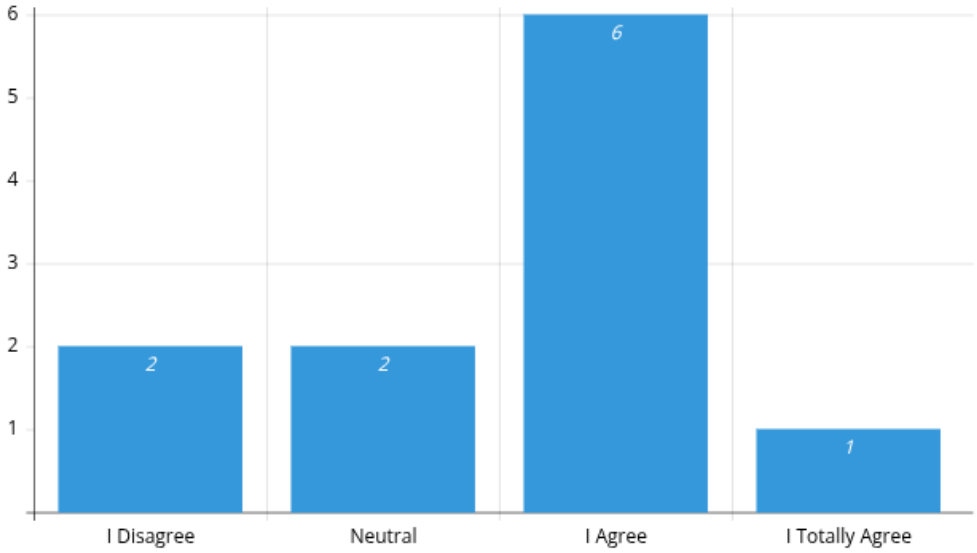

Figure 10 – Adoption in Projects Where You Work

Question Q7, illustrated in Figure 10, explores participants' willingness to adopt ISAA in their organizations' projects. Responses varied: seven respondents expressed confidence in ISAA's potential and stated that they would consider using it in their work environments. Conversely, two respondents adopted a neutral stance, citing a lack of practical experience with the framework as the main reason for their indecision. The remaining two respondents indicated that they would not adopt ISAA, primarily due to a preference for more familiar and simplified solutions to common problem situations, which, according to these participants, rarely require the comprehensive analysis that ISAA requires. Furthermore, some noted that the potential benefits of using ISAA might not outweigh the effort needed for its implementation, unless the organization faces persistent issues with systems failing to meet requirements.

Question Q8, answered in Figure 11, asked participants whether ISAA addresses architectural aspects aligned with industry best practices. The responses were largely positive: eight participants agreed that the tasks proposed by ISAA are consistent with commonly adopted practices in the software industry. However, three participants selected a neutral response, suggesting uncertainty or the need for more detailed information to confirm such alignment.

Question Q9, depicted in Figure 12, evaluates ISAA's flexibility in adapting to new technologies and emerging trends in Software Architecture. The results were remarkably favorable: ten of the eleven participants agreed or strongly agreed that ISAA has the ability to incorporate new tasks and adapt to technological evolution. Only one participant selected a neutral option, indicating a degree of uncertainty regarding the framework's adaptability.

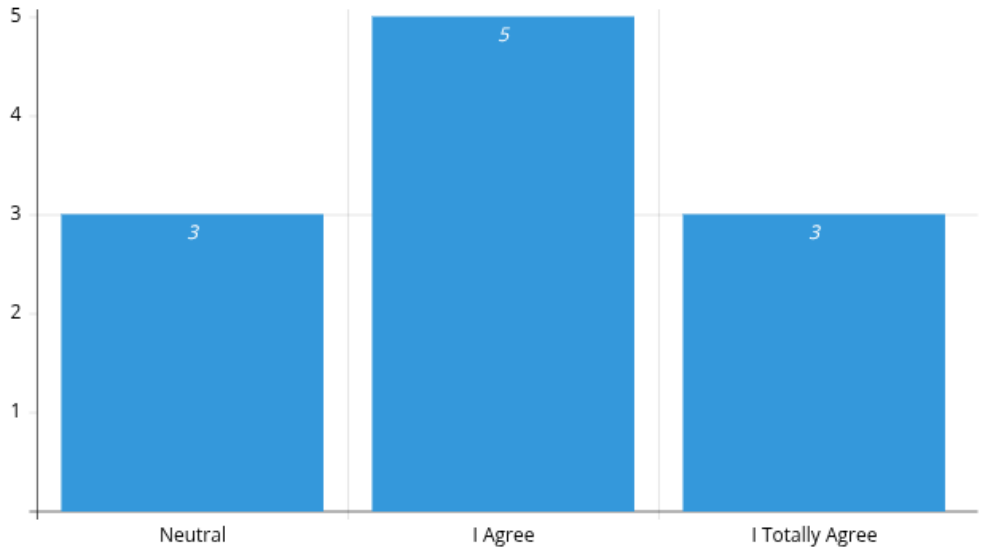

Figure 11 – Aligned with Industry Best Practices

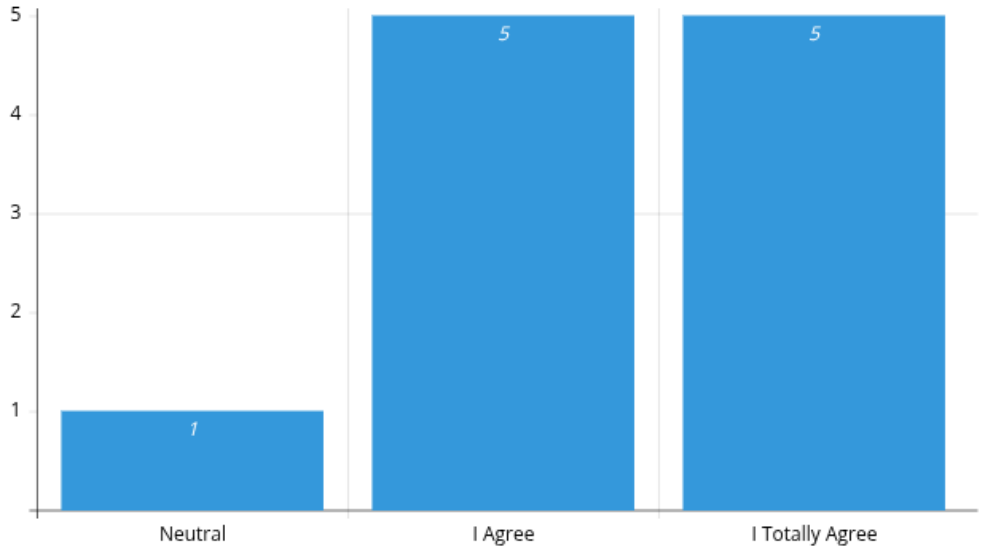

Figure 12 – Able to Adapt to New Technologies and Trends

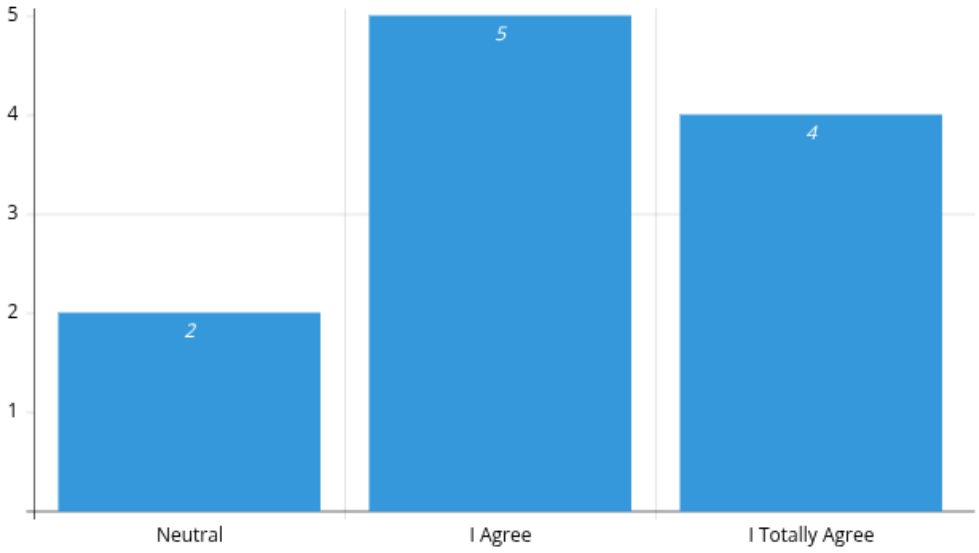

Figure 13 – Effectiveness in Identifying Architectural Risks

Question Q10, answered in Figure 13, addresses ISAA's effectiveness in identifying architectural risks before they materialize during development. Data show that nine respondents believe ISAA is effective in this regard. However, two respondents chose a neutral answer, reflecting some uncertainty regarding the framework's ability to consistently detect potential architectural issues early in the development process.

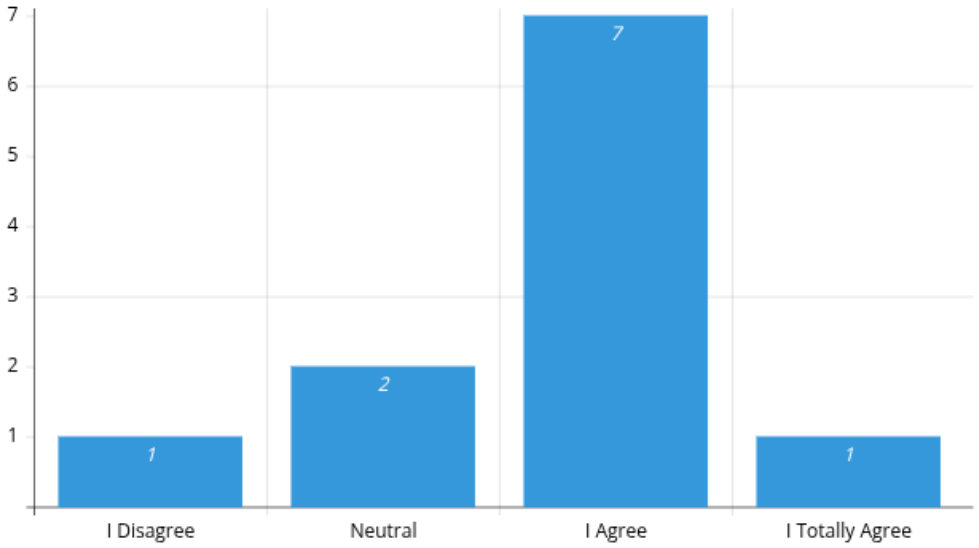

Figure 14 – Integration with Everyday Development Practices

Question Q11, answered in Figure 14, examines the feasibility of integrating ISAA into day-to-day development activities. The results were generally positive: eight participants agreed or strongly agreed that such integration is possible. Meanwhile, two participants expressed uncertainty, and one participant disagreed. Participants who did not consider ISAA suitable for daily use highlighted that its formal and extensively documented methodology may be better

suited for large projects. In smaller projects, they argued, the benefits may not justify the additional effort required for implementation.

Question Q12 asked participants to compare ISAA with other Software Architecture assessment frameworks based on their professional experience. The responses revealed that many participants lacked direct experience with formal Software Architecture evaluation methods. However, those familiar with established approaches such as ATAM (Architecture Tradeoff Analysis Method), RUP (Rational Unified Process), and ISO/IEC 25010 provided meaningful comparisons with ISAA.

Several participants also mentioned ISO/IEC 25010, which provides quality attributes for software product evaluation. However, unlike ISAA, it does not directly support the selection or assessment of architectural alternatives an important point of distinction. TOGAF was briefly cited as well, with some respondents suggesting that combining ISAA with other frameworks could lead to more comprehensive outcomes.

Finally, a fraction of participants reported not using any formal Software Architecture evaluation frameworks in their practice, highlighting an opportunity for broader dissemination and adoption of ISAA within the software development industry.

Question Q13 asked participants what distinguishes ISAA from other Software Architecture evaluation methods. Respondents identified several aspects that set ISAA apart from existing frameworks, including:

1. **Interactivity and Flexibility**: The framework can be tailored to the specific needs and context of each project.
2. **Customisation and Expandability**: ISAA supports adjustments based on architectural complexity and project-specific requirements.
3. **Solid Normative Foundation**: Alignment with ISO/IEC/IEEE 42020:2019 contributes to greater standardisation and reliability in the evaluation process.
4. **Continuous Monitoring**: Enables structured tracking of architectural evolution throughout the project lifecycle.
5. **Risk Prevention**: Facilitates early identification and mitigation of architectural risks before they affect software development.

Additionally, participants emphasised that ISAA fosters continuous communication among stakeholders, contributing to improved project coordination and strategic alignment. The availability of ongoing feedback was seen as a key mechanism for maintaining architectural alignment with evolving business objectives.

Another noteworthy feature mentioned was ISAA's incremental nature, which makes it particularly suitable for microservices-based systems and agile development environments. Nonetheless, some respondents acknowledged that, despite its benefits, ISAA may not fully replace more established frameworks such as ATAM especially in contexts involving critical architectural tradeoffs and high-risk decisions.

Finally, participants noted that ISAA holds significant value in educational and institutional settings, offering a structured and conceptually grounded approach to architectural evaluation while avoiding the procedural overhead often associated with traditional frameworks.

Question Q14 asked which aspects of ISAA should be improved or expanded. Participants identified several opportunities for enhancement, emphasizing the need to make the framework more accessible, automated, and aligned with contemporary development practices.

The main suggestions for improvement include:

1. **Formalisation of Standardised Criteria**: Promoting greater consistency and objectivity in the framework's application.
2. **Process Automation**: Reducing manual effort and facilitating adoption in day-to-day development workflows.
3. **Integration with Agile Methodologies**: Ensuring compatibility with the iterative and dynamic nature of modern software processes.
4. **Adaptation to Specific Domains**: Providing tailored guidelines for particular contexts, such as microservices and embedded systems.

Additional suggestions included restructuring certain activities to create a more linear and intuitive process especially regarding the relationship between architectural option prioritisation, criteria definition, and continuous monitoring. Some participants proposed enhancing the monitoring phase through dashboards capable of presenting metrics, improvements, and ongoing changes throughout the development lifecycle.

Participants also noted the need for greater clarity in the documentation of ISAA tasks. Although the framework emphasises flexibility, its current descriptions may unintentionally suggest a rigid sequence of steps. To support broader adoption, the inclusion of templates, checklists, and concrete examples was recommended.

Finally, respondents highlighted the importance of placing greater emphasis on specific quality attributes such as performance and security and improving integration with agile practices, DevOps environments, and empirical mechanisms like Proofs of Concept (PoCs) and measurable indicators.

Question Q15, the final question, asked participants whether there were any functionalities or features they would like to see incorporated into the ISAA framework. Respondents suggested several enhancements aimed at broadening ISAA's applicability across different contexts and facilitating its practical adoption. The main suggestions include:

1. **Risk Prioritisation Mechanisms**: Enabling architectural decisions to consider and address potential risks in a structured manner.
2. **Support for Runtime Assessment**: Particularly for cloud-based systems, allowing for continuous evaluation of architectural performance and compliance during execution.
3. **Enhanced Monitoring Capabilities**: Emphasizing the importance of ongoing architectural reviews as projects evolve.
4. **Targeted Recommendations for Evaluation**: Providing clearer guidance on how to assess specific aspects of the framework and define criteria for selecting architectural alternatives.
5. **Architecture Reassessment Mechanisms**: Reinforcing the need to revisit and update architectural decisions in response to significant project changes, acknowledging the dynamic nature of Software Architecture.

Additionally, participants pointed out that ISAA could place greater emphasis on the testability of architectural deliverables, ensuring that decisions are validated in a more rigorous and structured way.

Other suggestions included the incorporation of templates, checklists, and visual diagrams to make the framework more accessible and user-friendly in everyday development scenarios. The creation of a dedicated space for collecting feedback from various stakeholders, including developers, clients, and end users was also recommended, to integrate diverse perspectives into the evaluation process.

Finally, a noteworthy recommendation was the inclusion of a mechanism to compare the planned Software Architecture with the implemented one. Such comparison would help identify deviations and generate insights that could inform and enhance future projects.

4.4 Interviews

A total of 11 individuals participated in the survey. From this group, the six participants with the most experience in software development and Software Architecture were selected and subsequently invited to participate in follow-up interviews. A set of six structured questions was developed to guide these interviews, aiming to gather insights that could inform the future evolution of ISAA, particularly the design of a potential version 2 (V2) incorporating the improvements identified by the interviewees.

4.4.1 Q1 - What common challenges arise when evaluating Software Architectures?

The interviewees identified several recurring challenges in evaluating Software Architectures. One of the main issues highlighted was the absence of clear and structured information during the early stages of projects. Such lack of definition hinders the development of a consistent architectural plan from the outset, disrupting the alignment between technical decisions and business objectives and increasing the likelihood of rework or misguided choices.

Another major challenge involves the need for continuous architectural observability. Systematic monitoring is essential to detect technical debt, anticipate failures, and ensure architectural sustainability throughout the software lifecycle. In this context, participants also emphasised the difficulty of validating architectural resilience under stress conditions, particularly in web applications, where Software Architectures may appear robust in theory but fail to perform as expected under high data volumes or heavy traffic.

Financial considerations were also cited as a critical factor. An initially promising Software Architecture can become economically unfeasible when scalability demands significant infrastructure investments. Moreover, integrating new Software Architectures into existing environments in a decoupled and efficient manner remains a complex activity that often impacts architectural viability.

The lack of specific, actionable metrics for identifying bottlenecks and evaluating architectural effectiveness was also mentioned. In many cases, available measurements are too ambiguous or insufficient, limiting the ability to make informed decisions. Lastly, interviewees pointed out the frequent tension between technical and business teams: while stakeholders often prioritise rapid, functional deliveries, development teams face pressure to make architectural choices without due consideration of non-functional requirements, potentially compromising solution quality and long-term maintainability.

4.4.2 Q2 - Are some of these challenges not met by ISAA?

The interviewees recognised ISAA's generic and adaptable nature but identified notable gaps in its practical applicability. First, they emphasised the absence of concrete examples and case studies, particularly in domains such as DevOps, where detailed guidance on subtasks and real-world workflows is lacking. Such limitation hampers the understanding of how to effectively translate the framework's theoretical concepts into everyday practice, thereby reducing its usefulness as an operational guide.

A second limitation concerns the integration with external tools. Currently, ISAA does not offer automated mechanisms to synchronize its documentation with version control systems, CI/CD pipelines, or activity tracking platforms. Consequently, teams must maintain the architectural artifacts manually, which conflicts with agile methodologies that emphasise rapid

iteration and can undermine the credibility and accuracy of architectural records.

Furthermore, interviewees pointed out that certain specialised domains require more tailored approaches than the general framework ISAA provides. For instance, mission-critical systems governed by strict regulations, such as financial applications regulated by standards like BIAN for fintechs, demand dedicated frameworks that embed more rigorous controls and compliance requirements. In such contexts, a generic framework alone lacks the necessary depth and precision to ensure robustness and regulatory adherence (ABEDINIYAN; AZAR; NAZEMI, 2021).

Finally, the interviewees highlighted the challenge of integrating a new Software Architecture into an existing environment without operational disruptions or losses. Each organisation has its unique methods encompassing distinct processes, tools, and cultural aspects. ISAA currently lacks explicit guidelines to align its architectural planning with these internal factors, which may result in misalignments and rework during implementation.

4.4.3 Q3 - How does ISAA adapt to different project domains and scales?

ISAA's modular structure was highlighted as a key strength, offering the flexibility required for teams to tailor their activities to the specific characteristics of each project. Such adaptability facilitates adoption across diverse organisational contexts, particularly where process customisation is possible. However, the interviewees observed that flexibility alone does not fully address practical challenges. In environments demanding automated integration with CI/CD tools, monitoring systems, or complex pipelines, ISAA lacks sufficient technical depth, limiting its effective application in these scenarios.

Another issue raised concerns the absence of domain-specific guidelines, particularly for specialised areas like DevOps, which undermines the applicability in multidisciplinary or highly technical projects. While the framework's generic nature is advantageous for initial adaptation, it can become a constraint when precise instructions for particular workflows are necessary.

Additionally, participants reported that the effectiveness of ISAA varies according to the usage context. Professionals unfamiliar with architectural assessment frameworks tend to find ISAA's structure adequate for their needs. Nevertheless, this perception may not reflect the greater complexity present in more robust or regulated projects, where dedicated frameworks prove more suitable.

Finally, it was emphasised that ISAA's suitability is directly influenced by the scale and maturity of the project or team. In smaller initiatives, the effort required to implement and sustain the framework might not be justifiable, rendering its adoption disproportionate to the expected benefits. Therefore, the decision to use ISAA should always consider the organisational context, balancing implementation effort against practical value.

4.4.4 Q4 - Are there any noticeable limitations in ISAA in terms of integration with other tools or methodologies already used in the industry?

Criticism of ISAA frequently centers on its limited interoperability with established industry tools. The absence of native support for integration with backlog management systems, automated documentation platforms, and agile pipelines constitutes a significant barrier, particularly for organisations that adhere to agile methodologies and prioritise rapid delivery cycles. In such environments, this lack of integration creates operational friction, necessitating manual effort to keep architectural artifacts synchronized with continuous development workflows.

Despite these technical constraints, ISAA proves valuable in environments where architectural documentation is prioritised. In such contexts, the framework supports the standardisation of technical decisions and facilitates the onboarding of new team members by promoting clarity and consistency in established practices. The usefulness of ISAA, therefore, is closely linked to the degree of process formalisation within the organisation.

Another notable aspect is ISAA's flexibility within agile environments. Modular structure permits the inclusion or exclusion of subtasks as necessary, making it compatible with iterative approaches, provided that adaptations are well justified. However, this same flexibility may be regarded as insufficient in highly regulated domains such as financial, military, or aerospace sectors, where stringent standards and formal control over development processes are mandated.

Consequently, ISAA proves to be a useful and adaptable framework for organisations with less prescriptive processes but faces challenges when applied in contexts demanding rigorous regulatory compliance and sophisticated technical integrations. Therefore, the decision to adopt ISAA should carefully balance flexibility, rigour, and compatibility with the existing technological ecosystem.

4.4.5 Q5 - How would you rate the usefulness and learning curve of ISAA?

The usefulness of ISAA is widely acknowledged for its ability to systematise the documentation and evaluation of architectural decisions. The framework promotes structured technical discussions and encourages the continuous recording of changes, enhancing the traceability and coherence of decisions throughout the project life cycle. Such an approach facilitates communication among team members and supports technical alignment, particularly in environments where architectural governance is prioritised.

Nevertheless, the effective adoption of ISAA depends heavily on the technical maturity of the teams involved. Professionals with prior experience in Software Architecture generally perceive the learning curve as straightforward, whereas teams with less familiarity or those operating within less structured processes may face difficulties applying the framework effectively.

Such dependence highlights the need for more didactic support materials, practical examples, and targeted training to enhance accessibility and usability.

Additionally, some interviewees pointed out that, despite its direct and objective design, the introduction of ISAA adds an extra layer of process for teams. In contexts where agility and rapid delivery are emphasised, the framework may be perceived as bureaucratic or as an impediment to development speed, particularly when its immediate practical benefits are not clearly communicated.

Overall, ISAA's learning curve is comparable to that of other well-established methodologies in the software industry. However, its true usefulness is contingent upon the organisational context and the technical maturity of the team, necessitating a balance between process formalisation and development agility.

4.4.6 Q6 - If you could recommend one improvement based on current Software Architecture trends, what would it be?

Based on the feedback collected, two priority improvements emerge for the evolution of ISAA. The first involves incorporating observability mechanisms that enable continuous monitoring of architectural metrics such as performance, cost, and reliability. Such capability would facilitate real-time assessment of the Software Architecture's health and support the identification of bottlenecks and technical debt throughout the software lifecycle. The second improvement concerns automating architectural documentation through integration with widely used tools like Swagger, Confluence, and Git. Automation would reduce the manual effort required to update artefacts, increasing the framework's adoption, particularly in environments that follow agile methodologies and accelerated delivery cycles.

Beyond these core improvements, interviewees suggested expanding ISAA's scope to encompass domains related to modern Software Engineering practices, including DevOps. The inclusion of guidelines for continuous integration and delivery pipelines, infrastructure as code (IaC), and operational automation would render the framework more comprehensive and aligned with current industry demands.

Another critical point identified is the need to simplify ISAA's processes to mitigate negative impacts on team productivity. In high-velocity delivery contexts, frameworks perceived as overly bureaucratic increase risk rejection due to added workflow overhead. Consequently, a leaner and more adaptable ISAA, better suited to the realities of development teams, would likely gain broader acceptance.

Finally, interviewees emphasised the importance of ISAA contributing to the generation of metrics that provide objective insights into architectural quality. The absence of clear indicators in the industry is a persistent gap, and by offering mechanisms to measure quality, performance, and adherence to architectural decisions, ISAA could enhance its perceived value and encourage

wider adoption.

5

Future Trends for Software Architecture

The evolution of systems is closely linked to the emergence of new technological paradigms and the growing demand for security, resilience, and scalability. Softwares, often responsible for supporting essential services such as healthcare, finance, transportation, and energy, cannot afford downtime or failures, as their unavailability can result in serious social and economic impacts.

Future trends in Software Architecture point to a hybrid scenario: on one hand, the continuous integration of disruptive technologies such as artificial intelligence, 5G, and quantum computing; on the other, the adoption of architectural paradigms and evaluation methods that ensure robustness, adaptability, and compliance with increasingly demanding standards. Transformations are not isolated, but interconnected, converging toward the creation of ecosystems capable of withstanding complex physical and digital threats.

The following sections discuss the main technological trends that are reshaping Software Architectures.

5.1 Artificial Intelligence in Software Architecture Assessment

Artificial intelligence (AI) has shown significant potential in helping with the evaluation of Software Architecture, transforming traditional approaches, and offering new possibilities for optimisation, consistency, and decision-making. AI tools, such as those based on Large Language Models (LLMs), are increasingly integrated into software development workflows, redefining the way architects approach modern design (RAMACHANDRAN, 2025).

Traditionally, software architects employ a variety of tools to create architectural designs, including UML diagrams, such as Use Case, Class Sequence and State Machine diagrams. Many of these diagrams follow repetitive structures that can be automated with well-defined contexts. Assisted AI tools such as GitHub Copilot, ChatGPT and BlackBoxAI are being evaluated for

their ability to generate these designs based on contextual prompts (RAMACHANDRAN, 2025).

Applying AI to Software Architecture evaluation can improve productivity, ensure design consistency, and support architectural decision-making (RAMACHANDRAN, 2025). Studies have explored the effectiveness of these tools in generating diagrams, with comparative analyses seeking to determine how closely the results generated by AI align with established architectural principles (RAMACHANDRAN, 2025). Refined prompts and context addition can significantly improve the accuracy of outputs (RAMACHANDRAN, 2025).

The methodology for comparing AI and human-generated designs generally involves the creation of initial prompts from requirements documentation, the generation of diagrams by AI tools, and then the calculation of a similarity score between the generated diagrams and the reference diagrams (RAMACHANDRAN, 2025). This score takes into account the number of corresponding nodes and edges to assess structural similarity. Iteration in the refinement of prompts is crucial to achieving higher similarity scores, with fewer iterations indicating greater accuracy and effectiveness of the tool (RAMACHANDRAN, 2025).

While AI tools have shown excellence in fault detection and representation in Use Case, Sequence and State Machine diagrams, they can still present challenges with Class diagrams, where traditional literature can provide more robust models (RAMACHANDRAN, 2025). However, more recent models, such as BlackBoxAI with DeepSeek-V3 and ChatGPT with GPT-4o, tend to produce superior UML diagrams compared to earlier versions, such as GitHub Copilot based on GPT-4. This suggests that with the evolution of models, AI tools can generate designs that surpass those created by humans (RAMACHANDRAN, 2025).

Despite these advances, there are ongoing challenges in applying AI to Software Architecture. The confidentiality of companies' architectural data limits the availability of public data for training specific models for Software Architecture (JOSHI, 2024). In addition, security, privacy and ethical considerations are paramount, especially when dealing with sensitive data (BAO et al., 2023; JOSHI, 2024). A digital code of ethics for the use of AI in Software Engineering is needed to protect fundamental human values (BAO et al., 2023). The ability to explain the decisions made by AI models (Explainable AI - XAI) is also a significant challenge, as the "black box" nature of some models can diminish user confidence (AHMED et al., 2025).

Artificial intelligence is revolutionising Software Engineering by offering innovative support across various phases of the Software Architecture lifecycle. In the architectural analysis phase, which involves identifying new requirements, AI tools, such as ChatGPT, can articulate architecturally significant requirements (ASRs) based on an "Software Architecture story" provided in natural language. The architect offers a narrative text of the proposed solution, and ChatGPT can automatically generate functional requirements, non-functional requirements, and relevant constraints. The process is iterative, allowing the architect to refine (add, remove, update) these requirements through a continuous dialogue with the AI. This accelerates the initial elicitation and specification phase, ensuring that the AI understands the context and needs of the

software (AHMAD et al., 2023).

When it comes to scoring an Software Architecture to verify that it meets stakeholders' intentions, AI can play a role in the architectural evaluation phase. Once ASRs are synthesized into an architectural model (such as UML diagrams), AI can be employed to evaluate this design against requirements. For example, ChatGPT can be instructed to apply architectural evaluation methods, such as the Software Architecture Analysis Method (SAAM), to analyse specific software components. AI can identify and prioritise Use Cases to assess functionality, quality (response time), and constraints (data privacy). This allows architects to delegate evaluation tasks and gain insight into the design's compliance with stakeholder objectives, even though human intervention is still required to ensure the consistency and ethics of the generated outputs (AHMAD et al., 2023).

Supervised machine learning techniques, trained on historical data extracted from code metrics and architectural patterns in real projects, have been used to predict architectural decisions, such as the adoption of patterns (e.g., MVP or MVVM). A study conducted with 5973 Android projects identified important code metrics and used eight ML models (including Random Forest and SVM), achieving high accuracy in detecting the architectural pattern adopted by developers, revealing the ability of these models to predict decisions based on real data from public projects (KOMOLOV et al., 2022).

In the future, AI could become even more integrated into Software Engineering processes, augmenting human capabilities and enabling transformative breakthroughs (JOSHI, 2024). AI-based tools will automate routine tasks, optimise workflows and resource allocation, allowing software engineers to focus on creative problem solving. Future research should also address the generation of large, high-quality and unbiased datasets for training AI models, as well as the development of reliable benchmarks for an unbiased evaluation of all Software Engineering tasks (AHMED et al., 2025). Collaboration between humans and AI in Software Engineering teams will require an understanding of how to interpret AI-based decisions and recommendations (AHMED et al., 2025).

5.2 5G and High-Speed Networks

The advancement of 5G networks, and their evolution to Beyond 5G, is one of the central pillars for defining new trends in Software Architecture. This technology is not limited to increasing transmission rates, but also enables an ecosystem of distributed applications that require low latency, high reliability, and support for extreme mobility. For Software Architecture, this means designing systems that are resilient, adaptive, and capable of integrating with dynamic network infrastructures.

Studies on communications in high-mobility scenarios, such as those conducted by Han et al., demonstrate how beamforming techniques, channel prediction, and the use of location data

can significantly increase spectral efficiency (HAN et al., 2019). Although focused on the context of high-speed trains, these principles have broader implications: distributed system Software Architectures could benefit from similar strategies to ensure quality of service in dense urban environments, autonomous vehicles, or large-scale sensor networks.

Similarly, the work of Noh et al. demonstrates that 5G NR incorporates elements designed to handle scenarios of intense variability, such as flexible OFDM (Orthogonal Frequency Division Multiplexing) numerologies, adaptive reference signals, and new handover mechanisms with near-zero interruptions (NOH; HUI; KIM, 2020). This technical capability reinforces 5G's role as an infrastructure on which software Software Architectures can implement critical services, from real-time telemedicine to IoT-based industrial applications, requiring new standards of interoperability, elasticity, and continuous monitoring.

A relevant aspect for software architects is the convergence between 5G, edge computing, and distributed cloud computing. With the decentralization of edge processing, it becomes necessary to design Software Architectures capable of orchestrating components across multiple network layers, balancing loads, tolerating failures, and ensuring end-to-end security. Furthermore, in mission-critical systems such as industrial automation, digital healthcare, or smart transportation, sub-millisecond latency and massive device communication open up space for event-driven and self-adaptive Software Architectures that adjust according to the state of the network.

In this sense, standards such as ISO/IEC/IEEE 42020:2019 can play a central role in the methodological adaptation required to address the emerging demands of new infrastructures. According to recent studies (COSTA; SOARES, 2023b), architectural standards must evolve to encompass quality attributes related to resilience, interoperability, and continuous evolution, which become increasingly critical in ecosystems permeated by 5G and Beyond 5G technologies. Consequently, the impact of 5G on Software Architecture extends beyond mere connectivity; it represents a structural transformation that redefines how large-scale distributed systems are designed, evaluated, and sustained, making them increasingly dependent on close interaction with the underlying network infrastructure.

5.3 Cloud Computing

The evolution of cloud computing has been reshaping the way software systems are designed, deployed, and maintained. One of the key points is the integration between cloud computing and intelligent industrial systems, especially in Industry 4.0. The author Liu highlights that the adoption of cloud platforms in production lines, combined with advanced algorithms such as fuzzy PID, enables not only remote and real-time monitoring but also intelligent and predictive decision-making (LIU, 2024). This trend reinforces that, in the future, the cloud will be the backbone for autonomous and highly adaptive industrial Software Architectures.

In the field of Software Engineering and legacy systems reengineering, Raja et al. argue that the cloud represents a strategic opportunity for application modernization, especially through practices such as Business Process Model and Notation (BPMN) and reverse engineering techniques (RAJA et al., 2023). This makes it possible to gradually migrate legacy systems or completely restructure them on service-based platforms, ensuring greater scalability, interoperability, and quality of service (QoS). A key trend is the consolidation of tools and methodologies that facilitate this migration process, reducing the costs and risks associated with replacing critical systems still in operation.

Besides technical advances, cloud computing also stands out for its economic benefits compared to maintaining on-premises servers. As Raja et al. note, the cloud-based model allows organizations to reduce infrastructure costs by avoiding high initial investments in hardware, energy, and maintenance. Instead of purchasing and upgrading their own servers, companies are adopting a pay-as-you-go model, making the environment more flexible and responsive to demand (RAJA et al., 2023). This economic factor is particularly relevant for small and medium-sized companies, which can access advanced, high-availability computing resources without having to maintain dedicated data centers.

In the automotive sector, cloud platform Software Architecture is becoming essential to handle the growing volume of data generated by connected and autonomous vehicles. The authors Benijamali et al. point out that research and development in this area has been exploring solutions to balance scalability, latency, and security in distributed applications, such as advanced driver assistance systems (ADAS) and smart mobility platforms (BANIJAMALI et al., 2019b). In this context, it can be seen that the future of the cloud is moving toward increasingly domain-specific models, optimizing Software Architectures to meet the demands of specific sectors.

More broadly, future trends in cloud computing point to three major movements:

1. sector specialization, with Software Architectures adapted to industries such as healthcare, automotive, and manufacturing.
2. integration with edge computing, expanding the distributed hybrid computing paradigm.
3. trengthening cloud-oriented software engineering, including reengineering, process automation, and greater standardization in critical Software Architectures.

Such advances not only reinforce the cloud's role as a global infrastructure but also expand its relevance in supporting critical systems.

5.4 Edge Computing and IoT Integration

Edge computing emerges as a direct response to the limitations of the centralized cloud computing model, especially in Internet of Things (IoT) scenarios and latency-sensitive

applications. As billions of connected devices begin to generate data on a large scale, the need for processing close to the source becomes essential. This approach not only reduces bandwidth overhead on central servers but also enables near-real-time responses, an indispensable requirement for critical and distributed systems (DAYOUB, 2025; NAVEEN; KOUNTE, 2019).

In the IoT domain, integration with edge computing Software Architectures has proven strategic. In contrast to the traditional three-tier model (sensing, network, and cloud), edge-based Software Architectures add an additional layer responsible for local data preprocessing and analysis. This results in greater energy efficiency, lower latency, and increased software resilience. Recent studies demonstrate that, in mobility scenarios, the use of platforms such as AdvantEDGE reduces latency to approximately 30–50 ms, demonstrating the effectiveness of Multi-access Edge Computing (MEC) in time-sensitive IoT applications (DAYOUB, 2025).

Another emerging aspect is the integration of Edge Computing and blockchain, aiming to address the challenges of security, privacy, and resource allocation in distributed IoT systems. As discussed by Zihan et al., the use of blockchain in conjunction with Mobile Edge Computing (MEC) allows for the creation of decentralized consensus mechanisms to manage authentication, storage, and data sharing. Furthermore, intelligent resource allocation algorithms can optimize both energy consumption and latency, contributing to more scalable and reliable Software Architectures in large-scale environments (BAI et al., 2022).

Discussions reinforce the need for Software Architecture to evolve toward hybrid and decentralized models, in which cloud and edge work seamlessly. This imposes new demands on software architects: dealing with device heterogeneity, designing context-aware adaptive systems, and ensuring interoperability between multiple processing layers. The result is a transformation of the architectural paradigm, in which the edge is not just a supporting resource, but a strategic element for sustaining the next generation of distributed and mission-critical systems.

5.5 Security and Cyber-Resilience

Future Software Architectures must evolve beyond merely delivering functionality, integrating security and resilience as fundamental pillars from their inception. This paradigm shift requires a clear distinction between cybersecurity and cyberresilience. While cybersecurity focuses on protecting IT assets, such as data, cyberresilience is defined as the ability of a software or organization to defend itself against successful cyberattacks and revert to a normal operational state when defenses fail, as Alhidaifi argues. Therefore, the Software Architecture of the future can no longer treat security as an external layer or an afterthought where it must be designed with the assumption that failures will occur, focusing on the software ability to prepare for, absorb, recover from, and adapt to adverse events that affect business operations (ALHIDAIFI; ASGHAR; ANSARI, 2024).

An architectural trend for addressing this new reality is the “System of Systems” (SoS)

approach. Modern applications are no longer monolithic, isolated systems, but rather decentralized, distributed, and interoperable compositions of heterogeneous and semi-autonomous systems (SHARKOV, 2017). This global interconnectivity, while functionally powerful, introduces complex risks, such as unforeseen emergent behaviors and cascading failures, where an incident in one component can propagate and escalate throughout the entire network of dependent services. Consequently, the traditional perimeter-based defense model becomes obsolete, forcing architects to adopt models such as "Zero Trust" in which no network traffic is considered trustworthy by default, regardless of its origin (SHARKOV, 2017).

To manage the complexity inherent in Systems of Systems, future Software Architectures will require strategies that reduce the scope of the problem. One promising approach is the definition of an "Operational Design Domain" (ODD) for each software component. ODD delimits the set of operational conditions under which a software or one of its functionalities is specifically designed to function (MISON; DAVIES; EDEN, 2024). By focusing on operational intent and acceptable behaviors within well-defined boundaries, developers can implement an advanced form of defensive programming. This allows for more effective design and testing of resilience, ensuring that, even in the face of unforeseen parameters or scenarios, the software behaves as expected, allowing it to degrade gradually and safely rather than fail catastrophically (MISON; DAVIES; EDEN, 2024).

However, designing for resilience is not enough if its effectiveness cannot be proven. One of the most critical and urgent trends is the development and adoption of objective and quantifiable security metrics. Currently, the software industry lacks a reliable software for measuring security, leading to procurement and Software Architecture decisions based on vendor claims, anecdotal reports, or personal experience rather than hard data (EGGENDORFER; ANDRESEN, 2024). Software Architecture needs standardized methods for assessing and comparing the security level of different components and systems. The existence of such metrics would not only empower architects to make informed decisions but also create a market incentive for vendors to genuinely invest in more secure products, increasing collective cyber resilience.

Finally, the growing complexity of threats and Software Architectures will require a reassessment of competencies and greater team specialization. The notion of the IT specialist who masters all facets of security is unsustainable. The trend points toward a clearer separation of responsibilities: development and Software Architecture teams will focus on building resilience and dependability, primarily within the ODD of their applications. In parallel, specialized cybersecurity teams will focus on defending the underlying infrastructure, such as networks, operating systems, and endpoints. This division of labor allows each group to utilize their skill set more effectively, addressing the skills gap and ensuring that both the business logic and the infrastructure that supports it are robustly protected.

5.6 Microservices Architectures

A dominant trend in modern Software Architecture is the decomposition of legacy monolithic applications into microservices Software Architectures, driven by the promise of greater scalability, agility, and failure resilience. However, this transition is a complex and expensive undertaking, still in its early stages of maturity. Abgaz et al. systematic survey reveals that, although several approaches have been proposed, consolidated methods for combining different types of analysis data, such as static, dynamic, and evolutionary data, are lacking, and automated tool support remains insufficient (ABGAZ et al., 2023). This fragmentation leads to a lack of standardized metrics and benchmarks to assess the quality of the resulting microservices, making the decomposition process more of a craft exercise than a well-defined engineering discipline.

The complexity introduced by the distributed and heterogeneous nature of microservices poses significant challenges not only in decomposition but also in software testing. A systematic literature review on the topic highlights that the field of microservices testing, while progressing, remains fragmented, with most proposals in early stages of maturity and low levels of technology readiness. This indicates a lack of empirical validation and industry-grade benchmarks that could facilitate wider adoption of robust testing techniques. Research is dominated by laboratory experiments, and there are notable gaps in areas such as flexibility testing and, surprisingly, security testing, which were not directly addressed by any of the primary studies analyzed (ABGAZ et al., 2023).

Even after decomposition, ensuring a software ability to be adapted or extended efficiently remains a critical challenge. A study by Bogner et al. reveals a fundamental tension between decentralization, which promotes team autonomy, and the need for standardization to maintain consistency. Practices such as defining architectural principles, test automation, and the use of patterns such as Event-Driven Messaging are considered crucial (BOGNER et al., 2021). However, most industry tools and metrics still focus on the code quality of individual services, neglecting macro-architectural assessment.

The most obvious gap in industrial practice is the lack of tools and metrics for architectural assessment, even when the most important challenges, such as defining service boundaries and integration, are architectural in nature. Practitioners report not using specialized tools to support decomposition or metrics to assess the quality of the chosen cuts, such as coupling or cohesion at the service level. This is reflected in testing challenges, where software testing emerges as the most investigated level, followed by integration testing, indicating the significant focus on validating the behavior of the entire software and the interaction between services. The predominance of specification-based strategies in functional testing and the lack of focus on quality attributes such as security and flexibility reinforce the need for research and development of more comprehensive, macro-architectural validation approaches (BOGNER et al., 2021).

5.7 Quantum Computing and Post-Quantum Security

The rise of quantum computing represents a paradigm shift that fundamentally redefines Software Engineering, requiring a complete rethinking of its practices, especially in the area of Requirements Engineering (RE). Unlike classical software development, quantum software is intrinsically linked to quantum mechanics, introducing a new level of abstraction and a reliance on properties such as superposition and entanglement. This unique characteristic, where operations are inherently probabilistic, renders traditional RE methods inadequate. The threat becomes even more concrete with "harvest now, decrypt later" attacks, where adversaries capture data encrypted today to decrypt it in the future with quantum computers, posing an immediate risk to long-term sensitive data such as health records (SEPúLVEDA et al., 2024; EZEOGU, 2025).

One of the central challenges in RE for quantum computing (QRE) is the difficulty in defining specific requirements for systems that must be resilient to quantum attacks. Most practical applications will be hybrid systems, integrating quantum and classical components. This requires specifying complex interactions between different computational paradigms and choosing between various quantum security technologies, such as lattice-based cryptography (LBC), quantum key distribution (QKD), and quantum hash functions (QHF). The lack of established standards for RE in this domain exacerbates the problem, hindering consistent requirements collection, analysis, and validation (ALMOTIRI, 2025; SEPúLVEDA et al., 2024).

Another significant challenge is the knowledge gap and rapid technological evolution. There is a shortage of requirements engineers trained in the principles of quantum computing, which limits the ability to effectively translate quantum capabilities into software requirements. Rapid hardware evolution requires specifications to be continually updated to remain relevant. This need for readiness is reinforced by the urgency to develop "quantum literacy" among IT professionals so they can make informed decisions about adopting new technologies and security strategies (SEPúLVEDA et al., 2024; EZEOGU, 2025).

Despite the challenges, quantum computing presents opportunities for advancing RE. Specific techniques are emerging to adapt classical methods to the quantum context, such as the use of multi-criteria decision methodologies, such as the Fuzzy Analysis Hierarchy Process (F-AHP), to evaluate and prioritize the effectiveness of different quantum security approaches. This approach allows for a weighted ranking of security factors, offering a novel perspective for software evaluation in the quantum era. Such frameworks provide the means for architects to make data-driven decisions when selecting the most appropriate protection technologies for their systems (ALMOTIRI, 2025).

Integrating security from the early stages is a paramount concern. Quantum computing threatens classical cryptographic algorithms such as RSA and ECC, which are vulnerable to attacks using Shor's algorithm. This requires a transition to post-quantum cryptography (PQC), with the adoption of NIST-standardized algorithms such as CRYSTALS-Kyber, CRYSTALS-

Dilithium, FALCON, and SPHINCS+. A key architectural trend for managing this migration is "crypto-agility", which enables a smooth transition between traditional and post-quantum cryptographic methods, mitigating risks and ensuring operational continuity. This approach is crucial in environments such as healthcare, where software outages are not an option (AKBAR et al., 2025; EZEOGU, 2025).

In the future, formalizing QRE processes, empirically validating PQC algorithms, and standardizing architectural practices will be important. Implementing PQC introduces performance overheads in latency, storage, and compute, which should be considered as critical non-functional requirements during design. For example, studies show that Kyber-1024 is suitable for cryptographic tasks with minimal overhead, while Dilithium-5 offers a good balance for long-term signature security. Furthermore, the integration of quantum security into CI/CD pipelines, known as Quantum Secure DevOps (QSecOps), faces barriers such as a lack of standardization and the complexity of key management, indicating that the path forward will require both technical innovation and the development of new professional skills (AKBAR et al., 2025; EZEOGU, 2025).

6
Conclusion

The set of studies developed throughout this master's degree demonstrates the conceptual and practical evolution of Software Architecture assessment as a central pillar of Software Engineering. More than a technical activity, architectural assessment emerges as a strategic governance process, responsible for aligning technical decisions with organizational objectives, reducing risks, and supporting the sustainability of software in increasingly dynamic and complex environments. To guide the development of this dissertation and consolidate its contributions, five articles were produced.

The titles and publication status of each article are listed in chronological order:

1. **A Process to Compare ATAM and Chapter 9 of ISO/IEC/IEEE 42020:2019**
    - **Authors**: Gustavo S. Melo and Michel S. Soares
    - **Conference/Journal**: ICEIS
    - **Year of Publication**: 2025
    - **Status**: Published
    - **Reference**: Melo, G. S. and Soares, M. (2025). A Process to Compare ATAM and Chapter 9 of ISO/IEC/IEEE 42020:2019. In Proceedings of the 27th International Conference on Enterprise Information Systems - Volume 2: ICEIS; ISBN 978-989-758-749-8; ISSN 2184-4992, SciTePress, pages 37-46. DOI: 10.5220/0013351800003929
2. **Improving Architecture Assessment: An Analysis of ISO/IEC/IEEE 42020:2019, ATAM, and the Role of Artificial Intelligence**
    - **Authors**: Gustavo S. Melo and Michel S. Soares
    - **Conference/Journal**: Lecture Notes in Business Information Processing
    - **Year of Publication**: 2026

- **Status**: Accepted
- **Reference**: To be assigned upon publication

3. **A Comparison between ATAM and ISO/IEC/IEEE 42020:2019 for Software Architecture Evaluation**
    - **Authors**: Gustavo S. Melo and Michel S. Soares
    - **Status**: Submitted
    - **Reference**: Under review
4. **Architectural Practices for Future Trends in Mission Critical Systems**
    - **Authors**: Gustavo S. Melo and Michel S. Soares
    - **Status**: Submitted
    - **Reference**: Under review
5. **ISAA Framework - An Iterative Software Architecture Assessment Following the ISO/IEC/IEEE 42020 Standard**
    - **Authors**: Gustavo S. Melo and Michel S. Soares
    - **Status**: Submitted
    - **Reference**: Under review

The first three articles provide a comparative analysis between the Architecture Tradeoff Analysis Method (ATAM) and the ISO/IEC/IEEE 42020:2019 standard, two widely recognized references in Software Engineering. The main objective of these studies was to investigate how each approach addresses the documentation, communication, and assessment of quality attributes in Software Architectures, and to clarify their methodological and conceptual differences.

These studies highlight that, while ATAM focuses on analyzing tradeoffs and risks through scenarios, offering a structured and collaborative decision-making process, the ISO/IEC/IEEE 42020:2019 standard adopts a broader normative perspective, describing roles, processes, and artifacts that compose the architectural lifecycle. The comparison between both approaches made it possible to identify advances that occurred over the last two decades, reflecting the evolution of Software Engineering practices and the incorporation of guidance connected to organizational maturity, flexibility, and continuous integration. Overall, these contributions support both practitioners and researchers in selecting and tailoring architectural assessment approaches according to project context and organizational constraints.

The fourth article addresses architectural assessment in mission-critical systems, whose failure can result in significant human, economic, or social losses. In such contexts, Software Architecture must account not only for functional requirements and quality attributes, but also for

mission values, objectives, and fundamental principles that the system must preserve. Specialized literature emphasizes that the success of such software depends directly on the maturity of critical components and the effectiveness of risk management practices (HOULIOTIS et al., 2017; CRUZ et al., 2019). Therefore, architectural assessment in mission-critical environments requires more rigorous and continuous evaluation mechanisms, with an emphasis on resilience, predictability, operational continuity, and proactive risk mitigation.

The fifth article is propositional and experimental in nature, presenting the ISAA (Iterative Software Architecture Assessment) framework. ISAA was created based on the ISO/IEC/IEEE 42020:2019 standard, particularly Chapter 9, and was designed to support architectural assessment iteratively, allowing teams to continuously revisit architectural decisions as the system evolves. Its conception stems from the need to adapt assessment practices to agile and continuous delivery environments, where feedback cycles are faster and flexibility becomes essential to maintain alignment between architectural decisions, quality attributes, and organizational goals.

Taken together, the studies conducted from different perspectives reveal a transformation in how Software Engineering understands, structures, and conducts architectural assessment. Historically, methods such as ATAM have stood out for offering systematic and participatory approaches to identifying risks and tradeoffs between quality attributes. The consolidation of the ISO/IEC/IEEE 42020:2019 standard represents an important step forward by expanding the scope of architectural work to include not only assessment activities, but also development, maintenance, documentation, and continuous integration of Software Architecture throughout the software lifecycle.

This movement reflects a maturation of Software Engineering: a transition from approaches centered primarily on technical analysis toward a holistic and collaborative vision that treats Software Architecture as an evolving asset. By incorporating contemporary practices such as continuous integration and delivery (CI/CD), systematic monitoring, and broader stakeholder participation, the standard contributes to the formalization of architectural governance and supports the consolidation of architecture as an applied and evidence-informed engineering practice.

At the same time, the theoretical and empirical depth provided by the studies reinforces that architectural evaluation is neither uniform nor universal: each context introduces distinct challenges. Mission-critical systems, in particular, demand higher levels of rigor, resilience, and operational continuity, supported by mechanisms for observability, risk management, and cybersecurity. Furthermore, the evolution of edge computing, distributed architectures, and microservices changes the parameters of scalability and reliability, requiring new technical and organizational models for designing and evaluating robust Software Architectures in heterogeneous environments.

In this context, the development of ISAA emerges as a practical response to the gap between standard-level guidance and actionable assessment routines. Based on ISO/IEC/IEEE 42020:2019

and structured iteratively, ISAA provides a framework for continuous assessment that promotes decision traceability, systematic documentation, and alignment with agile practices. Empirical validation with industry professionals indicated its potential benefits, such as strengthening technical governance and reducing uncertainty in architectural decisions. At the same time, the validation also revealed opportunities for improvement, particularly regarding integration with automated tools, the definition of objective architectural quality metrics, and process streamlining for teams with lower technical maturity.

Overall, the results of this research demonstrate that Software Architecture evaluation is more than a technical activity: it is a strategic process for knowledge management, governance, and decision-making. The comparison between approaches, the proposal of an iterative framework, and reflections grounded in mission-critical settings converge toward a common purpose: strengthening the reliability, adaptability, and transparency of architectural decisions across the software lifecycle.

Emerging technologies, particularly Artificial Intelligence and intelligent automation of engineering processes, represent the next frontier for architectural evaluation. Integrating AI into design, evaluation, and optimization activities has the potential to expand predictive capabilities, automate parts of tradeoff analysis, and proactively identify risk and degradation signals. These developments indicate a future in which architectural evaluation becomes increasingly continuous, proactive, and metrics-driven, further consolidating its role as an essential mechanism for ensuring software quality and sustainability.

Findings from this dissertation also indicate that the next generation of assessment frameworks, including a future ISAA V2, should move toward closer integration with contemporary Software Engineering toolchains. This evolution should incorporate automation, continuous observability, and systematic data collection to reduce assessment overhead and improve the reliability of evaluation evidence.

In addition to the contributions achieved by this research, it is important to recognize that the evolution of Software Architecture assessment remains ongoing, influenced by new technologies, paradigms, and organizational demands. Future work should therefore aim not only to refine the ISAA framework, but also to expand its practical validation and theoretical foundations.

The following items outline potential research directions derived from this dissertation. Each of them is presented with a discussion of its relevance, possible methodological approaches, and its connection with current trends and studies in the international research community.

1. **Development of ISAA V2: Integration with Automated Architectural Metrics**
   Relevance: Although ISAA provides an iterative and flexible structure for architectural assessment, its current version depends largely on manual interpretation of results. Developing ISAA V2 with integrated automated metrics would enable objective, data-driven

evaluation of software quality attributes such as performance, modifiability, and reliability.
How the research could be done: Future work could define quantitative indicators for each architectural quality attribute, implement a prototype of ISAA V2, and validate it within continuous integration environments (e.g., Jenkins, GitLab CI). The research could leverage static analysis tools, architectural conformance checkers, and observability platforms to generate assessment dashboards and support continuous evidence collection.
Relation with existing research: International initiatives such as Software Analytics increasingly converge toward data-driven governance of software quality and architecture (BASS; CLEMENTS; KAZMAN, 2021; SILVA et al., 2023). ISAA V2 would align with this trend by bridging qualitative evaluation with automated measurement, strengthening the robustness of architectural decisions.

2. **Empirical Validation of ISAA in Real-World Industrial Scenarios**
Relevance: The current ISAA validation was experimental and exploratory. Applying ISAA in real-world contexts, especially in large-scale or critical systems, would provide stronger evidence regarding effectiveness, scalability, and adaptability to different organizational maturity levels.
How the research could be done: Future work could conduct multiple industrial case studies using Design Science Research as an overarching methodological approach. Partnerships with software organizations would allow the application of ISAA during real assessments and the collection of empirical evidence on effort, risk mitigation, and decision traceability.
Relation with existing research: This direction aligns with Empirical Software Engineering efforts that emphasize industrial validation as a key step toward strengthening scientific credibility and practical relevance of Software Engineering frameworks (PETERSEN et al., 2024).

3. **Incorporation of Artificial Intelligence for Predictive Architectural Assessment**
Relevance: With the evolution of AI-driven Software Engineering, there is a growing demand for tools capable of predicting architectural risks and recommending design alternatives. Incorporating AI into ISAA could transform the framework from primarily reactive assessment to predictive support for architectural governance.
How the research could be done: Future work could explore machine learning models trained on historical architectural data to identify patterns associated with architectural decay, performance bottlenecks, or quality degradation. Natural Language Processing (NLP) could also be used to analyze architectural documentation and extract quality signals automatically.
Relation with existing research: This direction follows current trends in AI-assisted architectural reasoning, where intelligent techniques support evaluation and decision-making. Integrating these techniques would position ISAA at the frontier of next-generation

architecture assessment tooling (MARTÍNEZ-FERNÁNDEZ et al., 2022; LAGO et al., 2024).

Bibliography

ABEDINIYAN, A.; AZAR, F. M.; NAZEMI, E. Designing a Model to Use Omnichannel in Banking Industry Based on BIAN Framework. In: *2021 5th National Conference on Advances in Enterprise Architecture (NCAEA)*. [S.l.: s.n.], 2021. p. 1–10. Citado na página 45.

ABGAZ, Y. et al. Decomposition of Monolith Applications Into Microservices Architectures: A Systematic Review. *IEEE Transactions on Software Engineering*, v. 49, n. 8, p. 4213–4242, 2023. Citado na página 56.

ABRAÃO, S.; INSFRAN, E. Evaluating Software Architecture Evaluation Methods: An Internal Replication. In: *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2017. (EASE '17), p. 144—-153. Citado na página 10.

AHMAD, A. et al. Towards Human-Bot Collaborative Software Architecting with ChatGPT. In: *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2023. (EASE '23), p. 279–285. Citado na página 51.

AHMADI, H. et al. Cross-layer Enterprise Architecture Evaluation: An Approach to Improve the Evaluation of TO-BE Enterprise Architecture. In: *Proceedings of the International Conference on Omni-Layer Intelligent Systems*. New York, NY, USA: Association for Computing Machinery, 2019. (COINS '19), p. 223—228. Citado na página 10.

AHMED, I. et al. Artificial Intelligence for Software Engineering: The Journey So Far and the Road Ahead. *ACM Transactions on Software Engineering and Methodology*, Association for Computing Machinery, New York, NY, USA, v. 34, n. 5, 2025. ISSN 1049-331X. Citado 2 vezes nas páginas 50 and 51.

AHMETI B., L.-M.-G. R. W. R. Architecture Decision Records in Practice: An Action Research Study. In: *European Conference on Software Architecture (ECSA)*. [S.l.: s.n.], 2024. Citado na página 13.

AKBAR, M. A. et al. Quantum Secure DevOps (QSecOps): Integrating Quantum-Based Security Checks Into CI/CD Pipelines: Next-Generation Software Security. In: *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2025. (FSE Companion '25), p. 1732—-1741. ISBN 9798400712760. Citado 2 vezes nas páginas 57 and 58.

AKRAM, F.; AHMAD, T.; SADIQ, M. Recommendation Systems-based Software Requirements Elicitation Process — A Systematic Literature Review. *Journal of Engineering and Applied Science*, v. 71, 02 2024. Citado na página 21.

ALHIDAIFI, S. M.; ASGHAR, M. R.; ANSARI, I. S. A Survey on Cyber Resilience: Key Strategies, Research Challenges, and Future Directions. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 56, n. 8, abr. 2024. ISSN 0360-0300. Citado na página 54.

ALMOTIRI, S. H. Quantum-Resilient Software Security: A Fuzzy AHP-Based Assessment Framework in the Era of Quantum Computing. *PLOS ONE*, Public Library of Science, v. 19, n. 12, p. 1–25, 12 2025. Citado na página 57.

AMORIM, S. d. S.; ALMEIDA, E. S. d.; MCGREGOR, J. D. Scalability of Ecosystem Architectures. In: *2014 IEEE/IFIP Conference on Software Architecture*. [S.l.: s.n.], 2014. p. 49–52. Citado na página 22.

BAI, Z. et al. Resource Allocation of IoT Systems Integrated with Blockchain and Mobile Edge Computing. In: *2022 4th International Conference on Frontiers Technology of Information and Computer (ICFTIC)*. [S.l.: s.n.], 2022. p. 377–382. Citado na página 54.

BALALAIE, A.; HEYDARNOORI, A.; JAMSHIDI, P. Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. *IEEE Software*, IEEE Computer Society Press, Washington, DC, USA, v. 33, n. 3, p. 42—-52, maio 2016. ISSN 0740-7459. Citado na página 12.

BANIJAMALI, A. et al. Software Architecture Design of Cloud Platforms in Automotive Domain: An Online Survey. In: *2019 IEEE 12th Conference on Service-Oriented Computing and Applications (SOCA)*. [S.l.: s.n.], 2019. p. 168–175. Citado na página 10.

BANIJAMALI, A. et al. Software Architecture Design of Cloud Platforms in Automotive Domain: An Online Survey. In: *2019 IEEE 12th Conference on Service-Oriented Computing and Applications (SOCA)*. [S.l.: s.n.], 2019. p. 168–175. Citado na página 53.

BAO, Z. et al. Software Architecture for Responsible Artificial Intelligence Systems: Practice in the Digitization of Industrial Drawings. *Computer*, IEEE Computer Society, v. 56, n. 4, p. 38–49, 2023. ISSN 0018-9162. Citado na página 50.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. 2nd. ed. Boston, MA: Addison-Wesley, 2003. Citado 2 vezes nas páginas 7 and 36.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. [S.l.]: Addison-Wesley Professional, 2021. Citado 7 vezes nas páginas 4, 10, 11, 12, 13, 14, and 63.

BASS, L.; NORD, R. L. Understanding the Context of Architecture Evaluation Methods. In: *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*. Helsinki, Finland: IEEE, 2012. p. 277–281. Citado na página 18.

BEYER, B. et al. *Site Reliability Engineering: How Google Runs Production Systems*. [S.l.]: O'Reilly Media, 2016. Citado 3 vezes nas páginas 4, 13, and 14.

BOGNER, J. et al. Industry Practices and Challenges for the Evolvability Assurance of Microservices. *Empirical Software Engineering*, v. 26, n. 104, 2021. Citado na página 56.

BOMSTRöM, H. et al. Information Needs and Presentation in Agile Software Development. *Information and Software Technology*, v. 162, p. 107265, 2023. ISSN 0950-5849. Citado 2 vezes nas páginas 32 and 33.

COSTA, A.; TEIXEIRA, L. Testing Strategies for Smart Cities applications: A Systematic Mapping Study. In: *Proceedings of the III Brazilian Symposium on Systematic and Automated Software Testing*. New York, NY, USA: Association for Computing Machinery, 2018. (SAST '18), p. 20—-28. Citado na página 23.

COSTA, J. E. C.; SOARES, M. S. A Review on the Capacities of the ISO/IEC/IEEE 42020:2019 Standard for Architecture Elaboration of Software and Systems. In: *Proceedings of the XIX Brazilian Symposium on Information Systems*. New York, NY, USA: Association for Computing Machinery, 2023. (SBSI '23), p. 412—418. Disponível em: <https://doi.org/10.1145/3592813.3592932>. Citado na página 7.

COSTA, J. E. C.; SOARES, M. S. S. A Review on the Capacities of the ISO/IEC/IEEE 42020:2019 Standard for Architecture Elaboration of Software and Systems. In: *Proceedings of the XIX Brazilian Symposium on Information Systems*. New York, NY, USA: Association for Computing Machinery, 2023. p. 412—-418. ISBN 9798400707599. Citado 4 vezes nas páginas 8, 9, 16, and 52.

CRUZ, P. et al. Assessing Migration of a 20-Year-Old System to a Micro-Service Platform Using ATAM. In: *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. [S.l.: s.n.], 2019. p. 174–181. Citado na página 61.

CUNNINGHAM, W. The WyCash Portfolio Management System. *SIGPLAN OOPS Mess.*, Association for Computing Machinery, New York, NY, USA, v. 4, n. 2, p. 29—-30, dez. 1992. ISSN 1055-6400. Citado na página 30.

CâNDIDO, J.; ANICHE, M.; DEURSEN, A. van. Log-based Software Monitoring: A Systematic Mapping Study. *PeerJ Computer Science*, v. 7, p. 1–38, 2021. Citado na página 22.

DAS, D. et al. Technical Debt Resulting from Architectural Degradation and Code Smells: A Systematic Mapping Study. *SIGAPP Applied Computing Review*, Association for Computing Machinery, New York, NY, USA, v. 21, n. 4, p. 20—36, jan. 2022. ISSN 1559-6915. Citado na página 31.

DAYOUB, A. J. The Integration of Edge Computing Into IoT Application Using AdvantEDGE Platform, Case Study: Mobility. In: *2025 7th International Youth Conference on Radio Electronics, Electrical and Power Engineering (REEPE)*. [S.l.: s.n.], 2025. p. 1–5. Citado na página 54.

DUARTE, Y. et al. Exploratory Testing Strategies for Video Games: An Experience Report. In: *Proceedings of the 22nd Brazilian Symposium on Games and Digital Entertainment*. New York, NY, USA: Association for Computing Machinery, 2024. (SBGames '23), p. 46—-55. Citado na página 23.

EGGENDORFER, T.; ANDRESEN, K. Using Security Metrics to Improve Cyber-Resilience. In: *Proceedings of the IARIA Congress*. [S.l.: s.n.], 2024. p. 152–157. Citado na página 55.

EIGLER, T.; HUBER, F.; HAGEL, G. Tool-Based Software Engineering Education for Software Design Patterns and Software Architecture Patterns - A Systematic Literature Review. In: *Proceedings of the 5th European Conference on Software Engineering Education*. New York, NY, USA: Association for Computing Machinery, 2023. (ECSEE '23), p. 153—-161. Citado na página 21.

ERAY, E.; HAAS, C. T.; RAYSIDE, D. Interface Health and Workload between Stakeholders in Complex Capital Projects: Assessment, Visualization, and Interpretation Using SNA. *Journal of Management in Engineering*, v. 37, n. 3, p. 04021006, 2021. Citado na página 33.

ERDER, M.; PUREUR, P.; WOODS, E. *Continuous Architecture in Practice: Software Architecture in the Age of Agility and DevOps*. [S.l.]: Addison-Wesley Professional, 2021. Citado na página 11.

ERL, T. *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ: Prentice Hall PTR, 2005. Citado na página 12.

EZEOGU, A. O. Post-Quantum Cryptography for Healthcare: Future-Proofing Population Health Databases Againts Quantum Computing Threats. *Research Corridor Journal of Engineering Science*, v. 2, n. 1, p. 29–56, 2025. Citado 2 vezes nas páginas 57 and 58.

FRANÇA, J. M. S.; LIMA, J. de S.; SOARES, M. S. Development of an Electronic Health Record Application Using a Multiple View Service Oriented Architecture. In: HAMMOUDI, S. et al. (Ed.). *ICEIS 2017 - Proceedings of the 19th International Conference on Enterprise Information Systems, Volume 2, Porto, Portugal, April 26-29, 2017*. [S.l.]: SciTePress, 2017. p. 308–315. Citado na página 8.

GARLAN, D. Research Directions in Software Architecture. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 27, n. 2, p. 257–261, 1995. Citado 2 vezes nas páginas 11 and 12.

HAN, S. et al. Achieving High Spectrum Efficiency on High Speed Train for 5G New Radio and Beyond. *IEEE Wireless Communications*, v. 26, n. 5, p. 62–69, 2019. Citado na página 51.

HINRICHS, M.; PRIFTI, L. Visualizing Maintenance Data to Support Decisions on Strategic Maintenance Planning. In: *Proceedings of the 15th International Conference on PErvasive Technologies Related to Assistive Environments*. New York, NY, USA: Association for Computing Machinery, 2022. (PETRA '22), p. 473—479. Citado na página 23.

HOULIOTIS, K. et al. An Efficient Approach to Designing Mission-Critical Systems: Case Study: Defensive Aid Suite (DAS) Systems. In: *2017 International Conference on Military Technologies (ICMT)*. [S.l.: s.n.], 2017. p. 402–409. Citado na página 61.

HOYO-GABALDON, J.-A. del et al. Automatic Dataset Generation for Automated Program Repair of Bugs and Vulnerabilities Through SonarQube. *SoftwareX*, v. 26, p. 101664, 2024. ISSN 2352-7110. Citado na página 31.

HUMBLE, J.; FARLEY, D. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. [S.l.]: Addison-Wesley, 2010. Citado 3 vezes nas páginas 11, 12, and 13.

ISO/IEC/IEEE. *Systems and Software Engineering — Architecture Description*. 2011. International Standard. Citado na página 10.

ISO/IEC/IEEE. *Systems and Software Engineering — System Life Cycle Processes*. [S.l.]: ISO, 2015. Citado 2 vezes nas páginas 14 and 16.

ISO/IEC/IEEE. *Enterprise, Systems and Software — Architecture Processes*. 2019. Final Draft International Standard (FDIS). Citado 7 vezes nas páginas 4, 10, 11, 12, 13, 14, and 16.

JOSHI, H. Secure Software Architecture for Enterprise Generative Artificial Intelligence. In: *Proceedings of the 2024 IEEE 19th International Conference on Computer Science and Information Technologies (CSIT)*. [S.l.]: Institute of Electrical and Electronics Engineers, 2024. (CSIT '24), p. 1–5. Citado 2 vezes nas páginas 50 and 51.

KADRI, S.; AOUAG, S.; HEDJAZI, D. MS-QuAAF: A Generic Evaluation Framework for Monitoring Software Architecture Quality. *Information and Software Technology*, v. 140, p. 106713, 2021. ISSN 0950-5849. Citado na página 26.

KALOGIANNIDIS, S. Impact of Effective Business Communication on Employee Performance. *European Journal of Business and Management Research*, v. 5, n. 6, Dec 2020. Citado 2 vezes nas páginas 18 and 23.

KAUR, K.; KHURANA, M.; MANISHA. Impact of Agile Scrum Methodology on Time to Market and Code Quality – A Case Study. In: *2021 3rd International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)*. [S.l.: s.n.], 2021. p. 1673–1678. Citado 2 vezes nas páginas 7 and 9.

KAZMAN, R. et al. SAAM: A Method for Analyzing the Properties of Software Architectures. In: IEEE. *Proceedings of 16th International Conference on Software Engineering*. [S.l.], 1994. p. 81–90. Citado na página 14.

KAZMAN, R. et al. ATAM: Method for Architecture Evaluation. In: ACM. *Proceedings of the 22nd International Conference on Software Engineering*. [S.l.], 2000. p. 478–487. Citado 3 vezes nas páginas 8, 9, and 14.

KLINAKU, F.; HAKAMIAN, A.; BECKER, S. Architecture-Based Evaluation of Scaling Policies for Cloud Applications. In: *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. [S.l.: s.n.], 2021. p. 151–157. Citado na página 18.

KOMOLOV, S. et al. Towards Predicting Architectural Design Patterns: A Machine Learning Approach. *Computers*, MDPI, v. 11, n. 10, 2022. ISSN 2073-431X. Citado na página 51.

KOTHAPALLI, S. et al. DevOps and Software Architecture: Bridging the Gap between Development and Operations. *American Digits: Journal of Computing and Digital Technologies*, v. 2, n. 1, p. 51–64, 2024. Citado na página 11.

KOZIOLEK, H. Sustainability Evaluation of Software Architectures: A Systematic Review. In: *Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical Systems – ISARCS*. New York, NY, USA: Association for Computing Machinery, 2011. (QoSA-ISARCS '11), p. 3—12. Citado na página 10.

LAGO, P. Architecture Design Decision Maps for Software Sustainability. In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*. [S.l.: s.n.], 2019. p. 61–64. Citado na página 29.

LAGO, P. et al. The Sustainability Assessment Framework Toolkit: A Decade of Modeling Experience. *Software and Systems Modeling*, v. 24, n. 2, p. 361–383, 2024. ISSN 1619-1374. Citado 2 vezes nas páginas 26 and 63.

LENARDUZZI, V. et al. A Systematic Literature Review on Technical Debt Prioritization: Strategies, Processes, Factors and Tools. *Journal of Systems and Software*, v. 171, p. 110827, 2021. ISSN 0164-1212. Citado na página 30.

LIU, X. Research on the Application of Electromechanical Automation Control Software Architecture to Production Line Inspection Under Cloud Computing Platform. In: *2024 IEEE 7th International Conference on Information Systems and Computer Aided Education (ICISCAE)*. [S.l.: s.n.], 2024. p. 1212–1216. Citado na página 52.

LÓPEZ, L. et al. QaSD: A Quality-Aware Strategic Dashboard for Supporting Decision Makers in Agile Software Development. *Science of Computer Programming*, v. 202, p. 102568, 2021. ISSN 0167-6423. Citado na página 31.

MAGABALEH, A. A. et al. Systematic Review of Software Engineering Uses of Multi-Criteria Decision-Making Methods: Trends, Bibliographic Analysis, Challenges, Recommendations, and Future Directions. *Applied Software Computing*, v. 163, p. 111859, 2024. ISSN 1568-4946. Citado na página 31.

MARTIN, J. N. Overview of an Emerging Standard on Architecture Processes — ISO/IEC/IEEE 42020. In: *2018 Annual IEEE International Systems Conference (SysCon)*. Vancouver, BC, Canada: IEEE, 2018. p. 1–8. Citado 3 vezes nas páginas 8, 9, and 16.

MARTIN, R. C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. [S.l.]: Prentice Hall Press, 2017. Citado na página 11.

MARTÍNEZ-FERNÁNDEZ, S. et al. Software Engineering for AI-Based Systems: A Survey. *ACM Transactions on Software Engineering and Methodology*, Association for Computing Machinery, New York, NY, USA, v. 31, n. 2, abr. 2022. ISSN 1049-331X. Citado na página 63.

MARY, S.; DAVID, G. Software Architecture: Perspectives on an Emerging Discipline. *Prentice-Hall*, 1996. Citado 2 vezes nas páginas 10 and 12.

MARZOONI, H. H.; MOTAMENI, H.; EBRAHIMNEJAD, A. Architecture Style Selection using Statistics of Quality Attributes to Reduce Production Costs. *International Arab Journal Of Information Technology*, v. 18, n. 4, p. 513–522, JUL 2021. Citado na página 21.

MELO, G.; SOARES, M. A Process to Compare ATAM and Chapter 9 of ISO/IEC/IEEE 42020:2019. In: . [S.l.: s.n.], 2025. p. 37–46. Citado 3 vezes nas páginas 8, 9, and 20.

MIRAKHORLI, M.; CLELAND-HUANG, J. Detecting, Tracing, and Monitoring Architectural Tactics in Code. *IEEE Transactions on Software Engineering*, v. 42, n. 3, p. 205–220, 2016. Citado na página 30.

MISON, A.; DAVIES, G.; EDEN, P. Cyber Resilience, Dependability and Security. In: Academic Conferences International Limited. *International Conference on Cyber Warfare and Security*. [S.l.], 2024. p. 177–184. Citado na página 55.

MÜLLER, H. Software Architecture Evaluation Methods and Tools: Analyzing Methods and Tools for Evaluating Software Architectures to Ensure Adherence to Quality Attributes and Design Principles. *Distributed Learning and Broad Applications in Scientific Research*, v. 6, p. 1–14, 2020. Citado na página 14.

MULLER, M. et al. Learning from Team and Group Diversity: Nurturing and Benefiting from our Heterogeneity. In: *Companion Publication of the 2019 Conference on Computer Supported Cooperative Work and Social Computing*. New York, NY, USA: Association for Computing Machinery, 2019. (CSCW '19 Companion), p. 498–505. Citado na página 23.

NAVEEN, S.; KOUNTE, M. R. Key Technologies and Challenges in IoT Edge Computing. In: *2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*. [S.l.: s.n.], 2019. p. 61–65. Citado na página 54.

NEWMAN, S. *Building Microservices: Designing Fine-Grained Systems*. [S.l.]: O'Reilly Media, 2015. ISBN 1491950358. Citado na página 12.

NOH, G.; HUI, B.; KIM, I. High Speed Train Communications in 5G: Design Elements to Mitigate the Impact of Very High Mobility. *IEEE Wireless Communications*, v. 27, n. 6, p. 98–106, 2020. Citado na página 52.

PARCHMAN, Z. W. et al. Adding Fault Tolerance to NPB Benchmarks Using ULFM. In: *Proceedings of the ACM Workshop on Fault-Tolerance for HPC at Extreme Scale*. New York, NY, USA: Association for Computing Machinery, 2016. (FTXS '16), p. 27—-34. Citado na página 23.

PARNAS, D. L. *Use of Abstract Interfaces in the Development of Software for Embedded Computer Systems*. [S.l.], 1977. Citado na página 12.

PEITEK, N. et al. Correlates of Programmer Efficacy and their Link to Experience: A Combined EEG and Eye-Tracking Study. In: *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2022. (ESEC/FSE 2022), p. 120—-131. ISBN 9781450394130. Citado na página 33.

PETERSEN, K. et al. Revisiting the Construct and Assessment of Industrial Relevance in Software Engineering Research. In: *2024 IEEE/ACM International Workshop on Methodological Issues with Empirical Studies in Software Engineering (WSESE)*. [S.l.: s.n.], 2024. p. 17–20. Citado na página 63.

POWER, K.; WIRFS-BROCK, R. An Exploratory Study of Naturalistic Decision Making in Complex Software Architecture Environments. In: Bures, T and Duchien, L and Inverardi, P (Ed.). *Software Architecture, ECSA 2019*. [S.l.], 2019. (Lecture Notes in Computer Science, v. 11681), p. 55–70. 13th European Conference on Software Architecture Engineering (ECSA), Paris, FRANCE, SEP 09-13, 2019. Citado na página 21.

RAJA, A. R. et al. Software Re-Engineering using Cloud Computing Platform. In: *2023 5th International Conference on Smart Systems and Inventive Technology (ICSSIT)*. [S.l.: s.n.], 2023. p. 812–816. Citado na página 53.

RAMACHANDRAN, R. Empowering Software Architects with Artificial Intelligence: Analyzing GitHub Copilot's Role in Modern Architecture Design. In: *Proceedings of the 2025 7th International Conference on Software Engineering and Computer Science (CSECS)*. Taicang, China: Institute of Electrical and Electronics Engineers, 2025. (CSECS '25), p. 1–9. Citado 2 vezes nas páginas 49 and 50.

RICHARDS, M. *Microservices vs. Service-Oriented Architecture*. [S.l.]: O'Reilly Media Sebastopol, 2015. Citado na página 11.

RICHARDS, M.; FORD, N. *Fundamentals of Software Architecture: an Engineering Approach*. [S.l.]: O'Reilly Media, 2020. Citado 2 vezes nas páginas 11 and 13.

SALDANA, Y. P. et al. Evaluation of the Modifiability of an Evolution System Using the ATAM Method. *International Journal of Science and Research (IJSR)*, v. 8, n. 2, p. 1772–1779, February 2019. Citado na página 15.

SANTOS, V. M.; MISRA, S.; SOARES, M. S. Architecture Conceptualization for Health Information Systems Using ISO/IEC/IEEE 42020. In: GERVASI, O. et al. (Ed.). *Computational Science and Its Applications - ICCSA 2020 - 20th International Conference, Cagliari, Italy, July 1-4, 2020, Proceedings, Part VI*. [S.l.]: Springer, 2020. (Lecture Notes in Computer Science, v. 12254), p. 398–411. Citado na página 8.

SEDAGHATBAF, A.; AZGOMI, M. A. SQME: A Framework for Fodeling and Evaluation of Software Architecture Quality Attributes. *Software & Systems Modeling*, v. 18, n. 4, p. 2609–2632, 2019. ISSN 1619-1374. Citado na página 26.

SEPúLVEDA, S. et al. Systematic Review on Requirements Engineering in Quantum Computing: Insights and Future Directions. *Electronics*, v. 13, n. 15, 2024. ISSN 2079-9292. Citado na página 57.

SHARKOV, G. A System-of-Systems Approach to Cyber Security and Resilience. *Information & Security*, v. 37, p. 69–94, 2017. Citado 2 vezes nas páginas 54 and 55.

SILVA, S. et al. Quality Metrics in Software Architecture. In: *2023 IEEE 20th International Conference on Software Architecture (ICSA)*. [S.l.: s.n.], 2023. p. 58–69. Citado 4 vezes nas páginas 7, 9, 10, and 63.

TADI, S. Architecting Resilient Cloud-Native APIs: Autonomous Fault Recovery in Event-Driven Microservices Ecosystems. *Journal of Scientific and Engineering Research*, v. 9, n. 3, p. 293–305, 2022. Citado na página 23.

ULUDAG, O.; MATTHES, F. Large-Scale Agile Development Patterns for Enterprise and Solution Architects. In: *Proceedings of the European Conference on Pattern Languages of Programs 2020*. New York, NY, USA: Association for Computing Machinery, 2020. (EuroPLoP '20). Citado na página 21.

van Vliet, H.; TANG, A. Decision Making in Software Architecture. *Journal of Systems and Software*, v. 117, p. 638–644, 2016. ISSN 0164-1212. Citado na página 7.

VERA, T.; PEROVICH, D.; OCHOA, S. F. An Instrument to Define the Product Scope at Preselling Time. In: *2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. [S.l.: s.n.], 2021. p. 604–608. Citado na página 30.

VERDECCHIA, R. et al. Building and Evaluating a Theory of Architectural Technical Debt in Software-Intensive Systems. *Journal of Systems and Software*, Elsevier, v. 176, p. 110925, 2021. Citado na página 11.

VERDECCHIA, R. et al. Estimating Energy Impact of Software Releases and Deployment Strategies: The KPMG Case Study. In: *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. [S.l.: s.n.], 2017. p. 257–266. Citado na página 29.

WAN, Z. et al. Software Architecture in Practice: Challenges and Opportunities. In: *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, NY, USA: Association for Computing

Machinery, 2023. (ESEC/FSE 2023), p. 1457—1469. ISBN 9798400703270. Citado na página 30.

ZAMPETTI, F. et al. Continuous Integration and Delivery Practices for Cyber-Physical Systems: An Interview-Based Study. *ACM Transactions on Software Engineering and Methodology*, Association for Computing Machinery, New York, NY, USA, v. 32, n. 3, apr 2023. Citado na página 22.

ZAROUR, M.; ALENEZI, M.; ALSARAYRAH, K. Software Security Specifications and Design: How Software Engineers and Practitioners Are Mixing Things Up. In: *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2020. (EASE '20), p. 451—456. Citado na página 22.