Agile Software Architecture: Perceptions on Quality and Architectural Technical Debt Management

Manoel Valerio Da Silveira Neto
Programa de Pós-graduação em Informática - Pontifícia Universidade Católica do Paraná (PUCPR)
Curitiba, Brasil
manoel.silveira@pucpr.edu.br

Andreia Malucelli
Programa de Pós-graduação em Informática - Pontifícia Universidade Católica do Paraná (PUCPR)
Curitiba, Brasil
andreia.malucelli@pucpr.br

Sheila Reinehr
Programa de Pós-graduação em Informática - Pontifícia Universidade Católica do Paraná (PUCPR)
Curitiba, Brasil
sheila.reinehr@pucpr.br

ABSTRACT

Software architecture plays a central role in sustaining quality in systems developed using agile methodologies, particularly in large-scale environments. However, the pressure for rapid deliveries and the decentralization of decision-making can compromise attention to quality attributes and exacerbate the accumulation of architectural technical debt. This paper presents the results of an exploratory empirical study involving 14 software architects from a Brazilian financial institution. Using a case study approach with structured interviews, the study investigated professionals' perceptions regarding the most relevant quality attributes during the architectural design phase and the management of technical debt after the product is released to production. The mixed-methods analysis revealed that functional suitability, security, and reliability are the most prioritized attributes during the design phase, while efficiency and maintainability become more prominent in the post-production phase. Qualitative responses indicated heterogeneous practices, a lack of formalization process for managing technical debt, and weaknesses in the application of architectural tactics. As a contribution, the study offers empirical insights to support future initiatives aimed at systematizing architectural decision-making in agile environments, with attention to quality attributes and organizational factors. This study also adheres to open science principles, making all research materials, anonymized data, and analysis scripts publicly available to foster transparency, reproducibility, and reuse.

KEYWORDS

Agile Software Architecture, Architectural Technical Debt, Architectural Tactics, Attribute- Driven Design, Software Quality Attributes.

1 Introduction

Since the publication of the Agile Manifesto, a significant transformation has taken place in how organizations develop software, emphasizing continuous value delivery and adaptability to change [9]. While agile methods foster greater flexibility and collaboration[24], their focus on rapid delivery can, in some cases, undermine rigorous software engineering practices, particularly with regard to software architecture planning, which often requires upfront strategic thinking that may conflict with agile's preference for minimal initial design.

In agile contexts, approaches such as Design Thinking and Lean Inception are commonly used to guide the design of digital products. However, these approaches tend to emphasize business vision over a systematic and intentional architectural process [2]. As a result, traditional practices like Big Design Up Front (BDUF), characterized by extensive documentation and detailed specifications before implementation, are often replaced by more flexible approaches aligned with the notion of emergent design.

As a response to the lack of formal architectural planning in agile environments, the concept of Minimum Viable Architecture (MVA) has emerged. MVA advocates for a minimal architectural foundation that supports essential product functionality during the early stages of development. However, adopting MVA can lead to the accumulation of architectural technical debt, especially when quality attributes are overlooked, necessitating future architectural refactoring [19].

To address these challenges, some authors [18] recommend adopting intentional architecture. This approach involves making proactive architectural decisions to guide system evolution while maintaining alignment with strategic goals and quality attributes from the outset. Strategies such as Architecture Haiku [33] and Architectural Hoisting [21] have been proposed to incrementally integrate quality attributes into the architecture as part of the agile development cycle.

The role of software architecture in enabling quality attributes such as security, modifiability, and usability is widely acknowledged [8]. However, applying intentional architecture in large-scale contexts, such as programs involving multiple agile teams, poses additional challenges, including coordination among teams and managing distributed architectural decisions [22, 28, 58].

Although the literature discusses the management of architectural technical debt and strategies for incorporating quality attributes, these practices remain infrequently applied in the industry. Bridging this gap requires empirical investigations involving software architects working in Large-Scale Agile Development (LSAD) settings to explore their perceptions, current practices, and the factors influencing quality attribute prioritization and architectural debt management.

This study, therefore, seeks to investigate software architects' perceptions of the most critical quality attributes during the early design phase of products developed in LSAD contexts. It also aims to identify which attributes are considered essential for managing architectural technical debt after the product is released to production. Findings from this study can contribute to improving architectural decision-making in LSAD settings by providing empirical insights into how quality is prioritized in practice.

2 Theoretical Background

2.1 Software Quality

Software quality is a core concern in software engineering, as it directly affects user satisfaction, maintainability, and the long-term sustainability of software solutions. According to ISO/IEC 25010:2023, quality refers to the "totality of characteristics of a software product that confer its ability to satisfy stated and implied needs" [30]. While stated needs are documented as requirements, implied needs are often essential for delivering a satisfactory user experience.

In software architecture, these characteristics influence structural decisions and critical system attributes. In [8] highlight that architecture directly impacts quality attributes such as availability, modifiability, and usability, which can be supported through architectural tactics like redundancy, partitioning, and load balancing.

Authors such as [44] and Richards and Ford [25] emphasize that in modern architectures, especially those based on microservices and distributed systems, attributes such as observability, monitoring, and operational efficiency become increasingly important. These attributes allow developers to track system behavior in production, enabling rapid failure recovery and continuous improvement.

Applying these quality attributes effectively throughout the software lifecycle ensures that systems not only meet functional requirements but also maintain performance, security, and reliability over time. This is particularly relevant in large-scale agile projects, where pressure for rapid delivery may compromise long-term architectural quality and increase the risk of accumulating technical debt.

2.2 Software Architecture, Intentional Architecture, Architectural Tactics, ADD, and Large-Scale Agile Development

Software Architecture (SA) defines the high-level structure of complex systems, bridging the gap between requirements and implementation [1]. It plays a key role in addressing cross-cutting concerns such as scalability, robustness, and security [26, 34]. In agile environments, the traditional Big Design Up Front (BDUF) approach [16] has been largely replaced by emergent design and iterative architectural refactoring (ARf), as highlighted by empirical studies [4, 62].

Integrating quality attributes early in the project lifecycle is critical for mitigating the accumulation of technical debt (TD) [12]. In large-scale environments, the absence of intentional architecture can jeopardize attributes such as security, performance, and maintainability. The intentional architecture (IA) approach addresses this by promoting deliberate architectural decisions that remain aligned with business goals and quality attributes from the outset [27].

To operationalize these decisions, architectural tactics provide concrete design strategies to address specific quality attributes [6, 8]. Examples include load balancing, component isolation, and data replication, which contribute to enhancing system reliability, performance, and maintainability. Complementarily, the Attribute-Driven Design (ADD) method [60, 61] supports architectural modeling by prioritizing quality attributes and iteratively guiding design decisions based on domain knowledge and system constraints.

The Agile Manifesto encourages emergent architecture driven by self-organizing teams [9], but this approach may fall short in complex, large-scale environments [49]. Large-Scale Agile Development (LSAD) refers to the application of agile methods within large, structured organizations [17, 58], where architectural governance, coordination among teams, and scaling practices are critical success factors. Recent research highlights the unique challenges LSAD faces in domains such as finance, where architectural practices must balance agility with stringent regulatory and operational requirements [29, 46].

In this study, these concepts form the theoretical foundation for examining how quality attributes are prioritized and how architectural technical debt is managed in large-scale agile financial environments.

2.3 Architectural Technical Debt

Technical debt arises when best practices are compromised, often due to short-term priorities [14]. Architectural technical debt (ATD), in particular, compromises long-term system qualities and is frequently associated with deadline pressure [35]. Studies show that ATD leads to high maintenance costs and system rewrites [20, 37, 40]. Several models have been proposed for identifying, measuring, and prioritizing ATD [11, 15]. One such model, PriorTD [15], supports informed decision-making based on the architectural impact of technical debt.

In agile contexts, constant delivery pressure and limited architectural awareness exacerbate the accumulation of technical debt [52]. Architectural refactoring is often required to recover key attributes such as security and reliability [30]. Failure to address ATD undermines scalability and system sustainability[1].

Moreover, while Continuous Software Engineering (CSE) practices, such as Continuous Integration (CI), Continuous Delivery (CD), and DevOps are broadly adopted, architectural technical debt remains underrepresented in these practices. In [13] found that most studies do not comprehensively address ATD across the full CSE lifecycle.

Recently, [3] proposed a manifesto that reframes ATD as a sociotechnical and strategic concern. The manifesto calls for traceable decisions, transparency, and alignment with business goals, while critiquing current tools and advocating for stronger integration between research, practice, and governance.

3 Related Work

Several studies address quality attributes in software architecture from different perspectives. Lichtenthaler [36][2] proposed a model focused on cloud applications, synthesizing characteristics extracted from books and scientific literature, and validating it with 42 professionals. Although the results generally support the model, they also highlight areas for improvement, leading to an updated version grounded in empirical and theoretical evidence.

[1] https://martinfowler.com/articles/bottlenecks-of-scaleups/01-tech-debt.html
[2] Available at: https://github.com/r0light/cna-quality-model

Study [39], also discussed by [41], analyzed reference architectures and cloud service components, identifying five recurring quality attributes: reliability, scalability, performance, availability, and security, the latter being strongly associated with various architectural tactics.

In [42], scalability is treated as a key concern for microservices-based architectures. The authors identified nine architectural frameworks that address scalability from different perspectives, along with five reusable design decisions that help achieve it. The systematic review [47] reported 44 academic and 80 industrial architectural patterns, many of which are related to DevOps and IoT. The most commonly cited quality attributes in these studies were scalability, flexibility, testability, performance, and elasticity.

Energy efficiency is emphasized in [50, 51] as an emerging concern in the ICT domain. These studies advocate for its inclusion in software quality models. In a survey involving 16 professionals, [48] found that maintainability and performance were the most valued attributes, while portability was the least mentioned. Notably, none of the participants reported using modeling techniques to assess quality early in the project lifecycle, instead relying on commercial tools during implementation.

According to [41], there is a lack of empirical evidence on the use of architectural tactics in industry. Although security is the most frequently addressed attribute, 75% of the reviewed studies do not clearly indicate the sources or techniques used to identify the tactics. Furthermore, many of the reported tactics diverge from the original definitions presented in [8], pointing to a need for formalization. This reinforces the urgency of conducting further empirical research on architectural tactics in industrial contexts.

In [56] performed a mapping study on continuous architecture evaluation within the context of Continuous Software Engineering (CSE), including scenario-based approaches, metrics, simulations, and modeling techniques. Despite the methodological diversity, such approaches are rarely adopted in practice, especially in agile environments, contributing to the lack of systematic mechanisms for evaluating architectural quality.

In [38], operational and development-related quality attributes were analyzed in internet banking architectures using the SAAM method. The study demonstrated the method's effectiveness in quality assessment through measurable metrics.

The survey conducted in [31] examined the influence of quality requirements on architectural decisions. It revealed several gaps, including insufficient documentation of patterns, weak quality management practices, and limited alignment between development methods and quality attributes. The authors also emphasized the inherently subjective and qualitative nature of the concept of software quality.

In [5], the authors criticized methods such as ATAM for requiring co-location of stakeholders, which can be costly and impractical in distributed development contexts. As an alternative, they proposed a groupware-based approach that enables distributed architectural evaluations—both synchronous and asynchronous—better suited for agile and dispersed teams.

In summary, the literature has broadened the understanding of software quality attributes and architectural tactics, particularly in cloud and microservices contexts. However, notable gaps remain in the systematization of tactics used in industry, as well as in the integration of architecture with domains such as DevOps, IoT, and energy sustainability. Furthermore, attributes such as portability and energy efficiency remain underexplored. In response, the present study adopts ISO/IEC 25010:2023 [30] as a conceptual foundation, prioritizing standardized quality characteristics to support sustainable architectural practices in agile and scalable environments.

4 Research Method

This study adopts a study case [63] as its methodological approach, which is appropriate for understanding perceptions within real-world agile development contexts, as highlighted by [59].

The development of the data collection instrument was guided by the Goal–Question–Metric (GQM) model, proposed by [7], a widely adopted framework in software engineering for structuring goal-oriented measurements. The GQM model consists of three main components: Goal – what one aims to understand or improve; Question – specific questions that translate the goal into observable terms; and Metric – indicators used to evaluate responses to those questions.

In this study, the defined goal was: "Identify the most relevant software quality characteristics for software architecture in order to understand their importance from the perspective of software architects working on agile projects, both during the initial design phase and in the management of architectural technical debt". Based on this goal, a structured interview was developed.

To analyze the open-ended responses, we employed the Qualitative Analysis methodology proposed by [54], which aims to extract categories and emerging patterns through an iterative coding process. The analysis was conducted in two cycles: First cycle – applying open, descriptive, and provisional coding based on prior concepts and emergent categories; Second cycle – grouping and abstracting codes into higher-level explanatory categories to identify recurring patterns in reported practices.

This combination of GQM and qualitative coding (inspired by Grounded Theory) enabled the integration of quantitative and qualitative data analyses, providing a comprehensive and in-depth understanding of architects' practices and perceptions regarding architectural quality and technical debt in agile contexts.

4.1 Research Context

The software architects who participated in this study work directly within agile teams at a financial institution comprising 32 development teams, 16 of which follow agile methodologies and are structured according to the Spotify Model. Each agile team (squad) typically consists of around eight members, including a Product Owner (PO), Team Leader (TL)[3]., a Software Architect, and members of the DevTeam[4].

In this context, the software architect plays a strategic role in defining solution design, orchestrating interactions with infrastructure, and ensuring operational excellence through monitoring, observability, and cost optimization practices. These responsibilities are amplified by the adoption of a hybrid and multi-cloud

[3]Role like Scrum Master ou Agile Coach

[4]Squads usually include backend, frontend, and mobile developers, as well as quality assurance analysts (testers).

infrastructure, supported by microservices and event-driven architecture. Architects are also co-responsible for assessing scalability and managing infrastructure costs, given the intensive use of cloud computing.

Each architect works with one or more agile teams and actively participates in agile ceremonies and refinement sessions alongside the PO and the team. During these sessions, new features and bug fixes are discussed, prioritized by the PO and Product Manager, and broken down into user stories, epics, or features. This process is aligned with the Zipper Model described by [10], which emphasizes integrating architectural decisions into strategic planning and continuous delivery cycles.

Strategic initiatives are prioritized through the use of Objectives and Key Results (OKRs)[5] during quarterly planning meetings known as Quarterly Business Reviews (QBR)[6]. Within this planning cycle, architects are also responsible for identifying architectural technical debt and collaborating with a software engineering committee to evolve the intentional architecture.

As discussed by [45], the architects in this study operate across three interconnected domains: (i) system architecture, (ii) organizational development structure, and (iii) production infrastructure. This multifaceted role highlights the need for the continuous involvement of architects in decision-making processes that affect technical sustainability, strategic alignment, and architectural evolution in agile environments.

One of the researchers was employed as a software architect at the institution during the time of the study and received formal authorization from the organization to conduct the research. However, despite being an employee, this researcher did not participate as a respondent in the study and had no influence on the answers provided by the participants.

4.2 Research Planning and Execution

The study was conducted using a structured interview with 14 software architects working at a Brazilian financial institution. The support instrument included eight questions (Q1–Q8), combining closed and open-ended questions to capture both quantitative data and qualitative perceptions: (Q1) How many months of experience do you have in software/solution architecture? (Q2) How long have you worked for this company (in months)? (Q3) How long have you worked with agile methodologies (in months)? (Q4) How do you perceive the importance of quality attributes in software architecture? (Q5) Describe how quality attributes are addressed during architectural design. (Q6) Which quality attribute do you consider most important for managing architectural technical debt? (Q7) Describe how architectural technical debt is managed. (Q8) Describe how architecture evaluations are performed.

As a preparatory activity for the interview, participants were provided with an introduction about quality attribute definitions of the ISO/IEC 25010:2023 [30], ensuring a shared conceptual understanding.

Prior to the main data collection, a pilot study was conducted with two software architects from the same company participating in the research. The pilot was carried out remotely over a four-week period in the first semester of 2024, using an asynchronous online form. Its purpose was to validate the clarity, terminology, and sequence of the structured interview questions, as well as to estimate the average completion time. Based on the feedback received, minor adjustments were made to the wording of certain questions and the inclusion of illustrative examples to ensure a consistent understanding of the ISO/IEC 25010:2023 quality attributes. The results obtained in the pilot study were used exclusively to refine the data collection instrument and were not included in the final analysis. For the main study, the researcher conducted a virtual meeting with all 14 participating software architects to present and explain the structured interview form, clarify definitions, and ensure consistent interpretation of the questions. After this session, the participants completed the form remotely through an asynchronous online submission.

Only professionals who signed an Informed Consent Form participated. In compliance with Brazil's General Data Protection Law (LGPD), no sensitive personal data were collected, exempting the study from formal ethics committee approval. Nevertheless, anonymity and the academic use of information were strictly maintained.

4.3 Results Analysis

A mixed-methods approach was adopted to analyze the responses, combining quantitative and qualitative techniques.

Quantitative Analysis: Responses to Q1–Q3 were analyzed using descriptive statistics (mean, minimum, and maximum values) to characterize the participants' profiles. For Q4, the Analytic Hierarchy Process (AHP) [53] method was used to rank and prioritize the importance of the quality attributes. Q6 was examined using a weighted scoring model.

Qualitative Analysis: Open-ended responses to Q5, Q7, and Q8 were analyzed following the coding methodology described by [55], structured in two cycles. In the first cycle, open, descriptive, and provisional coding techniques were employed. Open coding helped deconstruct the text into smaller analytical units, while provisional codes were guided by a predefined list derived from literature, the researchers' prior experience, and insights from the pilot study. According to [55], this step aims to capture the essence of participants' statements through representative labels.

In the second cycle, pattern coding was applied to group and abstract first-cycle codes into broader explanatory categories. This step followed Grounded Theory principles to reveal recurring patterns in the participants' architectural practices and perceptions of quality and technical debt management. The use of Computer-Assisted Qualitative Data Analysis Software (CAQDAS) tools[7] facilitated the systematic organization, segmentation, and analysis of the qualitative data.

This combined approach allowed the study to capture general trends (via quantitative analysis) and nuanced interpretations (via qualitative analysis), providing a rich and comprehensive view of

[5] OKRs were popularized by Andrew Grove at Intel and are described in the book High Output Management. See: https://rework.withgoogle.com/guides/set-goals-with-okrs/steps/introduction/.

[6] QBRs are periodic meetings with stakeholders to assess the business value and outcomes of products or services.

[7] Computer-Assisted Qualitative Data Analysis Software (e.g., NVivo, Atlas.ti) supports coding, analysis, and interpretation of textual data.

the role of quality attributes in software architecture within large-scale agile environments.

5 Results

The main findings of the study are presented below.

5.1 Respondents' Experience Profile (Q1-Q3)

This subsection presents an overview of the participants' professional experience, including their tenure in software architecture roles, duration of employment at the organization, and background with agile methodologies. The aim is to contextualize the respondents' perspectives by mapping their expertise and familiarity with the organizational and methodological environments in which they operate. The analysis of questions Q1, Q2, and Q3 was conducted using IBM SPSS[8] and a Python script, both of which are available in Section 8. Table 1 presents descriptive statistics (in years) for each of the three dimensions assessed.

Table 1: Summary of respondent experience

Question	Mean	Min	Max	Median
Q1 – Experience with Architecture	4.00	0.08	13.83	2.38
Q2 – Time at Current Company	2.73	0.33	12.00	1.92
Q3 – Experience with Agile	6.33	2.50	13.00	5.75

The results show that participants have, on average, 4 years of experience in software architecture, with values ranging from less than one year to over 13 years. The median value (2.38 years) indicates a moderately experienced group. Regarding tenure at the current company (Q2), the average is 2.73 years, reflecting a relatively high turnover rate typical in agile environments. As for agile experience (Q3), respondents report an average of over 6 years, suggesting strong familiarity with agile practices and frameworks.

5.2 Perceived Importance of Quality Attributes (Q4)

Table 2 and Figure 1 present the AHP-based ranking of software quality attributes according to respondents' perceptions within agile development contexts. This analysis applies the AHP to quantify and prioritize the relative importance of quality characteristics, transforming subjective judgments into an objective and comparable ranking.

The attribute **Security** emerged as the most valued, receiving the highest AHP weight (13.89%), which reflects a strong concern with preserving system integrity and mitigating risks such as operational failures, data breaches, and cyberattacks.

Next, both **Functional Suitability** and **Reliability** shared equal weights (13.10%), indicating a balanced emphasis on ensuring that the system fulfills its intended functionalities and operates consistently over time.

Attributes such as **Maintainability**, **Performance Efficiency**, and **Safety** also received notable weights (around 12%), reinforcing the relevance of operational effectiveness and long-term evolvability in agile environments.

[8]Available at: https://www.ibm.com/br-pt/analytics/spss-statistics-software

Conversely, **Flexibility** received the lowest weight (5.16%), which may indicate a lower perceived importance compared to attributes directly associated with security and robustness. This result could be related to the inherent adaptability of agile practices, which may compensate for architectural inflexibility through iterative development and continuous feedback.

Table 2: Q4 – AHP Ranking of Software Quality Attributes

Attribute	AHP Weight	Total Score
Security	0.1389	140
Functional Suitability	0.1310	132
Reliability	0.1310	132
Maintainability	0.1210	122
Performance Efficiency	0.1210	122
Safety	0.1210	122
Interaction Capability	0.1071	108
Compatibility	0.0774	78
Flexibility	0.0516	52

5.3 Addressing Quality Attributes During Architectural Design (Q5)

The responses to question Q5 were analyzed qualitatively using Atlas.ti software[9], following the coding methodology proposed by [55]. The analysis revealed significant heterogeneity in the approaches adopted by software architects for addressing quality characteristics during the architectural design phase. The results were grouped into five distinct patterns, as shown in Table 3. Interviewees are anonymized and identified as RESP1, RESP2, ..., RESP14 throughout the text.

These patterns range from decision-making based solely on personal experience, with no methodological foundation, to the systematic application of mature architectural practices. This diversity underscores the importance of having structured evaluation models that can guide architectural decisions from the early stages of the software development lifecycle, particularly in agile contexts.

5.4 Key Quality Attributes for Managing Architectural Technical Debt (Q6)

The objective of Q6 was to identify which quality characteristics from the ISO/IEC 25010:2023 standard are perceived by professionals as most relevant for the management of architectural technical debt (ATD). ATD refers to the accumulation of design decisions that, while expedient in the short term, compromise the system's maintainability, performance, security, and evolvability over time. Understanding which characteristics are perceived to most directly influence this form of debt is essential to support effective monitoring and mitigation strategies.

Respondents were asked to rank the quality characteristics in order of importance for architectural technical debt management. Each characteristic received a rank from 1 (most important) to 9 (least important).

To synthesize the data, a weighted scoring method was applied: each ranking position was assigned a score inversely proportional

[9]https://atlasti.com

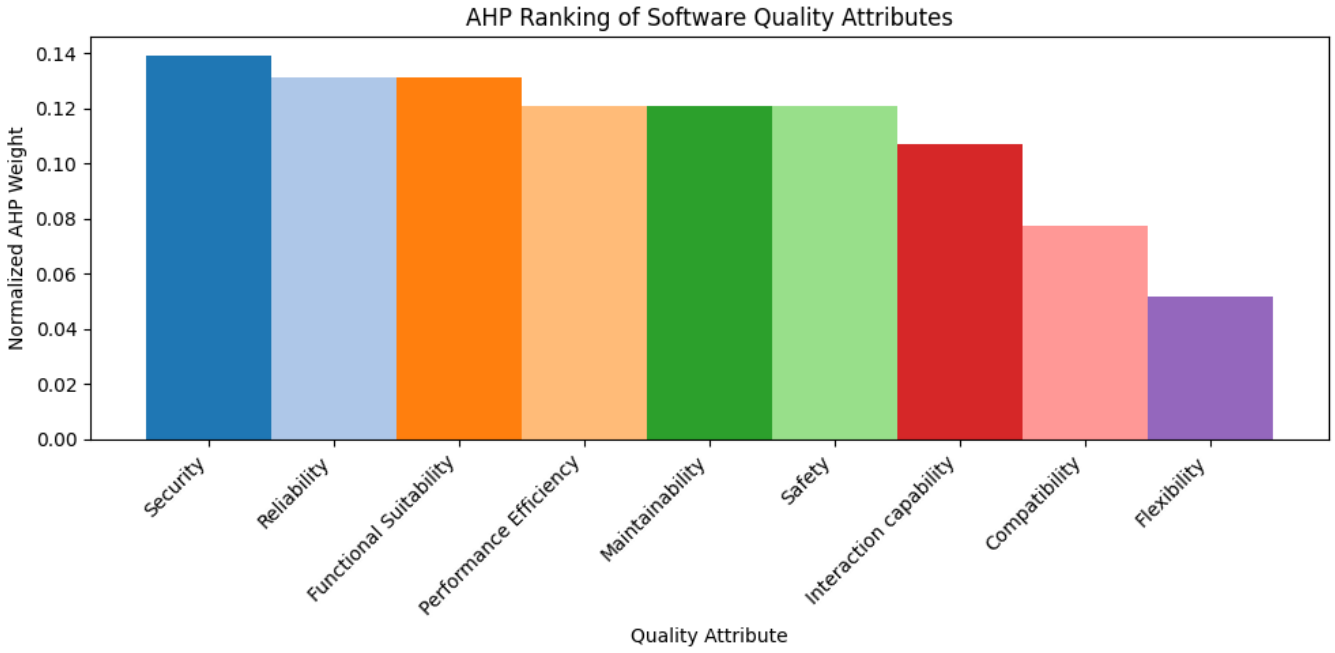

Figure 1: Q4 – Distribution of AHP Weights for Quality Attributes

Table 3: Q5 – Patterns in the Treatment of Quality Characteristics and Corresponding Respondents

Pattern	Description	Respondents
Empirical / Subjective	Decisions are made based on individual experience, without formal guidelines, explicit criteria, or established processes.	RESP14, RESP3, RESP4
Reactive / Post-Failure	Quality attributes are only addressed after the occurrence of failures, production incidents, or external pressure.	RESP10, RESP14, RESP3, RESP2
Initial Functionalism	Primary focus is on delivering minimum viable functionalities (MVP), with the treatment of other quality attributes deferred.	RESP14, RESP5, RESP6, RESP8, RESP2
Pragmatic / Minimal Engineering	Basic practices are adopted (e.g., unit testing, containerization, exception handling), but without systematic processes or formal architectural definitions.	RESP12, RESP6, RESP9
Structured / Systematic	Consistent use of architectural best practices such as design patterns (e.g., DDD), test automation, and validation through metrics and tools.	RESP11, RESP13, RESP5, RESP7, RESP9, RESP1

to its position (e.g., priority 1 = 9 points, priority 2 = 8 points, ..., priority 9 = 1 point). This approach captures both the frequency and intensity of perceived importance.

Figure 2 presents a heatmap visualizing the weighted scores assigned by each respondent. Darker shades represent characteristics more frequently considered high-priority in the context of architectural technical debt. Table 4 presents the consolidated total scores, ranking the quality characteristics in terms of their perceived relevance to managing architectural technical debt.

Table 4: Q6 – Weighted Ranking of Quality Characteristics

Characteristic	Total Score	Rank
Functional Suitability	96	1
Performance Efficiency	89	2
Maintainability	82	3
Reliability	73	4
Security	69	5
Safety	66	6
Interaction Capability	62	7
Compatibility	49	8
Flexibility	45	9

The results indicate that **Functional Suitability** was rated the most important characteristic, with a total of 96 points. This suggests that, in the respondents' view, architectural debt directly

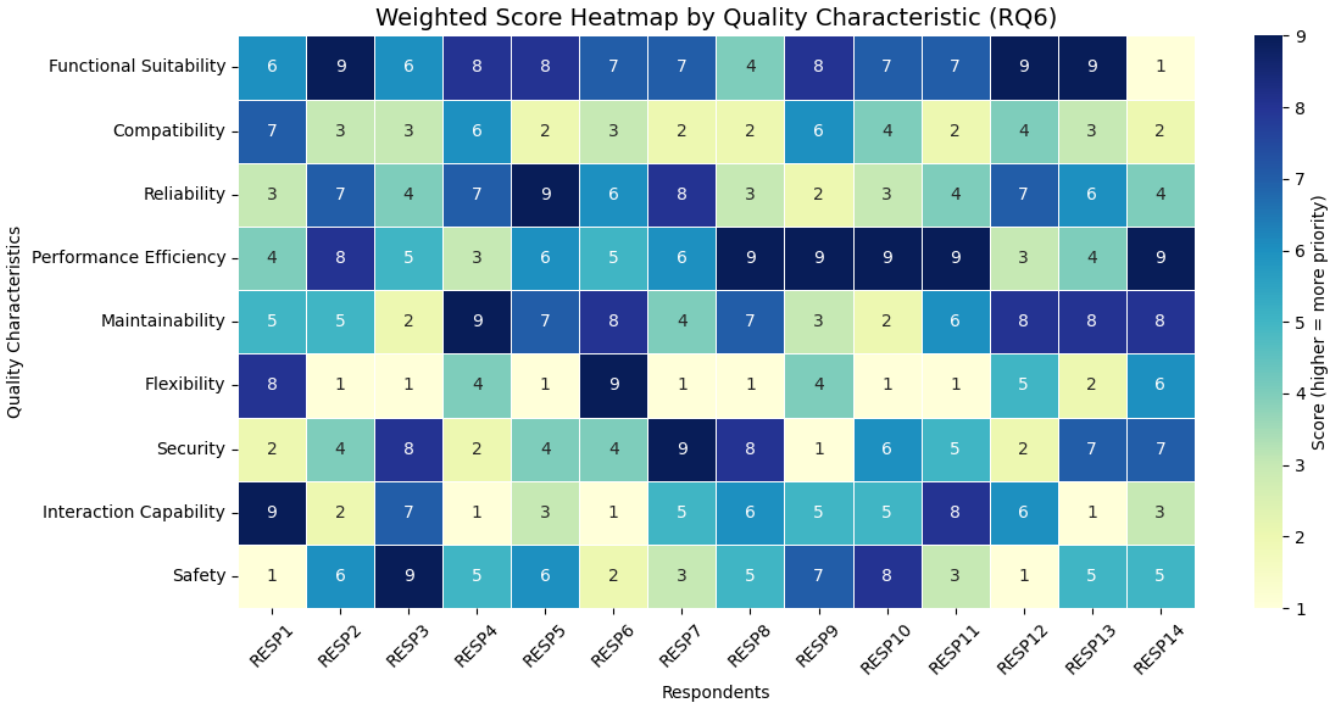

Figure 2: Q6 – Heatmap of Weighted Scores by Respondent

threatens the system's ability to fulfill its functional requirements over time, highlighting functional misalignment as a critical consequence of architectural decay.

Ranked second, **Performance Efficiency** accumulated 89 points, reflecting concerns that performance degradation is often a visible symptom of technical debt, especially in legacy or poorly optimized architectures.

Maintainability ranked third with 82 points, reaffirming its conventional link to technical debt. Maintainability reflects the effort required to modify and evolve a system and is one of the most impacted dimensions as debt accumulates.

Reliability, **Security**, and **Safety** followed in intermediate positions, suggesting these non-functional requirements are perceived as susceptible to architectural degradation, though to a lesser extent.

Conversely, **Compatibility** and **Flexibility** received lower rankings, possibly indicating that respondents associate these characteristics more with integration and technology strategy than with direct manifestations of architectural technical debt.

In summary, the Q6 findings highlight that professionals perceive architectural technical debt as primarily threatening functionality, performance, and maintainability. These perceptions can inform architectural monitoring strategies by emphasizing the quality characteristics that require closer observation to prevent or mitigate the accumulation of architectural debt throughout the software lifecycle.

5.5 Strategies for Managing Architectural Technical Debt (Q7)

This question aimed to explore the practices and routines adopted by teams to manage ATD, as perceived by the study participants. The responses underwent qualitative analysis through open coding, resulting in a set of emergent categories that represent the identified management approaches, as shown in Table 5.

The responses reveal a diverse set of practices for managing architectural technical debt, with a predominance of informal and reactive strategies. One of the most frequently mentioned categories is business-based prioritization, in which ATD is handled only when it impacts critical functionality or delays high-priority deliverables (RESP2, RESP4, RESP9, RESP10). In many cases, this prioritization emerges from alignment with technical leadership and POs (RESP1, RESP3, RESP13), often in the absence of objective criteria and based instead on subjective assessments of urgency or business value.

Other responses indicate the practice of incorporating ATD into the backlog, allowing debt to be addressed during regular sprint planning and execution (RESP6, RESP7, RESP14). A subset of participants described more systematic approaches based on mapping the criticality and impact of technical debt items (RESP5, RESP11), suggesting analytical awareness and structured decision-making.

Conversely, ATD was also described as invisible or neglected (RESP6, RESP8, RESP12), often being deprioritized or overlooked until it becomes a blocker. In such cases, remediation tends to occur only through top-down mandates or specialized task forces (RESP12, RESP14).

This variety of strategies points to the absence of a unified or standardized policy for managing architectural technical debt. Instead, the findings reveal a spectrum of practices ranging from opportunistic to more deliberate and structured approaches. These results underscore the need for transparent, objective, and value-driven mechanisms to manage ATD effectively in complex and dynamic organizational settings.

These findings further support the results of Q6, which identified **maintainability**, **performance**, and **functional suitability** as key quality attributes impacted by architectural technical debt. How teams handle this debt has a direct impact on these attributes, underscoring the need for alignment between architectural evaluation and daily development practices.

5.6 Architecture Evaluation Practices (Q8)

The objective of Q8 was to understand how software architecture evaluations are performed in agile teams. Qualitative analysis of the responses revealed that, although some recurring practices exist, these evaluations are predominantly technical and pragmatic, often lacking explicit support from evaluation models, metrics, or quality attributes.

Table 6 presents the qualitative categories that emerged from this analysis. The results indicate that, even when evaluations involve checkpoints, code reviews, deployment pipelines, or alignment with stakeholders, they are generally informal and not guided by recognized frameworks or systematic criteria.

The findings suggest that architecture evaluations are embedded in teams' technical routines, typically occurring during refinement meetings, technical discussions, and release checkpoints. These evaluations rely on established practices such as code reviews, unit testing, and CI/CD pipelines.

However, a notable gap emerges concerning the use of objective criteria. Most teams do not apply systematic methods to evaluate quality attributes such as maintainability, scalability, performance, or security. Several participants also mentioned uncertainty about whether the architecture is actually "healthy" or what steps would be needed to improve it.

This lack of structured architectural assessment limits the teams' ability to make informed decisions regarding architectural evolution and can contribute to the progressive accumulation of technical debt. The adoption of more formalized evaluation practices—such as the use of methods like ATAM (Architecture Tradeoff Analysis Method) [32] or the structured assessment of quality attributes based on ISO/IEC 25010 [30]—could provide a more objective and business-aligned basis for architectural evaluation and evolution.

These findings reinforce the results observed in Q6 and Q7. In Q6, maintainability, performance efficiency, and functional suitability emerged as the most critical attributes for managing architectural technical debt. Yet, as seen here, such attributes are rarely evaluated systematically. Q7 revealed that technical debt management is often reactive and negotiated informally, and the lack of structured architectural evaluations identified in Q8 helps explain the fragility of these management practices.

These observations are consistent with the findings of [56], who argue that architectural evaluation practices remain poorly integrated into agile and continuous development processes. The authors emphasize that the absence of systematic and automated approaches is one of the main industry challenges in maintaining architectural quality over time.

Although the qualitative analysis was supported by CAQDAS, such as Atlas.ti, the decision was made not to include graphical visualizations of code and category networks. This choice is justified by the study's focus on identifying recurring interpretive patterns, which are synthesized textually and presented in tabular form. Including such diagrams, while useful for internal analysis, would not significantly enhance the communication of results in this article format and could compromise the space allocated for discussion.

6 Discussion

The integrated analysis of the results reveals significant perceptions and practices related to software architecture in agile contexts, particularly regarding quality, technical debt management, and architectural evaluation mechanisms.

Initial Design and Quality Criteria: The findings indicate that, during the early stages of project development, most architects prioritize functional suitability, emphasizing the delivery of core features aligned with business goals. While this focus aligns with agile values, it often occurs at the expense of structural attributes such as security, maintainability, reliability, and usability, which tend to be addressed reactively or deprioritized. This tension reflects the classic trade-off between agility and architectural sustainability, as discussed by [35], where rapid design decisions may compromise structural health in the medium and long term.

Explicit Consideration of Quality Characteristics: Although participants recognize the relevance of ISO/IEC 25010 attributes, their integration into architectural decision-making is generally implicit and fragmented. These attributes are often treated as good engineering practices rather than formal evaluation criteria. There is no evidence of systematic use of frameworks, such as ATAM, for comparing architectural trade-offs. This underscores a fragility in quality governance, as highlighted by the Dagstuhl Manifesto [3], which advocates for transparency, traceability, and a strong technical foundation in architectural decisions.

Prioritization of Quality Characteristics for Technical Debt Management: The quantitative data show that Functional Suitability (96 points), Performance Efficiency (89 points), and Maintainability (82 points) were the top-prioritized attributes in the context of architectural technical debt. These preferences highlight a concern for delivering value-generating features while preserving system performance and evolvability. Conversely, lower prioritization of Flexibility (45 points) and Compatibility (49 points) suggests that long-term systemic attributes remain undervalued.

Technical Debt Management Practices: Three main patterns emerged from participants' responses: (i) ad hoc negotiation with the Product Owner, based on urgency and perceived impact; (ii) reactive handling, where debt is addressed only after it becomes critical; and (iii) structured alignment with technical leadership, which

Table 5: Q7 – Qualitative Categories on Architectural Technical Debt Management

Category	Description	Respondents
Business-Based Prioritization	ATD is addressed based on its impact on value delivery or its direct effect on product functionality.	RESP2, RESP4, RESP5, RESP6, RESP9, RESP10, RESP11
Alignment with Technical Leadership/PO	Prioritization is negotiated with TLs, POs, or PMs, often informally and without structured criteria, relying on consensus.	RESP1, RESP3, RESP4, RESP13
Backlog and Sprints	ATD items are logged in the backlog and planned for resolution during development sprints.	RESP6, RESP7, RESP10, RESP14
Mapping and Criticality	Management is guided by identification, impact analysis, criticality assessment, and effort estimation.	RESP3, RESP5, RESP11
Neglect and Invisibility	ATD is often forgotten or ignored until it becomes critical or blocks development.	RESP6, RESP8, RESP12
Task Forces and Top-Down Actions	Debt is addressed through top-down initiatives such as architecture committee mandates or dedicated task forces.	RESP12, RESP14

Table 6: Q8 – Qualitative Categories on Architecture Evaluation

Category	Description	Respondents
Unstructured Technical Evaluation	Architecture is reviewed informally based on team experience and basic engineering practices such as version control and automated testing.	RESP2, RESP4, RESP12, RESP14
Focus on Functional Delivery	Architectural decisions are validated primarily in terms of feasibility for delivery and expected system behavior.	RESP3, RESP6, RESP10
Lack of Objective Criteria	No use of quality metrics, architectural tactics, or formal patterns to guide or justify architectural decisions.	RESP1, RESP8, RESP13
Continuous but Subjective Review	The architecture is reviewed continuously, but with no clear understanding of its long-term sustainability or measurable health.	RESP5, RESP7, RESP9, RESP11

reflects greater maturity and strategic perspective. Notably, no participant mentioned the use of systematic prioritization mechanisms based on architectural value or technical complexity, revealing a gap in the continuous management of technical debt. This highlights the importance of practices such as Architecture Decision Records (ADRs) and traceability metrics, as also pointed out by initiatives like PriorTD [15].

Software Architecture Evaluation: Qualitative analysis revealed that, although architectural review practices are present—such as refinement meetings, checkpoints, and peer reviews—they are primarily driven by engineering practices (e.g., CI/CD, testing, code review) rather than formal quality attributes. Table 6 summarizes four identified categories: unstructured technical evaluation, focus on functional delivery, lack of objective criteria, and continuous but subjective review. These findings indicate that, despite adherence to technical practices, there is an overarching uncertainty regarding the actual architectural quality and its long-term sustainability.

Taken together, the findings suggest a disconnect between architectural intent and quality governance throughout the system lifecycle. While teams demonstrate strong technical skills, they often operate without systematic architectural evaluation mechanisms or formal criteria for prioritizing technical debt. The absence of structured metrics, complexity indicators, or evaluation frameworks limits informed decision-making and fosters reactive rather than proactive approaches.

This reinforces the importance of adopting standards such as ISO/IEC 25010 to guide architectural decision-making and methods like ATAM to explicitly support trade-off analysis. Furthermore, mechanisms for continuous management of architectural technical debt, such as dashboards, structured reviews, and visual tools for assessing technical impact, could increase team maturity and promote more sustainable decision-making in agile environments.

The lack of specific metrics for continuous architectural evaluation, as reported by participants, is also noted by [23], who identified a shortage of mathematical models and metrics capable of quantifying architectural impacts holistically across sustainability dimensions. This gap limits the potential for predictive analysis and evidence-based decision-making throughout the system lifecycle.

Based on these findings, future research should explore the implementation of hybrid frameworks that integrate agile practices with formal architectural evaluation methods, as well as the development of maturity or capability models specifically tailored to architectural quality management in agile environments.

7 Threats to Validity

Conducting empirical research presents methodological and interpretative challenges that may affect the validity of the results. This study considered four key dimensions of validity:

External Validity. The results should not be generalized to all software development organizations, especially those operating outside the financial sector. Although the study was conducted in a

single organization, steps were taken to mitigate threats to external validity. The participating architects had substantial experience in large-scale agile environments, which increases the likelihood that their practices and perceptions reflect broader industry patterns. The interview protocol was designed to elicit practices independent of company-specific terminology, and the results were reviewed by the authors, including one of the authors who is an experienced software solutions architect who did not participate in the study, to confirm their interpretability. In addition, the data collected was made openly available to facilitate replication in other contexts, which is part of our future research agenda.

Construct Validity. There may have been variability in how participants interpreted the quality characteristics defined by the ISO/IEC 25010:2023 standard, particularly during the prioritization task in Q6. To reduce this risk, operational definitions and illustrative examples were shown. Nonetheless, the absence of technical terminology, such as "architectural tactics" or "evaluation models", in the responses may reflect the participants' practical orientation and familiarity with software engineering rather than formal academic constructs.

Conclusion Validity. The quantitative analyses, such as the weighted ranking in Q6, are based on a small sample and do not support robust statistical inference. However, triangulation of data through the integration of quantitative methods (e.g., ranking, heatmaps) and qualitative techniques (e.g., categorization, textual analysis) — reinforces the interpretative validity of the findings. The consistency observed across questions, such as the lack of objective evaluation criteria in Q8, the functional emphasis in Q4, and the reactive nature of technical debt management in Q7, supports the internal coherence of the conclusions drawn.

8 Conclusion

This study explored software architects' perceptions regarding quality attributes and practices for managing architectural technical debt (ATD) in large-scale agile projects. The integrated analysis of questions Q4 through Q8 reveals a landscape characterized by tensions between agility, architectural sustainability, and the lack of systematic evaluation mechanisms.

Findings from Q4 and Q5 indicate that attributes such as Functional Suitability, Usability, and Security are prioritized during the initial design phase due to their direct alignment with rapid and reliable delivery. In contrast, structural attributes such as Maintainability and Performance Efficiency tend to gain prominence only after the system is in production, often triggered by incidents, reinforcing concerns raised by [35] about architectural decisions made under time pressure.

Q6 showed that participants identify Functional Suitability, Performance Efficiency, and Maintainability as the most critical attributes for managing ATD, emphasizing the centrality of functional and evolutionary sustainability in practice.

In Q7, three distinct patterns of ATD management emerged: (i) negotiation with the Product Owner; (ii) reactive resolution; and (iii) structured alignment with technical leadership. Although the latter reflects greater maturity in architectural governance, none of the approaches involved systematic employ metrics, formal evaluation practices such as ATAM, or architectural tactics referenced in the literature.

Q8 revealed that architecture evaluations are frequent but predominantly informal and based on software engineering practices such as version control, code reviews, and CI/CD pipelines. However, there is no clear consensus or objective evidence regarding the effectiveness of these evaluations or the actual state of architectural health, a challenge summarized in Table 6.

These findings underscore the need to institutionalize architecture evaluation practices guided by quality attributes (ISO/IEC 25010), prioritization mechanisms such as PriorTD [15], and collaborative initiatives like Technical Debt Guilds [16]. This aligns with the principles of the Dagstuhl Manifesto [3], which advocate for traceability, shared accountability, and evidence-based architectural decision-making.

The consistency between the empirical data and recent literature [57] highlights the importance of developing continuous architectural evaluation practices integrated into agile and DevOps processes, as a strategy for mitigating architectural technical debt.

As its primary contribution, this work proposes an interpretive model of architectural practice in large-scale agile development (LSAD) environments, emphasizing the articulation of continuous evaluation cycles, quality attributes, and sustainable architectural decisions. The study addresses a gap in the literature by offering a detailed empirical analysis of quality attribute prioritization based on the ISO/IEC 25010 standard and real-world technical debt management strategies, both still underexplored in industrial settings. The findings reveal recurring patterns and critical gaps, providing insights for the advancement of value-driven and intentional architectural practices.

Future research directions include: (i) empirical validation of architectural prioritization mechanisms that consider both technical impact and business value; and (ii) development of diagnostic instruments for the continuous assessment of architectural health in agile and dynamic contexts.

ARTIFACT AVAILABILITY

This study adopts open science practices to promote transparency and replicability. All empirical materials, including the structured interview, anonymized responses, analysis scripts, and visualizations are publicly available at: https://github.com/manoelvsneto/asapqatdm [43]. In addition to enhancing the traceability of findings, this open-access repository encourages reuse by researchers and practitioners interested in investigating architecture and technical debt management practices in agile environments.

ACKNOWLEDGMENTS

We thank the financial institution in which the case study was conducted and the software architects who generously participated in the research. We also acknowledge the valuable feedback provided by the anonymous reviewers, which helped improve the quality of this work.

The present work was carried out with the support of the Coordination for the Improvement of Higher Education Personnel - Brazil (CAPES) - Funding Code 001.

REFERENCES

[1] 2011. ISO/IEC 42010:2011 – Systems and software engineering – Architecture description.

[2] Silvio Alonso, Marcos Kalinowski, Marx L. Viana, Bruna Ferreira, and Simone D. J. Barbosa. 2021. A Systematic Mapping Study on the Use of Software Engineering Practices to Develop MVPs. In *47th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2021, Palermo, Italy, September 1-3, 2021*, Maria Teresa Baldassarre, Giuseppe Scanniello, and Amund Skavhaug (Eds.). IEEE, 62–69. https://doi.org/10.1109/SEAA53835.2021.00017

[3] Paris Avgeriou, Ipek Ozkaya, Heiko Koziolek, Zadia Codabux, and Neil Ernst. 2025. Manifesto from Dagstuhl Perspectives Workshop 24452 – Reframing Technical Debt. arXiv:2505.13009 [cs.SE] https://arxiv.org/abs/2505.13009

[4] Muhammad Ali Babar, Alan W. Brown, and Ivan Mistrik. 2013. *Agile Software Architecture: Aligning Agile Processes and Software Architectures* (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[5] Muhammed Ali Babar, Barbara Kitchenham, and Ian Gorton. 2006. Towards a distributed software architecture evaluation process: a preliminary assessment. In *Proceedings of the 28th International Conference on Software Engineering* (Shanghai, China) *(ICSE '06)*. Association for Computing Machinery, New York, NY, USA, 845–848. https://doi.org/10.1145/1134285.1134430

[6] Felix Bachmann, Len Bass, and Mark Klein. 2003. *Deriving Architectural Tactics: A Step Toward Methodical Architectural Design.* Technical Report CMU/SEI-2003-TR-004. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=6593

[7] Victor R. Basili and H. Dieter Rombach. 1988. The TAME Project: Towards Improvement-Oriented Software Environments. *IEEE Trans. Software Eng.* 14, 6 (1988), 758–773. https://doi.org/10.1109/32.6156

[8] L. Bass, P. Clements, R. Kazman, and an O'Reilly Media Company Safari. 2021. *Software Architecture in Practice, 4th Edition.* Addison-Wesley Professional. https://books.google.com.br/books?id=BWpuzgEACAAJ

[9] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. 2001. Manifesto for Agile Software Development. Retrieved May 29, 2007, from http://www.agilemanifesto.org/.

[10] Stephany Bellomo, Philippe Kruchten, Robert L. Nord, and Ipek Ozkaya. 2014. How to agilely architect an agile architecture. *Cutter IT Journal* 27, 2 (2014), 10 – 15. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84896271838&partnerID=40&md5=dfaac86f854b4758d7fb92b8d5623050 Cited by: 14.

[11] Terese Besker, Antonio Martini, and Jan Bosch. 2016. A Systematic Literature Review and a Unified Model of ATD. In *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 189–197. https://doi.org/10.1109/SEAA.2016.42

[12] Pierre Bourque and Richard E. Fairley (Eds.). 2014. *SWEBOK: Guide to the Software Engineering Body of Knowledge* (version 3.0 ed.). IEEE Computer Society, Los Alamitos, CA. http://www.swebok.org/

[13] Lucas Carvalho, João Biazotto, Daniel Feitosa, and Elisa Nakagawa. 2024. Technical Debt in Continuous Software Engineering: An Overview of the State of the Art and Future Trends. In *Anais do XXVII Congresso Ibero-Americano em Engenharia de Software* (Curitiba/PR). SBC, Porto Alegre, RS, Brasil, 313–326. https://doi.org/10.5753/cibse.2024.28456

[14] Ward Cunningham. 1993. The WyCash portfolio management system. *OOPS Messenger* 4, 2 (1993), 29–30. https://doi.org/10.1145/157710.157715

[15] Thober Detofeno, Andreia Malucelli, and Sheila Reinehr. 2022. PriorTD: A Method for Prioritization Technical Debt. In *Proceedings of the XXXVI Brazilian Symposium on Software Engineering* (Virtual Event, Brazil) *(SBES '22)*. Association for Computing Machinery, New York, NY, USA, 230–240. https://doi.org/10.1145/3555228.3555238

[16] Torgeir Dingsøyr, Tore Dybå, and Nils Brede Moe (Eds.). 2010. *Agile Software Development - Current Research and Future Directions.* Springer. https://doi.org/10.1007/978-3-642-12575-1

[17] Torgeir Dingsøyr and Nils Brede Moe. 2014. Towards Principles of Large-Scale Agile Development - A Summary of the Workshop at XP2014 and a Revised Research Agenda. In *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation - XP 2014 International Workshops, Rome, Italy, May 26-30, 2014, Revised Selected Papers (Lecture Notes in Business Information Processing, Vol. 199)*, Torgeir Dingsøyr, Nils Brede Moe, Roberto Tonelli, Steve Counsell, Çigdem Gencel, and Kai Petersen (Eds.). Springer, 1–8. https://doi.org/10.1007/978-3-319-14358-3_1

[18] Torgeir Dingsøyr, Nils Brede Moe, Roberto Tonelli, Steve Counsell, Çigdem Gencel, and Kai Petersen (Eds.). 2014. *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation - XP 2014 International Workshops, Rome, Italy, May 26-30, 2014, Revised Selected Papers.* Lecture Notes in Business Information Processing, Vol. 199. Springer. https://doi.org/10.1007/978-3-319-14358-3

[19] Murat Erder and Pierre Pureur. 2016. *Continuous Architecture: Sustainable Architecture in an Agile and Cloud-Centric World.* Morgan Kaufmann, Amsterdam. http://www.sciencedirect.com/science/book/9780128032848

[20] Neil A. Ernst, Stephany Bellomo, Ipek Ozkaya, Robert L. Nord, and Ian Gorton. 2015. Measure it? Manage it? Ignore it? software practitioners and technical debt. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, Elisabetta Di Nitto, Mark Harman, and Patrick Heymans (Eds.). ACM, 50–60. https://doi.org/10.1145/2786805.2786848

[21] George Fairbanks. 2014. Architectural Hoisting. *IEEE Softw.* 31, 4 (2014), 12–15. https://doi.org/10.1109/MS.2014.82

[22] George Fairbanks, Kevin Bierhoff, and Desmond D'Souza. 2006. Software architecture at a large financial firm. In *Companion to the 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006, October 22-26, 2006, Portland, Oregon, USA*, Peri L. Tarr and William R. Cook (Eds.). ACM, 815–823. https://doi.org/10.1145/1176617.1176729

[23] Iffat Fatima and Patricia Lago. 2023. A Review of Software Architecture Evaluation Methods for Sustainability Assessment. In *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*. 191–194. https://doi.org/10.1109/ICSA-C57050.2023.00050

[24] Rafaela Mantovani Fontana, Sheila S. Reinehr, and Andreia Malucelli. 2016. Maturity in Agile Software Development. In *Proceedings of the 15th Brazilian Symposium on Software Quality, SBQS 2016, Maceió, AL, Brazil, October 24-26, 2016*, Gleison Santos and Heitor A. X. Costa (Eds.). SBC, 406–419. https://doi.org/10.5753/SBQS.2016.15149

[25] Susan J. Fowler. 2016. *Production-Ready Microservices: Building Standardized Systems Across an Engineering Organization* (1st ed.). O'Reilly Media, Inc.

[26] David Garlan. 2000. Software architecture: a roadmap. In *22nd International Conference on on Software Engineering, Future of Software Engineering Track, ICSE 2000, Limerick Ireland, June 4-11, 2000*, Anthony Finkelstein (Ed.). ACM, 91–101. https://doi.org/10.1145/336512.336537

[27] The Open Group. 2017. Open Agile Architecture. Open Agile Architecture™: A Standard of The Open Group.

[28] Brian Hobbs and Yvan Petit. 2017. Agile Methods on Large Projects in Large Organizations. *Project Management Journal* 48, 3 (2017), 3–19. https://doi.org/10.1177/875697281704800301 arXiv:https://doi.org/10.1177/875697281704800301

[29] Chris H. Hoeseb and Maureen Tanner. 2020. Large-Scale Agile Implementation in Large Financial Institutions: A Systematic Literature Review. In *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*. 1780–1786. https://doi.org/10.1109/CSCI51800.2020.00329

[30] International Organization for Standardization. 2023. ISO/IEC 25010:2023, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.

[31] Mohamad Kassab. 2017. A Contemporary View on Software Quality Requirements in Agile and Software Architecture Practices. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*. 260–267. https://doi.org/10.1109/REW.2017.60

[32] Rick Kazman, Mark Klein, Mario Barbacci, Thomas Longstaff, Howard Lipson, and S. Carriere. 1998. *The Architecture Tradeoff Analysis Method.* Technical Report CMU/SEI-98-TR-008. https://insights.sei.cmu.edu/library/the-architecture-tradeoff-analysis-method/

[33] Michael Keeling. 2015. Architecture Haiku: A Case Study in Lean Documentation [The Pragmatic Architect]. *IEEE Software* 32, 3 (2015), 35–39. https://doi.org/10.1109/MS.2015.59

[34] Philippe Kruchten. 1995. Mommy, where do software architectures come from. In *Proceedings of the 1st Intl. Workshop on Architectures for Software Systems.* 198–205.

[35] Philippe Kruchten, Robert L. Nord, and Ipek Ozkaya. 2012. Technical Debt: From Metaphor to Theory and Practice. *IEEE Softw.* 29, 6 (2012), 18–21. https://doi.org/10.1109/MS.2012.167

[36] Robin Lichtenthäler and Guido Wirtz. 2022. Towards a Quality Model for Cloud-native Applications. In *European Conference on Service-Oriented and Cloud Computing*. Springer, 109–117.

[37] Antonio Martini, Jan Bosch, and Michel Chaudron. 2014. Architecture Technical Debt: Understanding Causes and a Qualitative Model. In *40th EUROMICRO Conference on Software Engineering and Advanced Applications, EUROMICRO-SEAA 2014, Verona, Italy, August 27-29, 2014.* IEEE Computer Society, 85–92. https://doi.org/10.1109/SEAA.2014.65

[38] A. Meiappane, B. Chitra, and Prasanna Venkataesan. 2013. Evaluation of Software Architecture Quality Attribute for an Internet Banking System. *International Journal of Computer Applications* 62, 19 (Jan. 2013), 21–24. https://doi.org/10.5120/10189-5062

[39] Hind Milhem and Neil B. Harrison. 2022. The Quality Attibutes and Architectural Tactics of Amazon Web Services (AWS). In *2022 Intermountain Engineering, Technology and Computing (IETC)*. 1–6. https://doi.org/10.1109/IETC54973.2022.9796821

[40] Ivan Mistrík, Rami Bahsoon, Rick Kazman, and Yuanyuan Zhang (Eds.). 2014. *Economics-Driven Software Architecture.* Morgan Kaufmann / Academic Press / Elsevier. https://doi.org/10.1016/c2012-0-02842-6

[41] Gastón Márquez, Hernán Astudillo, and Rick Kazman. 2023. Architectural tactics in software architecture: A systematic mapping study. *Journal of Systems and*

Software 197 (2023), 111558. https://doi.org/10.1016/j.jss.2022.111558

[42] Gastón Márquez, Mónica M. Villegas, and Hernán Astudillo. 2018. An Empirical Study of Scalability Frameworks in Open Source Microservices-based Systems. In *2018 37th International Conference of the Chilean Computer Science Society (SCCC)*. 1–8. https://doi.org/10.1109/SCCC.2018.8705256

[43] Manoel Valerio Da Silveira Neto, Andreia Malucelli, and Sheila Reinehr. 2024. Agile Software Architecture: Perceptions on Quality and Architectural Technical Debt Management. https://github.com/manoelvsneto/asapqatdm.

[44] Sam Newman. 2021. *Building Microservices* (2st ed.). O'Reilly Media, Inc.

[45] Robert L. Nord, Ipek Ozkaya, and Philippe Kruchten. 2014. Agile in Distress: Architecture to the Rescue. In *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation*, Torgeir Dingsøyr, Nils Brede Moe, Roberto Tonelli, Steve Counsell, Cigdem Gencel, and Kai Petersen (Eds.). Springer International Publishing, Cham, 43–57.

[46] Sascha Nägele, Jan-Philipp Watzelt, and Florian Matthes. 2022. *Investigating the Current State of Security in Large-Scale Agile Development*. 203–219. https://doi.org/10.1007/978-3-031-08169-9_13

[47] Felipe Osses, Gastón Márquez, and Hernán Astudillo. 2018. Exploration of Academic and Industrial Evidence about Architectural Tactics and Patterns in Microservices. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 256–257. https://doi.org/10.1145/3183440.3194958

[48] Mert Ozkaya and Nurdan Canbaz. 2019. Towards Understanding Industry?s Perspectives on the Software Quality Characteristics: A Survey. In *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering* (Heraklion, Crete, Greece) *(ENASE 2019)*. SCITEPRESS - Science and Technology Publications, Lda, Setubal, PRT, 417–426. https://doi.org/10.5220/0007742004170426

[49] Kai Petersen and Claes Wohlin. 2010. The effect of moving from a plan-driven to an incremental software development approach with agile practices - An industrial case study. *Empir. Softw. Eng.* 15, 6 (2010), 654–693. https://doi.org/10.1007/s10664-010-9136-6

[50] Giuseppe Procaccianti, Patricia Lago, and Grace A. Lewis. 2014. A Catalogue of Green Architectural Tactics for the Cloud. In *2014 IEEE 8th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems*. 29–36. https://doi.org/10.1109/MESOCA.2014.12

[51] Giuseppe Procaccianti, Patricia Lago, and Grace A. Lewis. 2014. Green Architectural Tactics for the Cloud. In *2014 IEEE/IFIP Conference on Software Architecture*. 41–44. https://doi.org/10.1109/WICSA.2014.30

[52] Nicolli Rios, Manoel G. Mendonça, Carolyn B. Seaman, and Rodrigo O. Spínola. 2019. Causes and Effects of the Presence of Technical Debt in Agile Software Projects. In *25th Americas Conference on Information Systems, AMCIS 2019, Cancún, Mexico, August 15-17, 2019*. Association for Information Systems. https://aisel.aisnet.org/amcis2019/systems_analysis_design/systems_analysis_design/1

[53] Thomas L. Saaty. 1980. *The Analytic Hierarchy Process*. McGraw-Hill, New York.

[54] J. Saldaña. 2021. *The Coding Manual for Qualitative Researchers*. SAGE. https://books.google.com.br/books?id=p6UIzgEACAAJ

[55] Johnny M. Saldana. 2015. *The coding manual for qualitative researchers* (3 ed.). SAGE Publications, London, England.

[56] Rodrigo C. Soares, Rafael Capilla, Vinicius dos Santos, and Elisa Yumi Nakagawa. 2023. Trends in continuous evaluation of software architectures. *Computing* (2023). https://doi.org/10.1007/s00607-023-01161-1

[57] Rodrigo C. Soares, Vinicius dos Santos, and Elisa Yumi Nakagawa. 2022. Continuous evaluation of software architectures: an overview of the state of the art. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing* (Virtual Event) *(SAC '22)*. Association for Computing Machinery, New York, NY, USA, 1425–1431. https://doi.org/10.1145/3477314.3507318

[58] Ömer Uludağ and Florian Matthes. 2020. Large-Scale Agile Development Patterns for Enterprise and Solution Architects. In *Proceedings of the European Conference on Pattern Languages of Programs 2020* (Virtual Event, Germany) *(EuroPLoP '20)*. Association for Computing Machinery, New York, NY, USA, Article 29, 22 pages. https://doi.org/10.1145/3424771.3424895

[59] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, and Björn Regnell. 2012. *Experimentation in Software Engineering*. Springer. I–XXIII, 1–236 pages.

[60] Rob Wojcik, Felix Bachmann, Len Bass, Paul C. Clements, Paulo Merson, Robert Nord, and William G. Wood. 2006. *Attribute-Driven Design (ADD), Version 2.0*. Technical Report. Software Engineering Institute.

[61] William G. Wood. 2007. A Practical Example of Applying Attribute-Driven Design (ADD), Version 2.0. https://insights.sei.cmu.edu/documents/781/2007_005_001_14867.pdf

[62] Chen Yang, Peng Liang, and Paris Avgeriou. 2016. A systematic mapping study on the combination of software architecture and agile development. *Journal of Systems and Software* 111 (2016), 157–184. https://doi.org/10.1016/j.jss.2015.09.028

[63] R.K. Yin. 2017. *Case Study Research and Applications: Design and Methods*. SAGE Publications. https://books.google.com.br/books?id=6DwmDwAAQBAJ