Architecture decisions in quantum software systems: An empirical study on Stack Exchange and GitHub

Mst Shamima Aktar [a], Peng Liang [a,*], Muhammad Waseem [b], Amjed Tahir [c], Aakash Ahmad [d], Beiqi Zhang [a], Zengyang Li [e]

[a] *School of Computer Science, Wuhan University, 430072 Wuhan, China*
[b] *Faculty of Information Technology, University of Jyväskylä, FI 40014 Jyväskylä, Finland*
[c] *School of Mathematical and Computational Sciences, Massey University, 4442 Palmerston North, New Zealand*
[d] *School of Computing and Communications, Lancaster University Leipzig, 04109 Leipzig, Germany*
[e] *School of Computer Science, Central China Normal University, Wuhan, China*

ARTICLE INFO

ABSTRACT

Context: Quantum computing provides a new dimension in computation, utilizing the principles of quantum mechanics to potentially solve complex problems that are currently intractable for classical computers. However, little research has been conducted about the architecture decisions made in quantum software development, which have a significant influence on the functionality, performance, scalability, and reliability of these systems.

Objective: The study aims to empirically investigate and analyze architecture decisions made during the development of quantum software systems, identifying prevalent challenges and limitations by using the posts and issues from Stack Exchange and GitHub.

Methods: We used a qualitative approach to analyze the obtained data from Stack Exchange Sites and GitHub projects — two prominent platforms in the software development community. Specifically, we collected data from 385 issues (from 87 GitHub projects) and 70 posts (from 3 Stack Exchange sites) related to architecture decisions in quantum software development.

Results: The results show that in quantum software development (1) architecture decisions are articulated in six linguistic patterns, the most common of which are *Solution Proposal* and *Information Giving*, (2) the two major categories of architectural decisions are *Implementation Decision* and *Technology Decision*, (3) *Software Development Tools* are the most common application domain among the twenty application domains identified, (4) *Maintainability* is the most frequently considered quality attribute, and (5) *Design Issues* and *High Error Rates* are the major limitations and challenges that practitioners face when making architecture decisions in quantum software development.

Conclusions: Our results show that the limitations and challenges encountered in architecture decision-making during the development of quantum software systems are strongly linked to the particular features (e.g., quantum entanglement, superposition, and decoherence) of those systems. These issues mostly pertain to technical aspects and need appropriate measures to address them effectively.

1. Introduction

Quantum computing is a rapidly developing technology that is a combination of physics, chemistry, mathematics, computer science, and information theory to revolutionize computation and problem-solving capabilities [1]. Classical computing processes information using classical bits, which can represent a value of either 0 or 1. In contrast, quantum computing introduces qubits or quantum bits, which can represent two-state $|0\rangle$ and $|1\rangle$, serving as the fundamental units of quantum information technology [2]. As a mature technology, classical computing is making significant contributions to the advancement of today's digital world [3,4]. Despite its invaluable contribution to the landscape of computation, classical computing faces inherent limitations in data encoding, processing, and transmission, particularly when managing complex tasks such as optimizing energy systems, conducting

large-scale simulations, and enhancing cryptographic security areas — where quantum computing shows enormous potential [5–8]. As a result, quantum computing has largely been motivated by academic (e.g., institutions like MIT and Stanford doing research in quantum algorithms) and industrial interests (e.g., companies like Google and IBM developing quantum technologies) to change the scenery of information technology, and the role of quantum software architecture has been recognized to bring impactful changes in this transformation process [9–11]. Integrating quantum software architecture and programming with quantum hardware is a foundational aspect of quantum computing research [12,13]. This integration makes it possible to develop and execute quantum algorithms designed to utilize quantum bit's power and solve complex problems more efficiently than classical algorithms [14]. The most fundamental difference between quantum and classical algorithms lies in their state representation — classical algorithms use probabilities while quantum algorithms use amplitudes [15]. Therefore, central to the potency of quantum computing are quantum algorithms, including Shor's large numbers factoring algorithm [16] and Grover's quantum search algorithm [17]. Quantum algorithms are used across various domains, such as cryptography, search and optimization processes, and solving linear equation systems [18,19]. Quantum programming entails developing algorithms specifically for quantum computers and reducing the execution costs of quantum algorithms on future devices [20]. Quantum programs are written using domain-specific programming languages such as Q# and Silq, and quantum software architecture is important to develop quantum software systems [21,22]. However, the fields of quantum software architecture and quantum software engineering are still in their early stages of development [23–25]. Compounding this, previous studies highlight the considerable obstacles that quantum software development grapples with, such as intricate programming paradigms, scarce simulation resources, high error frequencies, a dearth of efficient debugging tools, and limited software libraries. These obstacles accentuate the intricacies involved in architecture decisions for quantum software systems [26,27].

In the domain of software, quantum computing is ready to begin renovations "golden age", changing the landscape of software engineering [28]. Over the last few years, the development of quantum programming languages, algorithms, libraries, and tools has received much attention [29], quantum software architecture has so far received little attention, which significantly impacts the research and practice of quantum software systems [30]. Different companies (e.g., Qiskit[1] from IBM, Cirq[2] from Google Research, Q#[3] from Microsoft, and C++[4] from Intel) and cloud providers (e.g., Amazon Web Services, Alibaba Cloud, and Azure Quantum) have already invested in tools as well as programming languages that allow developers, researchers, and users to solve real-world problems (e.g., risk assessment and fraud detection in banking, drug discovery, protein folding, supply chain optimization, energy distribution, and advertising scheduling) [31]. Quantum programming languages emphasize computational and implementation specifics to create executable specifications, yet often fail to incorporate a comprehensive, global perspective on the software systems being designed can lead to a reduction in essential architecture views, potentially compromising the quality and functionality of the final quantum software product [21,32,33]. Concurrently, quantum computing has an impact on all software life cycle processes and techniques, and quantum software encounters multifaceted challenges [34], such as making existing software quantum-safe [35].

Software architecture is the high-level design and organization of a software system, including its structural components, interactions, and behavior [36]. Software architecture for quantum software systems structures quantum-intensive systems, utilizing quantum bits and quantum gates as the essential components and connectors for the software architecture design and implementation [37,38]. The architecture of a quantum software system is designed through a decision-making process, where the architecture is viewed as a composition of a set of explicit design decisions [39,40]. Decision-making significantly influences the success or failure of a software system, as a high-quality architecture decision-making process can lead to desirable results, while a poor-quality architecture and suboptimal architecture decisions more likely lead to undesirable outcomes and even architectural technical debt [41,42]. Making architecture decisions during the development process of quantum software systems is complex, requiring a comparative structure to analyze and select among various tools, services, and techniques based on specific use cases and algorithms [43]. Architecture decisions are increasingly influenced by practitioner feedback, focusing on hybrid quantum–classical patterns and service orientation to efficiently address emerging computational challenges in quantum software [44]. Architecture decisions also affect many aspects of a software system, including its quality (e.g., maintainability, testability, and reliability) of quantum software development [45]. Recent workshops, such as the Quantum Software Architecture Workshop (QSA), highlight the growing importance of architectural considerations in the domain of quantum computing [46]. Despite the essential importance of architecture decisions in the development of quantum software systems, architecture decisions are unexplored in the area of quantum software development [45]. Software architects encounter various quantum software architecture limitations and challenges and should carefully consider overcoming architecture challenges in a quantum software system [30,38]. Recent studies also identify complex challenges, particularly in quantum software development, which is directed towards the intricate process of architecting and implementing quantum software systems, with structural difficulties surfacing even for seasoned architects and developers [44]. To tackle the challenges of quantum software architecture, the current generation of architects and developers often find themselves inadequately prepared for the development of quantum software systems [47–49]. Despite the myriad of recent literature on quantum software engineering and architecture, a noticeable gap remains with the lack of empirical studies analyzing critical architecture decision-making within quantum software systems [30,37,50]. To this end, this study aims to analyze the architecture decisions as well as the challenges in architecture decision-making associated with quantum software development from discussions made between developers in GitHub issues and Stack Exchange posts.

To achieve the goal of this study (see Section 3.1), we conducted an empirical study to investigate architectural decisions made in quantum software development. We collected data from two sources: GitHub issues (of open-source quantum software projects) and Stack Exchange Q&A sites (Stack Overflow, Quantum Computing, and Computer Science). More specifically, we collected 3185 open and closed issues (from 192 GitHub projects) and 1494 posts (from the Stack Exchange sites). We manually filtered out irrelevant issues and posts, and we finally got 385 related issues (from 87 quantum software projects) and 70 posts from Stack Exchange. Furthermore, we manually extracted data from the 385 related issues and 70 posts and analyzed the extracted data using a predefined classification [51] and the Constant Comparison method [52] for answering the research questions.

Our study results show that (1) *Solution Proposal* and *Information Giving* are the most frequently used linguistic patterns, (2) *Implementation Decision* and *Technology Decision* are the two main types of architecture decisions, (3) *Software Development Tool* is the most common application domain identified, (4) *Maintainability* is the dominant quality attribute considered, and (5) *Design Issues* and *High Error Rates* are the main limitations and challenges when making architecture decisions in quantum software development.

---

[1] https://qiskit.org/.

[2] https://quantumai.google/cirq.

[3] https://learn.microsoft.com/en-us/azure/quantum.

[4] https://www.intel.com/content/www/us/en/developer/tools/quantum-sdk/overview.html.

The contributions of this work: (1) we identified the linguistic patterns practitioners utilized when making architecture decisions in developing quantum software systems; (2) we investigated several facets of architecture decisions in quantum software development, such as the categories of decisions made, the application domains, and the quality attributes considered; (3) we present and discuss the challenges and limitations developers face while designing quantum software systems; (4) we provide guidelines to help practitioners design quantum software systems with informed architectural decisions; and (5) we constructed a dataset [53] that collects the architecture decisions made in quantum software development.

The rest of this paper is structured as follows: Section 2 presents the background of the study. Section 3 describes the research methodology. Section 4 provides the study results, followed by a discussion of these results in Section 5. Section 6 clarifies the validity threats. Section 7 discusses the related work. Finally, Section 8 concludes the work with future research directions.

2. Background

In this section, we introduce the background concepts used in this study, including quantum computing, quantum software engineering, and architecture decisions. The overview of the background is shown in Fig. 1.

2.1. Quantum computing

Quantum computing is a model of computation that focuses on designing and developing computer-based systems utilizing the principles of quantum mechanics (e.g., superposition, entanglement, and interference) to process information.

- **Superposition**: Superposition is a fundamental principle of quantum mechanics that allows quantum systems, or quantum bits (qubits), to exist simultaneously in multiple states. Unlike classical bits that can only be 0 or 1 at any one time, a qubit can occupy the states $|0\rangle$ and $|1\rangle$, or any quantum superposition of these states. This superposition capability represents qubit states within a matrix format, as illustrated in Eq. (1). This capability allows qubits to perform parallel computations, enhancing computational efficiency and optimizing system performance.

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{1}$$

- **Entanglement**: Quantum entanglement is a unique feature of quantum physics, which is not present in classical mechanics. Quantum entanglement uniquely connects quantum bits (qubits), so the state of one qubit instantly correlates with the state of another, regardless of the distance between them. Consequently, researchers cannot describe the state of each qubit independently of the others. Entangled qubits can thus provide a robust synchronization necessary for quantum computing, which is essential for specific algorithms that require complex correlation patterns.

- **Interference**: Qubits, the fundamental units of quantum information, can exist in multiple states simultaneously due to superposition. This characteristic leads to quantum interference, which reflects the likelihood of qubits assuming specific states upon measurement. Quantum computing systems, therefore, implement strategies to minimize this interference, aiming to enhance the precision of their computational outcomes.

Quantum gates are fundamental to quantum computing, analogous to classical logic gates but designed for operations on qubits. These gates manipulate qubits in superposition states and are typically represented as matrices that execute specific operations on targeted qubits. The most frequently encountered gates include the Pauli X (Eq. (2)), Pauli Y (Eq. (3)), Phase Shift (Pauli Z) gate (Eq. (4)), and

the Hadamard gate (Eq. (5)). These gates play a foundational role in quantum mechanics, often referred to as the 'quantum state basis', translating algorithmic mathematics into quantum circuits, assessing state probabilities, and forecasting an algorithm's performance. For a qubit $|X\rangle$ to be in this state basis, it must conform to a linear combination represented by $|x\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are complex numbers ensuring the normalization condition equals 1 that is, $|\alpha|^2 + |\beta|^2 = 1$ This concept is also known as the superposition of two basis states.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2}$$

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \tag{3}$$

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{4}$$

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{5}$$

In the early 1980s, physicist Richard Feynman proposed the notion of simulating quantum systems using quantum mechanical processes [54]. Over a couple of decades, quantum computing has transitioned from theoretical concepts to practical implementations, driven by exponentially increasing computational potential [55]. This transformation has motivated research into quantum algorithms, error correction, and quantum simulation [56]. Quantum computing plays an important role in several fields of computer science most notably natural language processing [57], machine learning [58], and cyber security [59]. To realize practical quantum applications, it is essential to have quantum software tools for various design tasks across multiple abstraction levels, along with practical benchmarks to empirically evaluate and compare these tools against the current state of the art [60]. However, the development of practical quantum systems remains difficult due to the lack of reliability requirements of quantum computing algorithms, physical machines foreseen, and hardware limitations [61]. Quantum Computing as a Service (QCaaS) by establishing a reference architecture, demonstrates its practical application through microservices in the quantum-classic divide and validates its effectiveness and reusability via insights from professionals in the field of quantum software engineering.

2.2. Quantum software engineering

Quantum Software Engineering (QSE) represents a transformative approach combining the principles of quantum mechanics with the established practices of software engineering, aiming to design, develop, validate, and evolve software systems and applications fit for the quantum era [44]. The transition from theoretical concepts to practical implementations in quantum computing has seen significant milestones, such as Google's demonstration of "quantum supremacy" in October 2019, showcasing quantum computers solving problems beyond the reach of classical computers [62]. Classical software engineering involves methods and techniques for developing large-scale software systems, using the engineering metaphor to highlight a systematic approach that meets organizational requirements and constraints [63]. On the other hand, QSE signifies a new era of software-intensive engineering activities designed to develop applications that manage the underlying hardware, enabling the cost-effective and scalable development of dependable quantum software systems across various domains [27, 33]. Compared with classical software engineering, quantum software engineering has inherent complexity and presents multifaceted challenges (e.g., the requisite for physics knowledge among developers who often lack software engineering experience, particularly software processes, quality practices, complex quantum algorithms, and programming) that emphasize the significance of QSE [64]. The life cycle
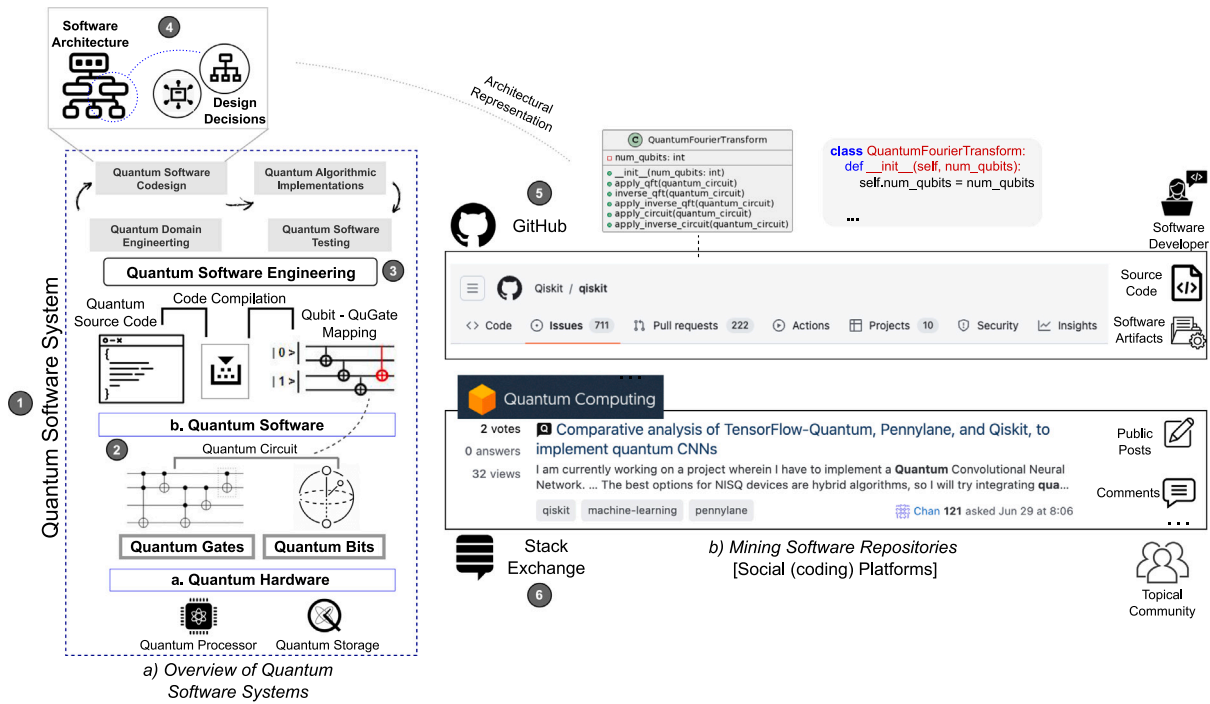
Fig. 1. (a) Overview of quantum computing systems, (b) Mining software repositories [Social (coding) Platforms].

in QSE is comprehensive, encompassing quantum requirements engineering, design, implementation, testing, and maintenance, requiring collaborative efforts from both scientific and industry to propel real-world applicability [65]. The intersection of software engineering and quantum software engineering underscores the necessity for a tailored software development life cycle, which accentuates the importance of requirements engineering, particularly in the quantum domain, where it facilitates communication among stakeholders and helps in comprehending complex application domains [66]. QSE is a growing field of study and Quantum Software Architectures (QSA) is a sub-domain of study — both are understudied and face research challenges [30]. More specifically, QSE provides the methodologies, techniques, and tools required to systematically develop, operate, and maintain software for quantum computers, while QSA defines the structure of the system, including the quantum software components, their properties, and the interfaces and interactions between them that are involved in the design of large-scale quantum software applications and their implementation on quantum hardware. This task is challenging, requiring software development that aligns with the laws of quantum comprehension [67,68]. QSA is often characterized by unique challenges such as defining suitable quantum algorithms, handling quantum data structures, and the crucial task of choosing suitable quantum hardware for the optimal execution of these software architectures, which are part of the larger research hurdles within QSE [12,32]. Both the software engineering and software architecture research communities have been making concerted efforts to cultivate dedicated platforms such as conferences and workshops. The aim is to define target outcomes, simplify emerging issues, and introduce collective initiatives pertaining to the engineering and architecture planning for the development and structural design of quantum software [46,69,70]. Recent studies emphasize several particularly difficult aspects of QSE that diverge from traditional software; including requirements issues and challenges for architecting, interpreting quantum program outputs, explaining the theory behind quantum computing code, reducing the understanding gap between quantum and classical computing, and testing QSE applications, along with pinpointing opportunities that arise from these very complexities [64,71]. QSE seeks to deal with those evolving challenges of quantum software development, requiring

particular architecture decisions within the decision-making process to overcome these challenges [72].

2.3. Architecture decision

Software architecture is a blueprint that guides the development process, ensuring the system's quality attributes (e.g., maintainability, performance, scalability) and aligning technical decisions with business goals [73]. Classical software architecture is the set of structures of a software system that consists of the various components, the relationships among those components, and their respective properties, which are necessary to understand and create the system [36]. Quantum software architecture represents a novel category of software architecture that can abstract complicated and implementation-specific activities while offering architectural descriptions (e.g., components, connections, and configurations) to design and develop quantum software [38]. In the realm of software engineering and architecture design, software architecture decisions can be defined as critical design decisions that address significant architecture requirements and are generally considered complex to formulate and potentially expensive to alter [74]. A comparative analysis reveals that architecture decisions in classical software development focus on optimizing performance, scalability, and maintainability using deterministic algorithms and well-defined architecture patterns and binary data structures [75], while architecture decisions in quantum software development focus on quantum algorithms and well-defined quantum architecture patterns and quantum data structures to address unique challenges (e.g., algorithms, programming, design, implementation, and testing), which require suitable architecture decisions [76,77]. The importance of architecture decisions in quantum software systems is also highlighted by a systematic review on software architecture for quantum computing systems [38]. In many contexts, including the Internet of Things (IoT) [78], machine learning-based systems [79], and microservices [80], empirical studies and systematic mappings emphasize the importance of architecture decisions. The documentation, communication, and evaluation of these crucial decisions are made easier by existing models and tools [81]. As new architectural styles and paradigms arise, these decisions help developers manage common challenges in

Table 1
Research questions and their rationale.

Research question	Rationale
RQ1. How do developers express architecture decisions made in quantum software development?	This RQ aims to provide the linguistic patterns used by developers to express architecture decisions in the development of quantum software systems. Developers express architecture decisions in quantum software systems through various linguistic patterns to explain their ideas and communicate with their peers on different platforms (e.g., GitHub, Stack Exchange). The answer to this RQ can help developers identify architecture decision-related content in quantum systems development.
RQ2. What types of architecture decisions are made in quantum software development?	This RQ intends to categorize the architecture decisions made in quantum software systems. Developers make a number of architecture decisions while developing quantum software systems, and these decisions cover various aspects of architecture design (e.g., architecture patterns, and technology). The answer to this RQ can offer insights into the specific categories of architectural decisions being made in quantum software systems.
RQ3. What are the application domains of quantum software in which architecture decisions are made?	This RQ aims to analyze the application domains of quantum software systems in which architecture decisions are made. Quantum technologies are widely employed in developing software systems across many application domains (e.g., optimization, cryptography, materials science, and machine learning). The answer to this RQ may provide guidance for making architectural decisions in particular domains.
RQ4. What quality attributes are considered when developers make architecture decisions in quantum software development?	This RQ aims to provide the quality attributes that developers focus on when making architecture decisions in quantum software development. Quantum software systems have dedicated concerns and the answer to this RQ can provide a comprehensive set of quality concerns highlighted in quantum software development.
RQ5. What are the limitations and challenges of making architecture decisions in quantum software development?	This RQ aims to investigate the limitations and challenges developers may encounter when making architecture decisions for quantum software systems with related quantum components. Making architecture decisions in the creation of quantum software still has a number of limitations and challenges. The answer to this RQ can provide the directions for future exploration.

architecture design [40]. The relevance of architecture decisions even in cutting-edge fields is reflected in practical perspectives on agile quantum software development [82]. When software systems become more complex, understanding, clearly expressing, and making effective architecture decisions are essential for achieving software quality and meeting stakeholder needs.

3. Research design

3.1. Research goal and research questions

The goal of this study is formulated using a Goal-Question-Metric approach [83]: We aim to ***analyze architecture decisions made during the development of quantum software*** *for the purpose of investigating their decision description and categorization, application domains covered, quality attributes considered, and the limitations and challenges* ***from the point of view of*** *practitioners* ***in the context of*** *architecture-related posts and GitHub issues of quantum software systems*. In order to perform this empirical study, we adhered to the recommendations for conducting empirical studies by Easterbrook et al. [84]. Following the defined goal, we provide five Research Questions (RQs), their rationale (see Table 1), and an overview of the research process (see Fig. 2) in the subsections below.

3.2. Data collection

We collected and analyzed data from 87 GitHub[5] projects and three Stack Exchange[6] sites. GitHub hosts a large number of open-source projects and offers access to rich data from those projects, including version history and issue tracking data [85]. Stack Exchange sites contain a number of technical Q&A sites that cover various topics and domains, which include Stack Overflow,[7] one of the most active Q&A sites that developers use from all over the world [86]. Quantum Computing Stack Exchange[8] site is a dedicated platform for topics related to quantum computing. Computer Science Stack Exchange[9] is specifically tailored to the field of computer science questions. Many researchers have used GitHub and Stack Exchange sites for quantum

software system research. For example, Li et al. [71] explored the challenges related to Quantum Software Engineering (QSE) by analyzing data from GitHub and Stack Exchange sites. Their research focused on understanding the broader challenges perceived by developers in the area of QSE. Additionally, Openja et al. [87] conducted an empirical analysis using GitHub data to explore the distribution and evolution of technical debt in quantum software, as well as their correlation with incidences of faults. Moreover, Zhang et al. [88] empirically analyzed architecture decisions by collecting data from GitHub and Stack Overflow in the development of AI-based systems. However, our focus is to analyze architecture decisions in quantum software systems. Therefore, we selected specific GitHub projects and Stack Exchange discussions based on their relevance to quantum software development to ensure a rich dataset.

(1) Data Collection from GitHub Projects

We followed a two-step process to collect quantum projects hosted on GitHub. First, we identified a set of quantum projects from GitHub. Then, we collect the issues from these projects.

In the first step, we performed a pilot search with several terms before we decided on the most appropriate terms for selecting relevant quantum projects on GitHub, namely "*quantum*" and "*quantum computing*". The search encompassed content within the projects (e.g., name, topics, descriptions, and README files). We found 74,432 projects by using the keyword "*quantum*". In contrast, using the keyword "*quantum computing*" resulted in significantly fewer projects (12,401), although with many irrelevant results such as books[10] and other learning resources.[11] Notably, "*quantum computing*" is also a subdomain of the keyword "*quantum*". During the search process by using the keyword "*quantum*", we also discovered that the retrieved projects contained the keyword "*quantum computing*". Therefore, we decided to use the keyword "*quantum*". To mine those projects, we used GitHub REST API.[12] We applied the following selection criteria: (1) projects that contain the word "*quantum*" (i.e., including the name, description, topics, and README file); (2) the number of stars ≥ 50; and (3) the number of forks is ≥ 15 to minimize the possibility of including student assignments following the idea of prior studies [89]. We identified 1226 projects that meet the above selection criteria. Later, we used the following criteria to filter out our search results [90]. We manually

---

[5] https://github.com.
[6] https://stackexchange.com/sites.
[7] https://stackoverflow.com/.
[8] https://quantumcomputing.stackexchange.com/.
[9] https://cs.stackexchange.com/.

[10] https://github.com/JackHidary/quantumcomputingbook.
[11] https://github.com/desireevl/awesome-quantum-computing.
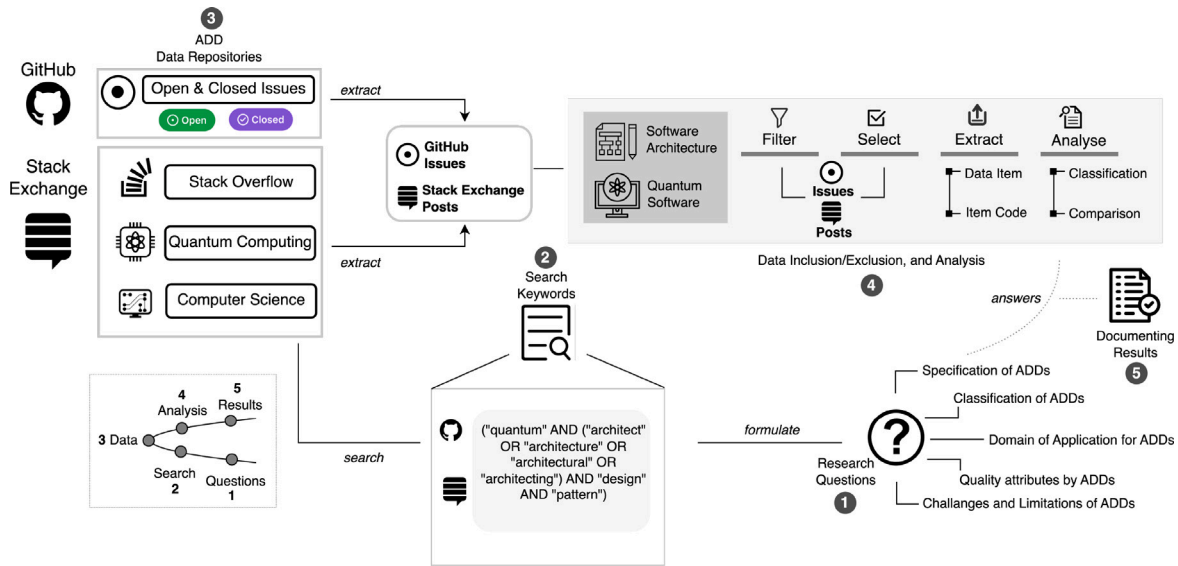[12] https://docs.github.com/en/rest?apiVersion=2022-11-28.

Fig. 2. Overview of the research process.

checked the repository (e.g., description, topics, and README file) and excluded those that pertained solely to quantum documentation or study materials and tutorials, not written in English, and repositories that are not related to quantum software systems but merely contain the word "*quantum*" in their descriptions (e.g., FirefoxColor[13]). In this step, 879 projects were removed, and 347 projects remained.

In the second step, we initially carried out a pilot search using various keywords that are the same as the architecture search terms "*architect**" (i.e., "*architect*", "*architecture*", "*architectural*", "*architecting*", "*architected*"), "*design**", and "*pattern**" and to search the title and body of the open and closed issues. We found issues that include the keywords ("*architect*", "*architecture*", "*architectural*", "*architecting*", "*architected*","*design*", and "*pattern*"). Therefore, we decided to use those keywords and searched the title and body of the open and closed issues of the 347 quantum projects. After searching with the search terms, we found 3185 open and closed issues containing the keywords, and those issues came from 192 quantum software projects. We ended up the data collection process from GitHub with 192 quantum software systems and 3185 open and closed issues.

(2) Data Collection from Stack Exchange Sites

To identify relevant posts in the selected Stack Exchange sites, we first conducted a pilot search to construct the search terms. The search terms were then applied to all parts of the posts, including the title, the question body, and the answers. Before we decided on the most suitable terms for capturing posts relevant to architecture of quantum software systems, we first performed a pilot search with several terms, namely "*quantum architect*" (i.e., "*quantum architect*", "*quantum architecture*", "*quantum architectural*", and "*quantum architecting*"), "*quantum design*", and "*quantum pattern*" within all three Stack Exchange sites. We found posts that include the keywords ("*quantum architect*", "*quantum design*", and "*quantum pattern*") using the original search terms. Second, Stack Exchange supports the use of wildcard (*) searches to broaden the search results. We defined "*quantum architect**", "*quantum design**", and "*quantum pattern**" as the initial search terms related to architecture decisions in quantum software. Table 3 shows the search terms that were utilized, and the number of posts that were obtained from all three sites. We retrieved a total of 1494 posts. Note that a post may contain multiple sets of keywords, which means that there could be duplicate items in the set of URLs of these posts. After removing the duplicated posts, we finally got 1424 unique posts. More specifically,

497 from Stack Overflow, 873 from Quantum Computing, and 54 from Computer Science Stack Exchange.

3.3. Data filtration

In our analysis of GitHub and Sack Exchange sites, we found many issues and posts containing the terms "*architecture*", "*design*" and "*pattern*" not only in the context of software architecture and design but also other contexts, including hardware (e.g., CPU architecture,[14] GPU architecture,[15] hardware design,[16] and hardware architecture[17]), when describing their concerns in the GitHub issues and Stack Exchange posts. We also found that both Stack Overflow and Computer Science Stack Exchange featured numerous posts with the term "*quantum*" not only in the context of a quantum software system but also in other contexts (e.g., Firefox Quantum,[18] quantum leap[19]). Therefore, we need to filter the retrieved 3185 open and closed issues from 192 quantum software systems and 1424 posts (497 from Stack Overflow, 873 from Quantum Computing, and 54 from Computer Science Stack Exchange) and exclude those posts that are not related to software architecture decisions in quantum software systems. To do so, we performed context analysis and applied our defined inclusion and exclusion criteria (see Table 4) to accurately filter out irrelevant issues and posts.

Before the formal filtering (manual inspection), we conducted a pilot filtering adhering to a set of inclusion and exclusion criteria (see Table 4) to identify GitHub issues and Stack Exchange posts related to architecture decisions in quantum software systems. Specifically, the pilot filtering process is composed of the following steps: (1) The first author randomly selected 10 issues from all the issues containing the keywords retrieved from the selected GitHub projects and 10 posts from the search results of Stack Exchange search terms; (2) The first and third authors manually checked independently whether the issues and posts should be included by following our defined criteria (see Table 4); (3) Data labeled by the two authors were compared, and the level of agreement between the two authors was calculated using the Cohen's Kappa coefficient [91]; (4) For any posts and issues that the two authors

---

[13] https://github.com/mozilla/FirefoxColor.

[14] https://github.com/m-labs/artiq/issues/1757.
[15] https://github.com/Qiskit/qiskit/issues/325.
[16] https://github.com/qiskit-community/qiskit-metal/issues/814.
[17] https://quantumcomputing.stackexchange.com/questions/16221.
[18] https://stackoverflow.com/questions/36007119.
[19] https://cs.stackexchange.com/questions/135237.

Table 2
GitHub projects, number of retrieved issues, and number of related issues.

#	GitHub project	Num of retrieved issues	Num of related issues	Stars	#	GitHub project	Num of retrieved issues	Num of related issues	Stars
GP1	Qiskit	285	40	4800	GP45	quantum-core	7	3	2200
GP2	Cirq	212	35	3900	GP46	qupulse	21	3	53
GP3	deepchem	148	23	4600	GP47	Strawberry Fields	7	3	735
GP4	ARTIQ	206	20	372	GP48	azure-quantum-python	6	2	119
GP5	qmcpack	109	17	245	GP49	deepqmc	3	2	300
GP6	PennyLane	88	14	1900	GP50	DFTK.jl	30	2	356
GP7	Qiskit Metal	122	13	250	GP51	ecosystem	12	2	70
GP8	qsharp	59	11	366	GP52	iqsharp	8	2	126
GP9	liboqs	63	8	1400	GP53	qBraid	8	2	62
GP10	QCoDeS	24	8	266	GP54	Qibo	20	2	271
GP11	PyQuil	31	7	1400	GP55	Qiskit Dynamics	4	2	96
GP12	Qiskit Nature	44	7	239	GP56	QRL	6	2	396
GP13	netket	39	5	450	GP57	QuEST	23	2	382
GP14	psi4	32	5	839	GP58	quisp	33	2	83
GP15	qsharp-language	24	5	233	GP59	tensorcircuit	10	2	250
GP16	QuTiP	30	5	1500	GP60	triqs	17	2	134
GP17	RMG-Py	34	5	329	GP61	Amazon Braket Python SDK	2	1	254
GP18	Covalent	48	4	500	GP62	Catalyst	8	1	101
GP19	cp2k	49	4	661	GP63	dftbplus	12	1	313
GP20	cuda-quantum	31	4	222	GP64	dwave-cloud-client	1	1	54
GP21	Microsoft Quantum	5	4	3900	GP65	dwave-hybrid	1	1	76
GP22	Mitiq	32	4	286	GP66	MQT DDSIM	1	1	124
GP23	MQT QMAP	7	4	88	GP67	pennylane-qiskit	2	1	175
GP24	QCFractal	10	4	143	GP68	ProjectQ	7	1	873
GP25	Qrack	29	4	154	GP69	Pulser	14	1	159
GP26	qsharp-runtime	21	4	280	GP70	QCEngine	2	1	141
GP27	Qualtran	25	4	129	GP71	qflex	3	1	97
GP28	QuantumLibraries	40	4	526	GP72	qiskit-aqua	19	1	564
GP29	Stim	14	4	303	GP73	qiskit-ibmq-provider	13	1	228
GP30	TensorFlow Quantum	125	4	3800	GP74	qiskit-ibm-runtime	21	1	95
GP31	tket	31	4	197	GP75	qiskit-ignis	18	1	167
GP32	circuit-knitting-toolbox	11	3	51	GP76	qiskit-optimization	5	1	215
GP33	dimod	14	3	120	GP77	qml	13	1	509
GP34	OpenQASM	18	3	1000	GP78	qpic	2	1	136
GP35	OpenQL	15	3	82	GP79	qsim	8	1	361
GP36	OpenSSL	16	3	172	GP80	quacc	7	1	155
GP37	OpenFermion	17	3	1500	GP81	QuantumKatas	20	1	4500
GP38	PQClean	16	3	423	GP82	QUICK	2	1	447
GP39	pyGSTi	31	3	131	GP83	Quilc	29	1	447
GP40	PyRPL	34	3	114	GP84	SIRIUS	2	1	103
GP41	PySCF	52	3	1200	GP85	Tangelo	3	1	98
GP42	QICK	17	3	175	GP86	toqito	12	1	129
GP43	qiskit-metapackage	29	3	3100	GP87	Yao.jl	14	1	907
GP44	qsharp-compiler	38	3	683					
	Total						2781	385	

Table 3
Stack exchange name, search term, retrieved posts, and related posts.

#	Stack exchange name	Search terms	Number of retrieved posts	Number of related posts
SEN1	Stack Overflow	*"quantum architect*"*	148	12
SEN2	Stack Overflow	*"quantum design*"*	260	20
SEN3	Stack Overflow	*"quantum pattern*"*	115	1
SEN3	Quantum Computing	*"quantum architect*"*	356	10
SEN4	Quantum Computing	*"quantum design*"*	455	13
SEN4	Quantum Computing	*"quantum pattern*"*	92	10
SEN5	Computer Science	*"quantum architect*"*	21	2
SEN6	Computer Science	*"quantum design*"*	39	1
SEN6	Computer Science	*"quantum pattern*"*	8	1
Total			1494	70

disagreed with, the two authors discussed them till an agreement was reached. The Cohen's Kappa coefficient before the discussion and resolution of the disagreements for the pilot data filtering was 0.80 for GitHub issues and 0.76 for Stack Exchange posts, both higher than 0.7. After resolving those disagreements through discussions, the two authors reached a 100% agreement, indicating a high degree of consistency between the two authors.

After the pilot data filtering, the first author carried on the formal data filtering with all the issues retrieved from GitHub projects and all the posts retrieved from Stack Exchange sites based on the inclusion and exclusion criteria. As it was easy to distinguish whether the posts and issues matched the criteria, the first author completed this step alone. There were only 9 issues and posts that the first author was unsure about, which were then discussed with the third author. After manually filtering all the candidates, we finally selected 385 related open and closed issues from 87 quantum software systems that related to architecture decisions in quantum software systems. These projects and their corresponding retrieved and related issues are detailed in

Table 4

Inclusion and exclusion criteria to manually identify Stack Exchange posts and GitHub issues related to software architecture decisions in quantum software systems.

Inclusion criterion
I1. We include a post or issue if it is related to both software architecture decisions [36,74] and quantum software systems [38]. For instance, if the content covers quantum software architecture patterns, design principles, structure, components, interactions, and frameworks in the development of quantum software systems, it will be included.

Exclusion criteria
E1. We exclude a post or issue if it is only related to quantum software systems but not related to software architecture decisions. For instance, if the content covers general principles of quantum computing, basic usage of quantum programming languages, quantum hardware, or quantum physics but does not provide specific architecture decisions.
E2. We exclude a post or issue if it is only related to software architecture decisions but not related to quantum software systems. For instance, if the content covers classical software architecture, design patterns, or development methodologies that do not involve quantum computing or quantum-specific components.

Table 2. Similarly, we retrieved 1494 Stack Exchange posts and identified 70 Stack Exchange posts that discussed architecture decisions in quantum software systems. This includes 33 posts from Stack Overflow, 33 posts from Quantum Computing, and 4 posts from Computer Science sites, as detailed in Table 3.

3.4. Data extraction and analysis

(1) Data Extraction

We conducted the data extraction process by identifying the relevant information to be extracted from the 385 GitHub issues and 70 Stack Exchange to answer our defined RQs (see Table 1). Table 5 outlines the data items extracted from the related posts and issues with the RQs addressed using the data items. Following this, we conducted both pilot and formal data extraction.

Before the formal data extraction, the first author conducted a pilot data extraction with the second author. The first author randomly selected 10 GitHub issues and 10 Stack Exchange posts. The first two authors independently extracted relevant information (i.e., Decision description, Decision type, Application domain, Quality attribute considered, and Limitation and challenge) as detailed in Table 5. In the event that disagreements arose, a third author was invited to discuss with the two authors and reached a 100% agreement. After the pilot data extraction, the first author extracted the data from the filtered posts and issues independently. To mitigate personal bias, the findings from this extraction were reviewed and validated by three additional authors of this study. During this process, if any disagreements appeared, we held a meeting and followed the negotiated agreement approach [92] to discuss and resolve any disagreement to ensure the accuracy and enhance the reliability of the extracted data regarding the uncertain portions.

(2) Data Analysis

To analyze our extracted data by following several existing studies (e.g., [88]), we employed a predefined classification [51] and the Constant Comparison method [52] to answer RQ1 and RQ2-RQ5 in our study. To address RQ1, we employed the predefined classifications of decision description contexts as described in [51,88]. The extracted data (i.e., decision descriptions) was then analyzed to investigate the linguistic patterns used in expressing architecture decisions. Linguistic patterns refer to grammatical rules that enable individuals to communicate effectively in a shared language [93]. Drawing inspiration from the study of Sorbo et al. [51], which categorized six linguistic patterns from development emails, we applied similar categorizations of linguistic patterns. Additionally, linguistic patterns have been employed in similar contexts, such as by Zhang et al. [88], to classify architecture decisions in AI-based systems. By applying a similar procedure (encoding and grouping of comparable codes into broader categories), we categorized linguistic patterns relevant to expressing architecture decisions to answer RQ1. Conversely, to address RQ2-RQ5, we conducted a qualitative analysis using the Constant Comparison method [52], which is a common technique in Grounded Theory (GT) [94]. GT adopts a bottom-up approach aimed at generating new theories rather than extending or verifying existing ones [94]. The constant comparison

involves an ongoing process to validate the emerging concepts and categories, which are refined and saturated until they accurately represent the data [52,94]. We used constant comparison to examine the different aspects of the data, including codes and categories, to unveil differences and similarities in the data [95].

We carried out a pilot data analysis for each RQ before the formal data analysis. The pilot data analysis encompassed several steps, as follows: (1) The first and second authors independently labeled the content of filtered posts and issues, cross-referencing them with the corresponding data items in Table 5; (2) The first author reviewed and verified the labeling results from the second author to ensure the accuracy of data extraction; (3) The first author aggregated all codes into higher-level concepts, subsequently converting them into distinct categories; (4) To ensure the rigor of our analysis, the other five authors (the third, fourth, fifth, sixth, and seventh authors) checked and validated the results from the pilot data analysis. The disagreements were resolved through meetings, utilizing a negotiated agreement method [92] to enhance the reliability of the pilot data analysis outcomes. In the next step, The first author carried on with the formal data analysis and followed similar steps used during the pilot data analysis. To mitigate personal bias and ensure the rigor of our analysis, all the authors checked and validated the results from the data analysis. During this process, if any disagreements appeared, the disagreements were fixed in a meeting utilizing the negotiated agreement technique [92] to improve the reliability of the analysis outcomes, resolving any discrepancies through collaborative discussions with the first author. A summary of the data analysis approaches is provided in Table 5. We have made all the data labeling and analysis results available in our replication package [53]. The subsequent paragraphs outline the specifics of the formal data analysis process:

(a) For analyzing RQ1

We adapted predefined classifications from [51,88] to address RQ1. The first author manually analyzed the extracted data for RQ1 (e.g., decision descriptions, see Table 5) from the related (e.g., quantum software architecture decisions) issues and posts to answering RQ1 (see Fig. 2). The data was then examined to investigate the linguistic patterns used in expressing architecture decisions in quantum software systems. By referring to the categories of linguistic patterns presented in the previous studies, six main categories were identified. To mitigate personal bias, the identified categories were validated by all the authors of this study. The disagreements were discussed and resolved in a meeting by using the negotiated agreement approach [92] to enhance the reliability of the results for RQ1.
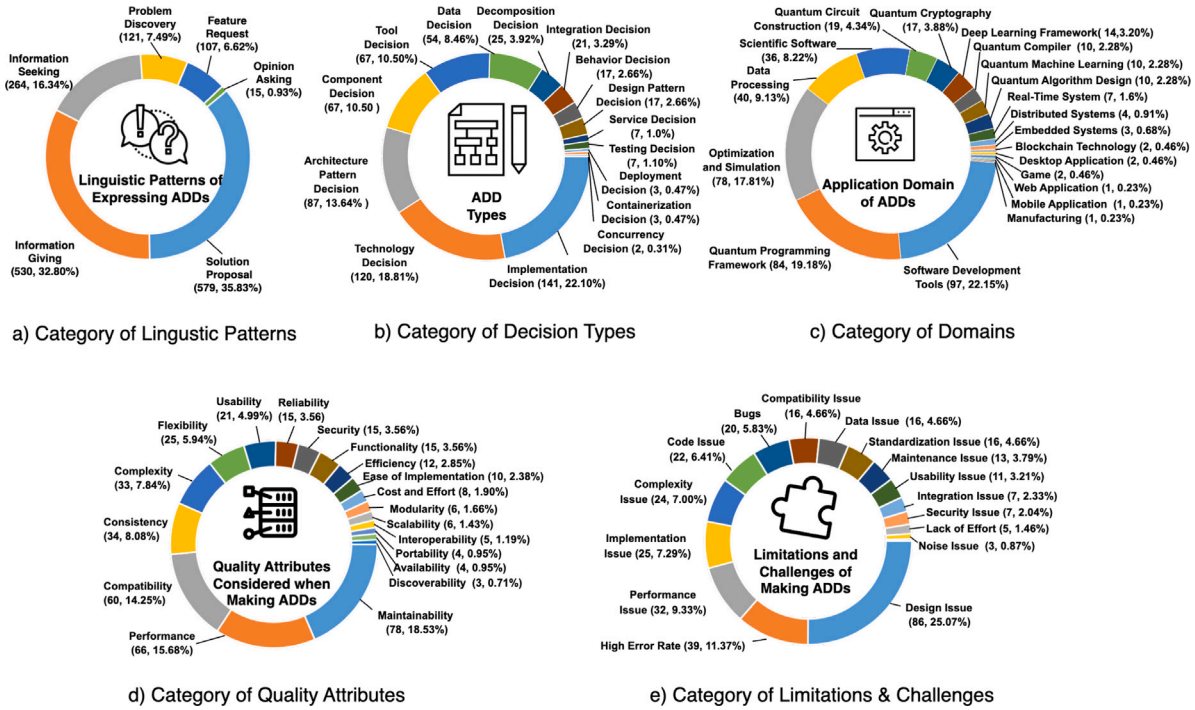
(b) For analyzing RQ2-RQ5

As abovementioned, we used constant comparison [52] to manually analyze the extracted data (i.e., the content of the issues and posts for the RQ2 decision type, the content of the issues and posts for the RQ3 application domain, the content of the issues and posts for RQ4 quality attribute considered, and the content of the issues and posts for RQ5 limitation and challenge) as shown in Table 5. With these RQs, we investigated architecture decisions related to issues and posts. Specifically, regarding the categorization of the questions, the first author studied the content of each post and issue related to quantum software architecture by exploring and identifying their main purposes (e.g., decision type, application domain, quality attribute, and limitation and

Table 5
Data items to be extracted with their description, analysis approaches, and relevant RQs.

#	Data item	Description	Data analysis approach	RQs
D1	Decision description	*Developers-based description of architecture decisions made in the development of quantum software systems*	Predefined classification [51]	RQ1
D2	Decision type	*A classification of architecture decisions taken during the development of quantum software systems*	Constant comparison	RQ2
D3	Application domain	*Application domains in which architecture decisions are made throughout the development of quantum software systems*	Constant comparison	RQ3
D4	Quality attribute considered	*Considered quality attributes when developers of quantum software systems make architecture decisions*	Constant comparison	RQ4
D5	Limitation and challenge	*The obstacles and difficulties posed by quantum software development when making architecture decisions*	Constant comparison	RQ5

a) Category of Linguistic Patterns b) Category of Decision Types c) Category of Domains

d) Category of Quality Attributes e) Category of Limitations & Challenges

Fig. 3. Linguistic patterns, architecture decisions, application domains, quality attributes, and limitations&challenges (results of RQ1 to RQ5).

challenge). The first author rigorously studied the data, summarized each finding into an easily understood sentence, and assigned a code. These categories, iteratively refined, encapsulated notable architecture decision types (RQ2), the different application domains where these decisions come into play (RQ3), the quality attributes developers considered during decision-making (RQ4), and the challenges faced during architecture decision-making (RQ5). Throughout the coding and categorization process, we embraced an iterative approach to enhance the accuracy of our categories. Subsequently, all the authors reviewed and validated the results of the data analysis, including the concepts and categories. Any disagreements were effectively resolved via the means of a negotiated agreement approach [92] in meetings to mitigate personal biases. As an output, we procured categories tied to each research question, representing the richness and complexity of the extracted data. The detailed elaboration of these categories for each RQ can be found in Section 4 and in Fig. 3.

4. Results

In this section, we present the results of the five RQs as outlined in Table 1. We first describe our methodology for analyzing instances drawn from GitHub issues and Stack Exchange discussions, and we then present the study results of each RQ. The results of the five RQs are visualized in Fig. 3.

4.1. RQ1. How do developers express architecture decisions made in quantum software development?

Our approach to answering RQ1 involves multiple stages. First, we collected 1616 instances of linguistic patterns (1406 from GitHub and 210 from Stack Exchange sites) of how practitioners articulate architecture decisions during the development of quantum software systems. One GitHub issue and one Stack Exchange post may identify multiple linguistic patterns because, within a single issue or post, practitioners might articulate their architecture decisions in various ways, using different phrases, terminologies, or contexts. To classify these decision expressions according to their intended objectives, we utilized a predefined classification scheme based on the linguistic patterns proposed by Sorbo et al. [51]. In a similar context, Zhang et al. [88] employed linguistic patterns to classify architecture decisions in the development of AI-based systems. Ultimately, we categorized the extracted architecture decision expressions into six types of linguistic patterns. Fig. 3a presents the six linguistic patterns and Table 6 contains these linguistic patterns as well as their examples and respective percentages.

Our analysis revealed that *Solution Proposal* and *Information Giving* were the two most common linguistic patterns, accounting for 35.83% and 32.80% of the instances, respectively. In contrast, *Opinion Asking* and *Problem Discovery* received relatively less attention from practitioners when conveying architecture decisions within the context

Table 6
Linguistic patterns of architecture decisions in quantum software systems.

Linguistic pattern	Example	Count	%
Solution Proposal	*One solution (to do it generically) that would involve a somewhat major architectural change is to implement a generic interface to asymmetric encryption then implement KEX in terms of that for the algorithms it applies to.* (liboqs issue #50)	579	35.83%
Information Giving	*Currently the hybrid Python/C++ architecture of Psi4 is in an odd spot where Psi4 itself is a C++ program that calls an input file as a Python executable. This circular process will be removed and Psi4 itself will become a Python library capable of being imported just like any other module.* (psi4 issue #468)	530	32.80%
Information Seeking	*After we have finished the decomposition and local Covalent has a microservice architecture, what should happen when a user does covalent start?* (Covalent issue #186)	264	16.34%
Problem Discovery	*These two software packages are very problematic: Conda is very poorly architected/designed.* (ARTIQ issue #1471)	121	7.49%
Feature Request	*Architectures supported in liboqs master branch should be updated* (liboqs issue #751)	107	6.62%
Opinion Asking	*Please feel free to suggest changes or alternate designs.* (deepchem issue #2059)	15	0.93%

of quantum software systems. A detailed overview of each of those categories is provided below:

(1) Solution Proposal: Our analysis shows that *Solution Proposal* is the most frequently used linguistic pattern. Those solutions are offered as pertinent information or assistance to fellow developers. This category often encompasses suggestions, recommendations, and best practices tailored to support and guide peers within the community. In our analysis, developers mentioned *Solution Proposal* in about 35.83% of the linguistic patterns we collected.

(2) Information Giving: This category relates to developers sharing insights, facts, or knowledge about architecture decisions. Developers might elucidate the nuances of a quantum algorithm, describe the interaction between classical and quantum components, or explain design choices made to optimize quantum coherence. This category embodies the proactive dissemination of architectural knowledge, ensuring that team members are aligned and informed about the architecture decisions made. In our analysis, developers mentioned *Information Giving* in about 32.80% of the linguistic patterns we collected.

(3) Information Seeking: *Information Seeking* revolves around developers looking to enhance product quality or gain clarity on specific functionalities within quantum software systems. This might manifest as questions, requests for clarification, or discussions aiming to understand architecture decisions better. In our analysis, developers mentioned *Information Seeking* in about 16.34% of the linguistic patterns we collected.

(4) Problem Discovery: In *Problem Discovery* linguistic pattern, developers identify, share, and discuss new problems, bugs, or issues they encounter in quantum software systems. It highlights the initial stages of problem-solving, where issues are first recognized and made known to the community. In our analysis, developers mentioned *Problem Discovery* in about 7.49% of the linguistic patterns we collected.

(5) Feature Request: When making decisions about the architecture design of quantum software systems, developers may ask for new features or enhancements to already existing libraries and software tools. These may also include recommendations to improve the functionality, usability, or flexibility of the systems. In our analysis, developers mentioned *Feature Request* in about 6.62% of the linguistic patterns we collected.

(6) Opinion Asking: *Opinion Asking* involves developers seeking the views, thoughts, or sentiments of peers concerning architecture decisions in quantum software systems. It may range from inquiring about the practicality of a proposed solution to gauging how the community feels about specific architectural decisions. In our analysis, developers mentioned *Opinion Asking* in about 0.93% of the linguistic patterns we collected.

> **Key Findings of RQ1**
>
> **Finding 1**: Developers represent architecture decisions with six categories of linguistic patterns, among which *Solution Proposal* (35.83%) and *Information Giving* (32.80%) are the two most frequently employed categories in quantum software development (see the examples in Table 6).

4.2. RQ2. What types of architecture decisions are made in quantum software development?

To answer RQ2, we conducted a comprehensive review of the content pertinent to architecture decisions in the context of quantum software systems, as gleaned from the accumulated 638 instances of architecture decisions (553 from GitHub and 85 from Stack Exchange sites). Note that a single post or issue often contains multiple architecture decisions because developers often explore various aspects of a problem, seeking information for comprehensive solutions or gathering diverse opinions. Additionally, the same architecture decision can be discussed multiple times within a post or issue due to its significance or as a common challenge faced by developers, leading to repetitive exploration for clarity or confirmation. Therefore, if a post or issue mentions multiple architecture decisions, each decision is counted. However, repeated mentions of the same architecture decision within a single post or issue were only counted once. We used the Constant Comparison method [52] to categorize the architecture decisions pertinent to the development of quantum software systems. Fig. 3b presents 15 types of architecture decisions and Table 7 provides an overview of these types of architecture decisions made in quantum software development, along with their respective quantities and percentages.

As shown in Table 7, the two common types of decisions are *Implementation Decision* (22.10%) and *Technology Decision* (18.81%). The remaining categories of architecture decisions account for a small percentage of the architecture decisions (less than 15%). We provide a detailed description of each of those categories below:

(1) Implementation Decision: Developers need to make informed decisions about implementing quantum algorithms and operations within their systems. These decisions often involve intricate technical considerations, such as selecting suitable programming languages and quantum computing platforms and optimizing code for quantum hardware. *Implementation Decisions* are found in 22.10% of the architecture decisions we collected.

(2) Technology Decision: Developers frequently find themselves at a crossroads, needing to decide which particular quantum technologies should be used due to the rapidly developing aspect of quantum technologies. These decisions have far-reaching implications for the overall system's efficiency and functionality, as selecting the right technology can significantly impact the system's performance. We found that

Table 7
Types of architecture decisions in quantum software development and their counts & percentages.

Decision type	Example	Count	%
Implementation Decision	*We designed and implemented an architectural framework that aims to standardize the interaction of an existing infrastructure with IBM Quantum, leveraging the most used open source technologies. (ecosystem issue #266)*	141	22.10%
Technology Decision	*Also, since the technology of partial measurement/non-demolition measurement is under development (I believe in some quantum architectures), may I expect to use this SDK to change when the technology is mature? (amazon-braket-sdk-python issue #158)*	120	18.81%
Architecture Pattern Decision	*If we are willing to provide extensibility (which sounds reasonable in a pluggable ecosystem), we should shift to a more suitable pattern such as the observer or pub-sub pattern (...).Again, the pub/sub or observer patterns are examples of decoupled architectures. (Qiskit #825)*	87	13.64%
Component Decision	*If we want to be strict about the architecture, the log forwarding component should be a controller. (artiq issue #691)*	67	10.50%
Tool Decision	*Unfortunately this is something coming from Amber build system. It compiles cuda source files for all possible architectures that a given toolkit supports. (QUICK issue #218)*	67	10.50%
Data Decision	*This seems like part of the discussion about long term architecture, harmonizing dc.data with tf.data and TensorGraph layers with Keras layers. (deepchem issue #1036)*	54	8.46%
Decomposition Decision	*After we have finished the decomposition and local Covalent has a microservice architecture, what should happen when a user does covalent start? (covalent issue #186)*	25	3.92%
Integration Decision	*I will try integrating quantum layers into existing CNN architectures (QC post #33215)*	21	3.29%
Behavior Decision	*I would propose not keeping the old behavior, simply to make the refactor easier and avoid code complexity (as well as design decisions). (PennyLane issue #226)*	17	2.66%
Design Pattern Decision	*Builder Pattern for defining control problems(...).The optimize_pulse API takes in a lot of optional arguments. Wouldn't it make more sense to use a builder pattern to define a control problem in a ControlProblem class? (qutip issue #692)*	17	2.66%
Service Decision	*The sync method in covalent/_results_manager/results_manager.py needs to be moved and updated to be compatible with the new covalent services architecture. (Covalent issue #370)*	7	1.10%
Testing Decision	*The assert for the third derivatives in test_einset.cpp fails due to what looks like some finite precision issue, but the issue can't be reproduced on other architectures. Unit test for thirdderivative components disabled until this issue is identified. (QMCPACK issue #1218)*	7	1.10%
Deployment decision	*QPIC is designed to be compatible from Python 2 up to the latest Python 3 (...). Is already fixed in this repo but not deployed to PyPI. Could you update the package in PyPI? (qpic issue #12)*	3	0.47%
Containerization decision	*I agree that there is a problem (smell) with the design. We are treating the amount of work as related to the size of the container when with padding this is not the case. (qmcpack issue #4290)*	3	0.47%
Concurrency decision	*The architecture implemented at the moment is showing some issues that are non trivial to fix (...).and then carefully design how we want to do concurrency and parallelism. (Qcodes issue #272)*	2	0.31%

Technology Decisions represent 18.81% of the architecture decisions we collected.

(3) Architecture Pattern Decision: *Architecture Pattern Decision* refers to the high-level organizational structure and modular design choices developers adopt when designing quantum software or integrating quantum components within larger systems. We identified the most common architecture patterns in quantum software systems as qubit gate patterns, layer patterns, composite patterns, decorator patterns, pipe and filter patterns, and transformer patterns. These could include modular quantum algorithm design, pipeline structures for quantum operations, or layered approaches separating quantum and classical components. Just as classical systems might leverage patterns like layers or service orientation, quantum systems require patterns optimized for quantum mechanics. These patterns shape the workflow, scalability, and maintainability of quantum systems. In a review of architecture decisions made in quantum software systems, *Architecture Pattern Decisions* were discovered in 13.64% of the architecture decisions we collected.

(4) Component Decision: Quantum software systems, like other software systems, benefit from modular and well-structured designs. Developers often need to decide how to organize and design system components, considering factors like quantum modules, quantum layers, and the integration of quantum and classical components. *Component Decision* is integral to achieving scalability, maintainability, and overall system reliability. *Component Decisions* were identified in 10.50% of our collected architecture decisions.

(5) Tool Decision: Quantum systems utilize a myriad of tools tailored for quantum-specific tasks. In the process of developing quantum software, developers commonly have to make an essential decision: choosing the right toolset or frameworks. This decision deeply influences the system's overall performance, fidelity in simulation, ease of debugging, and efficient resource orchestration. *Tool Decisions* were found in 10.50% of the architecture decisions we collected.

(6) Data Decision: *Data Decision* refers to how quantum data structures and quantum bits (qubits) are managed, accessed, and manipulated. Given the inherent differences between classical and quantum data, determining the right strategies for quantum data storage, retrieval, and processing is essential. This type of decision encompasses selecting appropriate quantum algorithms, choosing between quantum states, and handling superposition and entanglement. *Data Decisions* were discovered at 8.46% of the architecture decisions we collected.

(7) Decomposition Decision: *Decomposition Decision* pertains to the choices made when breaking down complex quantum algorithms or processes into simpler and more manageable components. Quantum algorithms, given their intricate nature, often need to be decomposed into a series of quantum gates or subroutines. The decision on how to decompose not only affects the efficiency of the algorithm but also its compatibility with specific quantum hardware. Such decisions may also involve deciding the granularity of modules or components, ensuring that the architecture remains modular, scalable, and maintainable. *Decomposition Decisions* were identified in 3.92% of the architecture decisions we collected.

(8) Integration Decision: *Integration Decision* refers to decisions made concerning how various software components, modules, or subsystems communicate and work together within quantum software systems. These decisions can involve the selection of quantum libraries, ensuring compatibility between classical and quantum sub-systems, or determining how quantum algorithms interface with other software functionalities. *Integration Decisions* were discovered at 3.29% of the architecture decisions we collected.

(9) Behavior Decision: *Behavior Decision* revolves around determining how quantum components or modules will interact and behave over time. Given that quantum software systems are inherently probabilistic and are susceptible to decoherence and external interference, *Behavior Decisions* often involve trade-offs between performance, accuracy, and fault tolerance. *Behavior Decisions* were discovered at 2.66% of the architecture decisions we collected.

(10) Design Pattern Decision: *Design Pattern Decision* in quantum software systems involves strategically selecting and applying design patterns to address common architecture challenges specific to quantum computing. These patterns help structure the software systems in a maintainable, scalable, and efficient way, addressing challenges such as state management, circuit construction, and operation optimization. By understanding and applying these well-known design patterns (e.g., visitor, builder, and recursive patterns) developers can ensure that their quantum software systems are built on robust architectural foundations. *Design Pattern Decisions* were discovered at 2.66% of the architecture decisions we collected.

(11) Service Decision: *Service Decision* pertains to the orchestration of quantum computational tasks within software frameworks. This involves decisions related to offering quantum functionalities as modular services or APIs. Considerations of *Service Decision* include the encapsulation of specific quantum algorithms, the design of interfaces for quantum service interaction, and the efficient management of quantum data structures. The focus is on the structural organization of quantum services within the software, ensuring modularity, scalability, and robustness while maintaining the peculiarities of quantum computation. *Service Decisions* were identified in 1.10% of the architecture decisions we collected.

(12) Testing Decision: *Testing Decision* in quantum software architecture refers to the process of selecting and designing appropriate testing strategies and practices to verify and validate the functionality, performance, and reliability of quantum software systems. This involves choosing the types of tests (unit tests, integration tests, etc.), identifying the scope of each test, and determining how to handle specific challenges related to quantum computing, such as finite precision issues, architecture-specific behaviors, and the infeasibility of exhaustive testing for large qubit systems. *Testing Decisions* were identified in 1.10% of the architecture decisions we collected.

(13) Deployment Decision: *Deployment Decision* encapsulates strategies and choices regarding how and where quantum computing applications will be executed and utilized. Further, deployment decisions extend to defining how quantum solutions are made available to end-users or other systems, ensuring that the deployment aligns with the computational, security, and performance requirements of quantum applications. *Deployment Decisions* were found in 0.47% of the architecture decisions we collected.

(14) Containerization Decision: *Containerization Decision* in a quantum software system involves the strategic use of containers to encapsulate and manage the software components and their dependencies. These decisions focus on ensuring the portability, scalability, and consistency of the quantum software across different environments. Containerization enables developers to package quantum applications with all necessary libraries and dependencies, ensuring that the application runs seamlessly on any platform that supports containers. *Containerization Decisions* were found in 0.47% of the architecture decisions we collected.

(15) Concurrency Decision: *Concurrency Decision* is concerned with managing multiple quantum operations or processes within software frameworks. *Concurrency Decisions* must consider the potential simultaneous execution of quantum tasks. This ensures that the quantum software efficiently utilizes available quantum resources while ensuring the integrity and fidelity of quantum computations. *Concurrency Decisions* were discovered at 0.31% of the architecture decisions we collected.

> **Key Findings of RQ2**
>
> **Finding 2**: *Implementation Decision* and *Technology Decision* are the most common decision types when making architecture decisions in quantum software development. Specifically, *Implementation Decisions* account for 22.10%, *Technology Decision* account for 18.81% of the architecture decisions we collected (see more details with examples in Table 7).

4.3. RQ3. What are the application domains of quantum software in which architecture decisions are made?

To answer RQ3, we analyzed the application domains where architecture decisions are made in the context of the development of quantum software systems. The finding of application domains is presented in Fig. 3c and summarized in Table 8, which contains 438 instances of application domains in total (373 from GitHub issues and 65 from Stack Exchange sites). During the data analysis process, we identified one application domain from each post and issue because our focus was on the primary application domain in which an architecture decision was made in quantum software systems, and most of the time the same domain was mentioned multiple times within a post or an issue, and we rarely found one post or issue that mentioned multiple domains where architecture decisions were made.

We identified 20 application domains, in which *Software Development Tools* are dominant, accounting for 22.15% of the application domains. *Quantum Programming Framework* represents 19.18% of the application domains. The remaining application domains account for less than 18% of the application domains. A detailed description of the top 9 application domains is provided below.

(1) Software Development Tool: Tools are pivotal for designing and building quantum software systems. These tools, ranging from Integrated Development Environments (IDEs) to debugging platforms, ensure efficient code development, testing, and deployment. In a quantum context, these tools are specialized to handle quantum-specific challenges, like qubit manipulation and entanglement. *Software Development Tool* was found at 22.15% of the application domains we collected.

(2) Quantum Programming Framework: The most common domain is revealed to be the *Quantum Programming Framework*. This domain holds substantial importance in quantum software systems, as it forms the foundation for the development of quantum algorithms and applications. It enables programmers to harness the power of quantum computation by providing the necessary tools, libraries, and abstractions, thus facilitating the creation of quantum software solutions. *Quantum Programming Framework* was found in 19.18% of the application domains we collected.

(3) Optimization and Simulation: optimization problems are central to quantum software systems, aiming to model and improve real-world quantum scenarios. By simulating quantum environments, developers can test algorithms or explore quantum behavior, making decisions about system robustness, efficiency, and accuracy. *Optimization and Simulation* instances were found at 17.81% of the application domains we collected.

(4) Data Processing: *Data Processing* emphasizes handling and analyzing quantum data, which can be vastly different from classical data. Architecture decisions here revolve around quantum data structures, efficient algorithms for quantum data manipulation, and ensuring data integrity. *Data Processing* was identified at 9.13% of the application domains we collected.

(5) Scientific Software: *Scientific Software* in quantum systems deals with applications tailored for research and complex computations. These tools are designed to understand and utilize quantum phenomena, aiding scientists in areas like quantum physics, chemistry,

Table 8

The domains of quantum software systems in which architecture decisions are made.

Domain	Count	%	Domain	Count	%
Software Development Tools	97	22.15%	Quantum Algorithm Design	10	2.28%
Quantum Programming Framework	84	19.18%	Real-Time System	7	1.6%
Optimization and Simulation	78	17.81%	Distributed Systems	4	0.91%
Data Processing	40	9.13%	Embedded Systems	3	0.68%
Scientific Software	36	8.22%	Blockchain Technology	2	0.46%
Quantum Circuit Construction	19	4.34%	Desktop Application	2	0.46%
Quantum Cryptography	17	3.88%	Game	2	0.46%
Deep Learning Framework	14	3.20%	Web Application	1	0.23%
Quantum Compiler	10	2.28%	Mobile Application	1	0.23%
Quantum Machine Learning	10	2.28%	Manufacturing	1	0.23%

and material science. The architectural decision here focuses on precision, scalability, and accuracy. *Scientific Software* was found at 8.22% of the application domains we collected.

(6) Quantum Circuit Construction: *Quantum Circuit Construction* in quantum software systems involves the systematic design, creation, and optimization of quantum circuits, leveraging architectural patterns and frameworks that facilitate hardware–software co-organization. This process integrates considerations of both the logical design of quantum algorithms and the physical constraints and capabilities of the quantum hardware, ensuring efficient and reliable execution of quantum computations. *Quantum Circuit Construction* was found at 4.34% of the application domains we collected.

(7) Quantum Cryptography: This domain is primarily concerned with utilizing the principles of quantum mechanics to secure communication against any potential interception. It involves techniques like Quantum Key Distribution (QKD), where cryptographic keys are securely distributed, and eavesdropping can be detected. The main architectural decisions in quantum cryptography revolve around ensuring the security of data during transfer. It is about securing communication channels in a way that is theoretically immune to any computational hacks, including those using quantum computers. *Quantum Cryptography* was found at 3.88% of the application domains we collected.

(8) Deep Learning Framework: *Deep Learning Framework* can help facilitate the fusion of quantum computing and neural networks. Such frameworks allow quantum systems to process vast amounts of data rapidly, making decisions about quantum neural architectures, training methods, and data-feeding mechanisms. *Deep Learning Framework* was found at 3.20% of the application domains we collected.

(9) Quantum Compiler: Those specialized compilers translate high-level quantum programming languages into machine-level instructions for quantum processors. Decisions in this domain concern optimization techniques, error corrections, and ensuring the efficient execution of quantum operations. *Quantum Compiler* was found at 2.28% of the application domains we collected.

> **Key Findings of RQ3**
>
> **Finding 3**: Our analysis of application domains of quantum software in which architecture decisions are made classified a total of 438 instances into 20 categories. *Software Development Tools* are dominant in these categories, accounting for 22.15% of the application domains we collected (see more details with examples in Table 8).

4.4. RQ4. What quality attributes are considered when developers make architecture decisions in quantum software development?

To answer RQ4, we extracted data concerning *quality attributes*. We collected 421 instances of quality attributes (376 from GitHub Issues and 45 from Stack Exchange posts). We discovered that a single post or issue mentioned multiple quality attributes when discussing architecture decisions because quantum software systems, being inherently complex, require developers to consider several quality attributes simultaneously to ensure robust and effective architecture decisions. If we identified the same type of quality attributes within the same post or issue, we considered them only once. Subsequently, we categorized these instances into 19 types of quality attributes, which are presented in Fig. 3d and detailed in Table 9.

Among these quality attributes, *Maintainability* appears as the most frequently considered by practitioners, constituting 18.53% of the total instances. Furthermore, *Performance* and *Compatibility* are accounting for 15.68% and 14.25%. Additionally, other quality attributes fall below the 10% mark, signifying that these quality attributes are the least considered by practitioners in their architecture decision-making of quantum software development. A detailed description of the top 7 quality attributes is provided below.

(1) Maintainability: *Maintainability* refers to the ease with which quantum software systems can be modified to correct faults, improve performance, or adapt to evolving requirements. This quality attribute is crucial given the rapid evolution of quantum technologies, where software must be structured in a way that accommodates frequent updates and enhancements. Achieving high maintainability requires clear, modular architecture and comprehensive documentation to facilitate future updates, testing, and debugging in the inherently complex quantum computing landscape. In our dataset, developers mentioned *Maintainability* quality attribute in about 18.53% of the quality attributes we collected.

(2) Performance: *Performance* plays a pivotal role as it directly influences the computational power and efficiency of quantum algorithms. Quantum computers are designed to excel in specific computational tasks, and optimizing performance ensures that these systems can execute complex quantum algorithms with minimal execution time, making them viable for practical applications such as cryptography, optimization, and material science simulations. In our dataset, developers mentioned *Performance* quality attribute in about 15.68% of the quality attributes we collected.

(3) Compatibility: *Compatibility* ensures that quantum software systems can interact with classical systems, enabling hybrid quantum–classical computing approaches. This quality attribute is crucial for practical quantum adoption, as quantum computers are often used in conjunction with classical systems for solving complex problems. In our dataset, developers mentioned *Compatibility* quality attribute in about 14.25% of the quality attributes we collected.

(4) Consistency: *Consistency* plays an important role in quantum software systems, ensuring that quantum computations produce reliable and reproducible results. Achieving consistent outcomes is essential in quantum computing, as it guarantees the correctness of quantum algorithms and maintains data integrity throughout complex quantum operations. Quantum systems must exhibit a high degree of internal and external consistency to meet the rigorous demands of quantum

Table 9

The quality attributes considered when making architecture decisions in quantum software development and their counts & percentages.

Quality attribute	Example	Count	%
Maintainability	*It would mean a more confused internal architecture, which would hurt us in maintainability in the long term.* (Qiskit issue #1852)	78	18.53%
Performance	*First the current design is intrinsically bad performance for hot loops because we have to look up the state (even if it doesn't change).* (DFTK.jl issue #313)	66	15.68%
Compatibility	*The "my foundational program can do many things, how do I box its capabilities where the QCArchive architecture views it as separate pieces".* (QCEngine issue #197)	60	14.25%
Consistency	*Ensure the API is as consistent as possible (both between PyTorch based models.* (deepchem issue #2059)	34	8.08%
Complexity	*In order to keep Psi4 up to date with best C++11 practices, reduce complexity (...). Currently the hybrid Python/C++ architecture of Psi4 is in an odd spot where Psi4 itself is a C++ program that calls an input file as a Python executable.* (psi4 issue #468)	33	7.84%
Flexibility	*I think RMG should be designed to not require changes to the core functionality, but have various inherited classes that allow for flexibility and robustness.* (RMG-Py issue #1136)	25	5.94%
Usability	*D-Wave Hybrid is a (Python) framework designed for creating classical-quantum hybrid workflows with an emphasis on ease of use.* (dwave-hybrid issue #245)	21	4.99%
Reliability	*MSYS also has straightforward benefits for users, such as speed and reliability.* (ARTIQ issue #1471)	15	3.56%
Security	*I think it is a very important project for the future of our security: being able to work with these new algorithms.* (OpenSSL issue #295)	15	3.56%
Functionality	*The Q# Formatter should have a design specification document, written in Markdown like a README, that describes the functionality of the tool as we want it to be and details our goals for the tool.* (qsharpcompiler issue #1074)	15	3.56%
Efficiency	*Samplers use Pauli frames to efficiently handle noisy circuits behind the scenes, but hide this information in the Python API.* (Stim issues #135)	12	2.85%
Ease of Implementation	*I think we all agree that caching is a good idea; the question is just how easy it is to implement in the current architecture.* (ARTIQ issue #1542)	10	2.38%
Cost and Effort	*Lots of money has been poured already into Conda (Continuum Analytics, NumFocus) and VS (Microsoft) with such a shoddy final result, so those tools seem like a dead-end to me.* (ARTIQ issue #1471)	8	1.90%
Modularity	*We have a good modularity in the compiler, specifically with respect to the distinctions above.* (cuda-quantum issue #85)	6	1.66%
Scalability	*Using db will solve the scalability issue and will be able to handle a large number of request.* (QRL issue #1545)	6	1.43%
Interoperability	*We also get better interoperability with the broader python ML ecosystem.* (deepchem issue #/1036)	5	1.19%
Portability	*dc.data is one of the hackier parts of DeepChem and has caused us a good number of scaling and portability headaches.* (deepchem issue #1039)	4	0.95%
Availability	*Availability of QML tools such as optimizers and different encoding schemes" and "availability of simulators and circuit visualizers.* (QC post #33215)	4	0.95%
Discoverability	*In addition to the general Q# API Design Principles, the changes detailed in this proposal follow a few principles specific to arithmetic representation to help ensure a consistent and discoverable user experience for developers working with the Quantum Development Kit.* (QuantumLibraries issue #337)	3	0.71%

computing tasks. In our dataset, developers mentioned *Consistency* quality attribute in about 8.08% of the quality attributes we collected.

(5) Complexity: *Complexity* in quantum software systems refers to the intricacy and difficulty in ensuring scalability, efficiency, and robust quantum software design. Balancing these factors demands a profound understanding of both quantum mechanics and software structures, emphasizing the need for advanced tools and methodologies to navigate the technology landscape. In our dataset, developers mentioned *Complexity* quality attribute in about 7.84% of the quality attributes we collected.

(6) Flexibility: *Flexibility* in quantum software systems ensures that the systems can adapt to varied computational tasks, accommodating different quantum algorithms and protocols. It is vital for catering to a broad spectrum of quantum applications. In our dataset, developers mentioned *Flexibility* quality attribute in about 5.94% of the quality attributes we collected.

(7) Usability: *Usability* is a key consideration in quantum software systems to facilitate the effective utilization of quantum resources by developers and end-users. User-friendly interfaces, tools, and documentation are essential for enabling practitioners to harness the power of quantum computing without requiring an in-depth understanding of quantum mechanics. In our dataset, developers mentioned *Usability* quality attribute in about 4.99% of the quality attributes we collected.

Key Findings of RQ4

Finding 4:
We identified 19 categories of quality attributes that are considered by developers when making architectural decisions in quantum software systems, with *Maintainability* (18.53%), *Performance* (15.68%), and *Compatibility* (14.25%) being the most considered attributes by developers (see more details with examples in Table 9).

4.5. RQ5. What are the limitations and challenges of making architecture decisions in quantum software development?

To answer RQ5, we collected and analyzed a total of 343 instances of limitations and challenges (320 from GitHub issues and 41 from Stack Exchange posts). During this process, we found that one GitHub issue and Stack Exchange post may discuss multiple limitations and challenges related to architecture decision-making in quantum software development. If we found the same limitations and challenges within the same post or issue, we counted them only once. We finally identified 16 types of limitations and challenges that developers encounter when making architectural decisions in quantum software development, as presented in Fig. 3e and detailed in Table 10.

Table 10

Limitations and challenges of making architecture decisions in quantum software development, their counts & percentages.

Limitations & Challenges	Example	Count	%
Design Issue	*A CurveManager SoftwareModule replacing pyinstruments would definitely fit very nicely in this modular architecture. In this case, there are a lot of design mistakes that we could avoid with our experience of pyinstruments.* (PyRPL issue #84)	86	25.07%
High Error Rates	*The build systems has also only seen compilation on half a dozen architectures and setups, you may encounter errors in the build process.* (psi4 #468)	39	11.37%
Performance Issue	*The architecture implemented at the moment is showing some issues that are non trivial to fix: pickling, monitoring, performance, in general seems to hard to grasp for the users intermittent test failures due to syncing issues.* (QCoDeS issue #272)	32	9.33%
Implementation Issue	*Currently the API for KEX makes it a bit difficult to implement a KEM transform like Fujisaki-Okamoto generically. One solution (to do it generically) that would involve a somewhat major architectural change is to implement a generic interface.* (liboqs issue #50)	25	7.29%
Complexity Issue	*While this architecture is fine for the first ARTIQ systems that were rather small and simple and even for single-crate Metlino/Sayma systems, it shows limitations on more complex ones.* (ARTIQ issue #778)	24	7.00%
Code Issue	*Just reproducing the first issue with code that is fully executable: (..). I think the issue here is purely to do with labelling. When specifying subsystem_list = [1], DynamicsBackend is only aware that it has a system labelled by index 1 (and it will internally automatically build some measurement definitions based on this labelling).* (qiskit-dynamics issue #235)	22	6.41%
Bugs	*Bugs: there are a variety of bugs in the current compiler which have been known about for a long time.* (ARTIQ issue #1542)	20	5.83%
Compatibility Issue	*At the same time, we need to plan a transition that doesn't entirely break backwards compatibility.* (deepchem issue #1039)	16	4.66%
Data Issue	*Problem is my data consists of 2 arrays consisting of Double numbers representing the real and imaginary parts of complex numbers.* (SO post #55763054)	16	4.66%
Standardization Issue	*But until/unless NIST gives a standardized version of that, we plan to stick with just the KEM formulation.* (qmcpack issue #649)	16	4.66%
Maintenance Issue	*Supporting it would mean a more confused internal architecture, which would hurt us in maintainability in the long term;* (Qiskit issue #1852)	13	3.79%
Usability Issue	*To be honest the BackendConfiguration class is not the most ergonomic to use.* (QC post #28416)	11	3.21%
Integration Issue	*Integration with Keras: Although they're very similar, TensorGraph layers have certain features that it isn't obvious to me how to reproduce with Keras layers.* (deepchem issue #1038)	8	2.33%
Security Issue	*The pseudorandom number generator used by random.random(), Mersenne Twister, is not suitable for cryptography or information security.* (SO post #63609099)	7	2.04%
Lack of Effort	*A new interface would provide the most benefit here, but it would also require the most effort, both in development and long-term maintenance.* (qsim issue #293)	5	1.46%
Noise Issue	*This feature will enhance the current PyQuil simulator's capabilities (noise.py module) by providing a more accurate representation of noise.* (pyquil issue #1575)	3	0.87%

We identified two main types of limitations and challenges: *Design Issues* and *High Error Rates* which practitioners encountered in 25.07% and 11.37% of the limitations and challenges we collected. Other types were mentioned in less than 10% of the limitations and challenges we collected. A detailed description of the top 10 limitations and challenges is provided below.

(1) Design Issues: *Design Issues* are key challenges that developers encounter most when making architecture decisions in quantum software development. These challenges arise when developing quantum algorithms, optimizing quantum circuits, and seamlessly integrating them with components. Decision-making on design issues is particularly challenging due to the evolving standards and tools in quantum computing, which require learning and adaptation. To address *Design Issues*, it is important for developers to first learn about quantum principles and use quantum algorithms. Developers also need to consider the specific hardware and software stack being used. In our dataset, developers discussed *Design Issues* in about 25.07% of the limitations and challenges we collected.

(2) High Error Rates: *High Error Rates* in quantum software architecture refer to the frequent occurrence of errors during the execution of quantum algorithms, compilation, or deployment of quantum software systems. High error rates in quantum software architecture are a significant challenge arising from various sources, including asynchronous operations, build and compilation issues, inadequate error mitigation techniques, and incorrect configurations. Addressing these errors involves improving synchronization, enhancing error mitigation

techniques, ensuring cross-platform compatibility, correctly implementing quantum algorithms, managing configurations properly, and conducting thorough testing. Developers can reduce error rates and build more robust quantum software systems by focusing on these areas. In our dataset, developers discussed *High Error Rates* in about 11.37% of the limitations and challenges we collected.

(3) Performance Issues: *Performance Issues* in quantum software arise when applications do not fully leverage quantum speedups, often due to inefficient algorithms or sub-optimal resource utilization. Developers face these challenges when making architectural decisions because identifying and using quantum advantages requires a deep understanding of both quantum mechanics and the problem domain. To prevent *performance Issues*, developers need to focus on algorithm optimization specific to quantum processing, ensuring that quantum resources are effectively utilized to offer a true computational advantage. In our dataset, developers discussed *Performance Issues* in about 9.33% of the limitations and challenges we collected.

(4) Implementation Issues: *Implementation Issues* in quantum software development primarily stem from the intricate nature of quantum algorithms and the complexities inherent to nascent technologies, as illustrated by challenges in integrating quantum algorithms with existing classical computing systems. Specifically, developers often encounter difficulties when trying to generalize functions or leverage advanced features due to hardware–software co-optimization, and practical and robust frameworks that can seamlessly integrate with various quantum algorithms and protocols. These challenges are magnified by the high complexity of quantum computations, nuances of post-quantum

cryptography, and the infancy of quantum programming languages. To prevent these issues, developers should focus on fostering adaptive software designs, ensuring rigorous testing and validation of newer features, deepening collaboration between quantum software communities, and enhancing the flexibility and documentation of quantum development tools and APIs. In our dataset, developers discussed *Implementation Issues* in about 7.29% of the limitations and challenges we collected.

(5) Complexity Issues: Quantum software inherently presents *Complexity Issues* due to the multi-faceted nature of quantum mechanics. Developers wrestle with this type of issue when making architecture decisions, as understanding and managing the intricacies of superposition and entanglement can be daunting. This complexity makes designing efficient quantum algorithms challenging. To alleviate *Complexity Issues*, a deep dive into quantum principles, alongside leveraging abstraction layers and tools that simplify quantum processes, is indispensable. In our dataset, developers discussed *Complexity Issues* in about 7.00% of the limitations and challenges we collected.

(6) Code Issues: *Code Issues* in quantum software involve problems like unreadable code, lack of proper documentation, and non-reusability, impeding the functionality or efficiency of applications. This type of issue is prevalent due to the developing stage of quantum programming languages and the complex nature of quantum computations. To mitigate code issues, developers should adhere to best coding practices, including thorough documentation, code reviews, and leveraging classical coding standards where applicable. In our dataset, developers discussed *Code Issues* in about 6.41% of the limitations and challenges we collected.

(7) Bugs: In quantum software systems, *Bugs* refer to inconsistencies, or unintended behaviors within the software that hinder its correct or efficient operation. *Bugs* can arise from a multitude of sources, including long-standing compiler issues, incorrect dataset creation, and improper support for certain measurements due to runtime updates. Developers can address these issues by conducting thorough testing, profiling for resource management, and ensuring backward compatibility with runtime updates. Addressing *Bugs* also involves refining unit tests to reflect actual usage patterns, consolidating coding styles, and improving error messages to reduce user confusion. Regular profiling and debugging, especially for resource-heavy operations, are crucial in identifying and mitigating these bugs, ultimately enhancing the reliability and user experience of quantum software systems. *Bugs* accounts for 5.83% in our analysis of quantum software systems.

(8) Compatibility Issues: *Compatibility Issues* in quantum software development are primarily related to the challenges of creating compatible interfaces between quantum and classical systems. These difficulties are compounded by the absence of universally accepted quantum programming frameworks and the significant obstacles in integrating quantum technology with existing IT infrastructures. Together, these factors pose substantial challenges to the advancement and application of quantum computing in current technological environments. This incompatibility extends to difficulties in data formats, communication protocols, and system interoperations, which underscore an essential need for enhanced functionalities across different layers of the software stack as well as improved interfaces between quantum and classical computing environments. In our dataset, developers discussed *Compatibility Issues* in about 4.66% of the limitations and challenges we collected.

(9) Data Issues: *Data Issues* refer to challenges in data encoding, storage, and retrieval using quantum bits in quantum computing. Developers encounter this type of issue when making architecture decisions because the intricate process of encoding classical information into qubits is influenced by the nature and encoding type of data, which can significantly impact the performance of algorithms. To counteract *Data Issues*, developers should choose the appropriate qubit data encoding patterns, understand quantum data operations, and utilize appropriate quantum data structures. In our dataset, developers

discussed *Data Issues* in about 4.66% of the limitations and challenges we collected.

(10) Standardization Issues: *Standardization Issues* affect quantum software development, as the field suffers from a lack of standardized tools and frameworks. Encouraging collaborative efforts towards standardizing quantum computing protocols and interfaces, and contributing to community-driven quantum software libraries can help establish much-needed uniformity. In our dataset, developers discussed *Standardization Issues* in about 4.66% of the limitations and challenges we collected.

> **Key Findings of RQ5**
>
> **Finding 5**:
> *Design Issues* (25.07%) and *High Error Rates* (11.37%) are the most common challenges in architecture decision-making in the development of quantum software systems. Other notable issues include *Performance Issues* (9.33%), *Implementation Issues* (7.29%), *Complexity Issues* (7.00%) and *Code Issues* (6.12%) (see the examples in Table 10).

5. Discussion

In this section, we revisit the findings of this study by interpreting the results in Section 5.1 and discussing their implications for researchers (indicated with the 📖 icon) and/or practitioners (the 👤 icon) in Section 5.2.

5.1. Interpretation of results

5.1.1. RQ1: How do developers express architecture decisions made in quantum software development?

Table 6 presents the linguistic patterns identified in the architecture decisions made during the development of quantum software systems. These patterns serve as a means to understand the diverse intentions behind decision expressions, shedding light on the reasoning process that drives the architects and developers of quantum software systems. Among these linguistic patterns, two arise as particularly dominant: *Solution Proposal* and *Information Giving*. *Solution Proposal* is crucial for suggesting multiple solutions to architectural challenges in quantum technologies. Its frequent use underscores its importance in enriching decision-making processes by exploring alternative pathways. *Information Giving* plays a critical role in quantum software development by conveying essential context and insights necessary for informed decision-making. It facilitates effective communication among architects and developers, ensuring a comprehensive understanding of technical details. While Solution Proposal and Information Giving are prominent, *Feature Request*, *Problem Discovery*, and *Opinion Asking* are less frequently utilized in expressing architecture decisions in quantum software development. These patterns offer potential avenues for enhancing collaborative decision-making but require further exploration within this context.

5.1.2. RQ2. What types of architecture decisions are made in quantum software development?

Our investigation indicates the presence of 15 types of architecture decisions in the context of quantum software development, as shown in Table 7. The prominent *Implementation Decision* underscores the necessity for detailed and precise implementation guidelines in quantum systems. This emphasizes the importance of thoroughly understanding how to integrate and utilize quantum components, given their complexity and the current nascent stage of quantum technology. Similarly, *Technology Decision* is critical due to the rapid evolution of quantum technologies. Practitioners must navigate a rapidly changing landscape

to select appropriate technologies that align with their specific system requirements. For instance, selecting a quantum programming framework or backend configuration can significantly impact system performance and maintainability, as illustrated by the ongoing developments in tools like Qiskit and OpenQL. *Architecture Pattern Decision* guides the selection of suitable architectural patterns to meet specific quality attributes like maintainability and performance. This decision is crucial for ensuring that quantum systems can adapt and evolve over time, accommodating new features and optimizations without significant rework. *Component Decision*, which emphasizes modular and layered architectures, facilitates maintainability and scalability, enabling developers to manage complex systems more effectively. *Tool Decision* and *Data Decision* highlight the importance of selecting optimal tools and managing data efficiently. Given the significant and dynamic nature of data generated during quantum computing tasks, these decisions are pivotal for ensuring robust and efficient data handling and processing. Other architecture decisions within quantum systems encompass *Decomposition Decision.Integration Decision* focuses on melding quantum components with traditional systems for smooth data and process interactions. *Behavior Decision*, *Design Pattern Decision*, *Service Decision*, *Testing Decision*, *Deployment Decision*, *Containerization Decision* and *Concurrency Decision*. Collectively, these decisions underscore the multifaceted considerations that architects and developers need to address while designing quantum software systems.

5.1.3. RQ3. What are the application domains of quantum software in which architecture decisions are made?

As illustrated in Table 8, architecture decisions in quantum software development are made in a total of 20 application domains, in which *Software Development Tools* are dominant. The prominence of *Software Development Tools* is attributed to their critical role in accessing and managing quantum-specific modules and functionalities required for quantum software systems. These tools facilitate the development, integration, and optimization of quantum algorithms and applications, leveraging frameworks that bridge classical and quantum computing paradigms. Quantum-specific software development environments, such as those provided by Qiskit, Cirq, and PennyLane, enable developers to implement quantum algorithms, manage quantum states, and interface with quantum hardware effectively. Beyond *Software Development Tools*, *Quantum Programming Framework* plays a pivotal role in quantum software architecture decisions, particularly in defining the foundational structures and methodologies for quantum algorithm implementation. These frameworks encompass libraries, compilers, and runtime environments tailored to quantum computing requirements, influencing architecture decisions around algorithm design, optimization techniques, and hardware compatibility. *Scientific Software* harnessing the power of quantum mechanics, and *Optimization and Simulation* solutions that leverage principles of quantum mechanics. Quantum phenomena also play a pivotal role in domains such as *Quantum Cryptography*, offering unmatched security features. Likewise, the synergy between quantum mechanics and artificial intelligence is evident in the *Deep Learning Framework* domain, where quantum-enhanced neural networks are under exploration. Quantum technology is causing significant changes (e.g., optimized quantum algorithms, enhanced computational power, and increased data processing capabilities) in domains like *Quantum Compiler*, *Real-Time System*, *Distributed System*, and *Embedded System*. Moreover, *Blockchain Technology* is witnessing innovative approaches influenced by quantum mechanics, ensuring robust and tamper-proof systems. Traditional *Desktop Applications*, *Web Applications*, and *Mobile Applications* are also experiencing a transformative phase, with quantum algorithms enhancing their efficiency and capabilities. Finally, *Games* in quantum software systems are still in their nascent stages, but hold the promise of offering incomparable gaming experiences, tapping into the probabilistic nature of quantum mechanics.

5.1.4. RQ4. What quality attributes are considered when developers make architecture decisions in quantum software development?

As shown in Table 9, we identified 19 types of quality attributes considered by architects when they make architecture decisions in quantum software development. In our study, two prominent quality attributes, *Maintainability* and *Performance* were mostly discussed in quantum software development. *Maintainability* was mentioned 78 times, constituting over 18% of the identified quality attributes, while *Performance* was found 66 times, accounting for approximately 15% of the total quality attributes. *Maintainability* involves ensuring that the system remains easy to manage and evolves over time. *Maintainability* plays an essential role in quantum software systems since it ensures that quantum software systems can evolve as technology advances or requirements change. As one developer proposed "*I think the node should be kept minimal and specific for a lot of good architectural reasons. (…) More non-core features, more on/off settings => more complexity, less maintainability, higher risk*" (QRL issue #1545). *Performance* is another vital quality attribute for quantum software systems, since *Performance* is crucial for long-term viability and adaptability. *Performance* is primarily associated with time behavior, resource utilization, and system capacity. Practitioners in quantum software development are particularly concerned about factors such as training and computing time, as well as the speed of data processing. These aspects are crucial for optimizing the performance of quantum software systems, and suggestions include the adoption of better-distributed architectures to enhance these performance aspects [96], as mentioned in a GitHub issue "*First the current design is intrinsically bad performance for hot loops because we have to look up the state (even if it doesn't change)*" (DFTK.jl issue #313).

5.1.5. RQ5. What are the limitations and challenges of making architecture decisions in quantum software development?

As shown in Table 10, developers encounter 16 types of limitations and challenges during architecture decision-making of quantum software development, in which *High Error Rates* and *Design Issues* are the main types. *Design Issues* frequently arise in quantum software development due to e.g., complex quantum technology, quantum software development tools, and quantum programming frameworks. Quantum software not only requires a redesign of applications, but also mandates a reconceptualization of the tools and languages used for quantum software development. Furthermore, from a coding perspective, there are hurdles to overcome to maintain clarity and scalability in the quantum context, as highlighted in an issue that states, "*This issue is meant to highlight some design issues from the coding point of view that will make life hard for current and future coders of RMG-Py*" (RMG-Py issue #521). *High Error Rates*, on the other hand, are tightly intertwined with the intrinsic characteristics of quantum systems. Given the delicate nature of quantum states and the probabilistic outcomes they produce, ensuring consistent and efficient performance is a challenge. For instance, a developer noted: "*The current design is an intrinsically bad performance for hot loops because we have to look up the state (even if it doesn't change)*" (DFTK.jl issue #313). This example touches upon the nuances of state management in quantum software systems, where even seemingly minor operations can entail significant overheads. In another context, as quantum algorithms are integrated into larger systems, unpredictable performance hiccups can occur, as evidenced by the feedback from one developer that "*But the performance got slightly worse (higher variance) or the model crashed due to NaNs*" (deepqmc issue #25).

5.2. Implications

5.2.1. Linguistic patterns

Identifying linguistic patterns helps researchers understand architecture decisions decision description, while practitioners can use these insights to improve architecture communication in quantum software projects: The identification of linguistic patterns

(e.g., *Solution Proposal, Information Giving, Information Seeking*, and less commonly used *Feature Request, Problem Discovery*, and *Opinion Asking*) provides a structured framework for understanding decision-making processes in quantum software architecture. 📖 For researchers, linguistic patterns enhance the study of decision-making dynamics, offering insights that can inform the development of improved decision-support tools and methodologies tailored to quantum software architecture. By analyzing these linguistic patterns, researchers gain deeper insights into how architectural decisions are formulated, communicated, and evolved within quantum computing environments. This foundational knowledge can lead to theoretical advancements in decision-making models, contributing significantly to the broader field of software engineering and quantum computing research. 👤 For industry practitioners, understanding and leveraging these patterns fosters better collaboration and communication on architecture among teams in quantum software development projects. The dominance of *Solution Proposal* and *Information Giving* underscores the importance of fostering flexible and communicative decision-making frameworks within industry settings. 👤 Practitioners can utilize these insights to streamline decision processes, improve the quality of architectural solutions, and effectively manage project complexities. *Information Seeking* though less frequent, plays a critical role in seeking clarifications and resolving technical uncertainties, which is essential for optimizing decision-making efficiency. 👤 Practically, integrating features into architecture tools and platforms that support *Solution Proposal, Information Giving*, and *Information Seeking* can lead to the development of advanced Integrated Development Environments (IDEs) and collaborative platforms tailored to the needs of quantum software practitioners.

***Feature Request, Problem Discovery*, and *Opinion Asking* can be considered by architects and developers in describing their architecture decisions in quantum software development:** To advance quantum software architecture effectively, *Feature Request, Problem Discovery*, and *Opinion Asking* were found to be less frequently used for expressing architecture decisions during the development of quantum software. *Feature Request* typically outlines a new functionality or enhancement being added to the system. The given example, *"Pull requests implementing new features: Datalayer type creation, conversion and dispatch #1338 implementing the data-layer creation, conversion and dispatcher routines"* (QUTIP issues #1278), specifies a technical enhancement, indicating that pull requests have been made for new data layer functionalities, which are crucial for the data handling capabilities of software systems. Furthermore, *Problem Discovery* is about identifying and describing issues or limitations within the current architecture. The example *"SIKE not properly disabled on unsupported architectures"* (liboqs issue #1128) highlights a specific issue where a cryptography algorithm SIKE is not being correctly disabled in environments where SIKE is not supported, potentially leading to errors or security vulnerabilities. Lastly, *Opinion Asking* is a linguistic pattern where developers seek feedback or alternative perspectives on an architecture decision, promoting collaborative decision-making. The example *"Please feel free to suggest changes or alternate designs"* (deepchem issue #2059) opens the floor for community input on the project, which can lead to innovative solutions and shared ownership of the project's evolution. 📖 Researchers and 👤 practitioners are encouraged to integrate these patterns into their decision-making processes to enhance the agility, robustness, and innovation of quantum software architectures.

5.2.2. Architecture decisions

***Implementation Decisions* and *Technology Decisions* are critical architecture decisions, requiring deep understanding and strategic planning in quantum software development:** The identified decisions directly influence the structure and functionality of quantum software architectures. Prominent among these decisions are *Implementation Decision* and *Technology Decision*, the complexity of which is underlined by the need to manage qubit. The intricacies of these

decisions require a deep understanding of quantum components and their integration, contributing to more mature and robust architecture designs. These decisions do not merely guide the development process; they also ensure that the architecture can handle the unique requirements of quantum computing. For instance, the architecture decision-making process becomes intricately complex. 📖 Practitioners often find themselves at a crossroad, deliberating on the most suitable means to incorporate quantum-related components. Such as, determining the feasibility of introducing a feature, like caching, within a given architecture poses questions related to effort, compatibility, and user comprehension, as evidenced in this example *"The C++ API of LLVM is complicated and has poor forward compatibility (...). I think we all agree that caching is a good idea; the question is just how easy it is to implement in the current architecture (in terms of implementation effort, as well as ease of understanding for users)"* (ARTIQ issue #1542). On the other hand, *Technology Decision* delves into the selection and integration of appropriate technologies that cater to the needs of quantum software systems. 📖 The rapid evolution of quantum technologies and tools bewilders many practitioners. A case in point is the quandary expressed by a developer, *"What's the latest in Python LLVM bindings anyway is there one that supports the (C++) IRBuilder interface?"* (ARTIQ issue #1370), where the binding specifics for a language (Python) with a compiler (LLVM) are in question.

Researchers should innovate methodologies to address challenges, while practitioners need modular, layered approaches to handle evolving quantum technology: For the research community, these findings highlight key areas that require further exploration and innovation. 📖 Researchers can focus on developing new methodologies and frameworks to address the identified challenges, such as creating more effective architecture patterns and optimization techniques for quantum software systems. This can lead to significant advancements in the field of quantum software engineering. Therefore, we echo the need for 📖 more empirical studies to fully understand and quantify the benefits of these insights for quantum software architecture. 👤 For industrial practitioners, *Implementation Decisions* and *Technology Decisions* underscore the importance of strategic planning and continuous learning. The findings also impact design practices, encouraging practitioners to adopt a modular and layered approach to architecture design. This approach enhances the flexibility and adaptability of quantum software systems, allowing for easier updates and integration of new technologies. 👤 Practitioners were also faced with significant challenges when making *Implementation Decisions* and *Technology Decisions* during the development of quantum software, particularly due to the integration of quantum-related components and the intricate nature of quantum technology, which often leads to critical issues. Consequently, we suggest that practitioners should carefully consider *Implementation Decisions* and *Technology Decisions* in quantum software development.

5.2.3. Application domains

Researchers should focus on refining integration, developing modular designs, and implementing advanced error mitigation for reliable quantum software: Our findings have several key implications for both the academic community and industry professionals. 📖 Researchers should focus on refining the integration procedures between various quantum programming frameworks and tools. Ensuring seamless compatibility between different modules and APIs is critical. For example, *"Align current BaseJob API with the specification"* (Qiskit issue #933) and *"The SDK also provides the ability to request specific result types on a subset of qubits"* (amazon-braket-sdk-python issue #158) illustrate the need for continuous improvement in integration to support diverse quantum computing tasks. Emphasis should be placed on the design and evolution of modular quantum software architectures. The decomposition and refactoring of components, such as *"After we have finished the decomposition and local Covalent has a microservice architecture"* (covalent issue #186), highlight the importance of maintaining flexibility and scalability in quantum software

systems. 📖 Researchers should explore innovative modular designs to accommodate the evolving requirements of quantum computing. The development of robust error mitigation techniques, as indicated by "*Mitiq needs more error mitigation techniques*" (mitiq issue #821), is crucial for improving the reliability and accuracy of quantum computations. 📖 Researchers should investigate and implement advanced error correction methods to enhance the performance of quantum algorithms. The rapid evolution of quantum computing paradigms necessitates adaptive and forward-thinking research approaches. Proposals like "*This issue is motivated by problems emerged in #471 (Bayesian ZNE), #477 (Clifford-data regression)*" (mitiq issue #517) underscores the need for continuous adaptation to emerging quantum computing challenges and opportunities.

Practitioners must ensure tool compatibility, create user-friendly interfaces, and adopt rigorous testing and modern development practices for enhanced quantum software: 👤 Practitioners must ensure the compatibility and coherence of tools and interfaces used in quantum software development. Issues like "*The sync method in covalent/_results_manager/results_manager.py needs to be moved and updated to be compatible with the new covalent services architecture*" (covalent issue #370) highlight the necessity for ongoing maintenance and updates to ensure smooth operations across different components. Developing user-friendly and flexible tools is essential for the practical implementation of quantum software. Statements such as "*We are also interested in using the Python SDK*" (Qiskit issue #487) and "*While the actual operation requested is technically possible without going to a DAG, it doesn't fit into the compilation tooling architecture of Terra*" (Qiskit issue #1852) emphasize the need for designing tools that are both technically robust and accessible to users. 👤 Practitioners should adopt rigorous testing and continuous improvement practices. Examples like "*Fix unit tests (locally there is no problem, but travis keeps on failing due to connection issues) modify docs*" (pyrpl issue #350) and "*Adapt coverage testing to require 100% of all new code*" (circuit-knitting-toolbox issue #107) illustrate the importance of thorough testing and validation to ensure the reliability of quantum software systems. The adoption of modern development practices, such as microservices [97] and agile methodologies [98], can significantly enhance the development process. The shift towards architectures like "*local Covalent has a microservice architecture*" (covalent issue #186) demonstrates the benefits of modular, scalable, and maintainable software architecture designs in quantum system development.

5.2.4. Quality attributes

The quality attributes considered when making architecture decisions in quantum software development offer valuable insights for both researchers and practitioners: The quality attributes delve into the crucial aspects of software architecture in quantum software development. 📖 Understanding these quality attributes is essential for developing efficient and maintainable quantum software systems, guiding both the research agenda and practical approaches in quantum software development. Therefore, there is a need to provide guidelines for both 📖 researcher and 👤 practitioners. 📖 Researchers can explore the ways in which quality attributes are considered when formulating architectural decisions in quantum software development. 📖 Researchers can also use this knowledge to focus on the identified quality attributes (e.g., *Maintainability*, *Performance*, and *Compatibility*), tailoring their theoretical frameworks and empirical studies to address the specific challenges and needs in quantum software architecture. Recognition of the importance of quality attributes in quantum software development can be directed towards constructing methodologies and tools that enhance these quality attributes in quantum software systems. This focus is crucial, given the inherent complexities of quantum software and the need for integrating quality attributes in quantum software with classical software paradigms. 👤 Practitioners can utilize our findings of quality attributes to make well-informed architecture decisions during the development of quantum software. The emphasis on

Maintainability, *Performance*, and *Compatibility* as key quality attributes indicates the areas where 👤 practitioners should focus their efforts. This knowledge assists in creating more robust and efficient quantum software systems, directly impacting the practical application of quantum technologies. Furthermore, understanding these quality attributes can aid in making valid architecture decisions, ensuring that quantum software not only meets technological demands but is also maintainable and compatible to future advancements and requirements.

5.2.5. Limitations and challenges

Practitioners and researchers face significant *Design Issues* in quantum software development, requiring adaptations of traditional practices and exploration of new frameworks like layered software architecture: 👤 Practitioners and 📖 researchers alike must grapple with critical challenges and limitations when making architecture decisions in quantum software development. These challenges encompass various aspects, prominently including *Design Issues* that arise from the unique demands of quantum computing, as mentioned by a developer, "*the authors do lay out some of the challenges faced when designing quantum computing design tools and programming languages*" (CS post #29032); this indicates recognition of difficulties in adapting traditional software design practices to the quantum realm and marks the exploration of 👤 practitioners into the field of quantum software, focusing on the technical and conceptual challenges in designing effective quantum programming languages and design tools. The proposed solution involves adopting a layered software architecture framework, as mentioned by a developer, "*a Layered Software Architecture for Quantum Computing Design Tools by Svore et al. lays out a framework that I believe cuts across what you're getting it*" (CS post #29032). 📖 For researchers, understanding and addressing these challenges pave the way for advancing quantum software engineering, and the exploration of quantum software design tools and programming languages marks an important frontier in quantum computing research.

High Error Rates demand advanced error management and innovative error mitigation techniques to improve reliability in quantum software development: Another prominent challenge identified in this study is the management of *High Error Rates* inherent in quantum software systems. A notable example of this challenge is encountered in quantum chemistry simulations where high error rates can lead to inaccuracies in computational results, as mentioned by a developer, "*The build systems have only seen compilation on half a dozen architectures and setups, so you may encounter errors in the build process*" (Psi4 issue #468). To mitigate this challenge, one potential solution was discussed by a developer, "*In addition, python-based error messages will be more expressive and accurate. For users who enjoy the current Psithon interface, the current Psi4 Psithon parsing equipment will be moved to a script named 'psi4' in the binary location. This will allow all current Psi4 input files to be run normally*" (Psi4 issue #468). In this example, python-based error messages can be more expressive and accurate, thereby simplifying the debugging process and addressing *High Error Rates*. Addressing *High Error Rates* challenges requires a dedicated focus on developing advanced error management techniques. Therefore, 👤 practitioners should implement enhanced error management capabilities within their quantum software systems. 📖 Researchers should explore innovative methodologies for error mitigation and correction to improve the reliability of quantum software systems.

Design documentation is crucial to avoid failure in quantum software: Both 📖 researchers and 👤 practitioners in the field of quantum software development can benefit significantly from recognizing the importance of design documentation. As highlighted by one developer, "*the absence of a design document enumerating realistic use cases hides the design goals and blurs the definition of done*" (Qiskit issue #825), the absence of a design document that includes realistic use cases can obscure the design goals and make it challenging to define project completion. 📖 Therefore, it is crucial for researchers to create design

documentation that clearly outlines the intended objectives, realistic use cases, and specific requirements of quantum software systems. This design documentation serves as a roadmap for development and helps avoid misalignment between the design goals and the actual implementation of quantum software systems, preventing discrepancies and misunderstandings during the development process. 🧍 Practitioners can rely on design documentation to understand the intended functionality and requirements of the quantum software system. With well-defined documentation that enumerates realistic use cases, 📖 researchers and 🧍 practitioners can better understand the design goals and the definition of project completion. This clarity can lead to more successful outcomes and prevent the failures of quantum software systems.

5.2.6. Relationship between the RQs

Critical application domains: The most critical application domains in quantum software in which *Implementation Decisions* are made include: *Software Development Tools*, *Quantum Programming Frameworks*, *Optimization and Simulation*, *Data Processing*, and *Scientific Software*. *Software Development Tools* have the highest number of *Implementation Decisions* (27 out of 125), representing 21.6%. This indicates a strong focus on tools that support the development process. *Quantum Programming Frameworks* with 23 (18.4%), this domain emphasizes the importance of frameworks that facilitate quantum programming. *Optimization and Simulation* accounts for 17 *Implementation Decisions* (13.6%), showing the significance of optimizing and simulating quantum systems. *Scientific Software* with 10 (8%), this domain highlights the role of scientific applications in quantum computing. *Data Processing* holding 8 (6.4%), this domain underlines the necessity for efficient data handling in quantum software. These findings underscore the critical roles that *Software Development Tools* and *Quantum Programming Frameworks* play in the architecture of quantum software. 🧍 Practitioners should focus on selecting and utilizing tools that enhance *Maintainability* and *Performance*, given their significant impact on the success of quantum software applications.

Key quality attributes: The analysis of quality attributes reveals the following insights: *Maintainability* has the highest frequency of *Implementation Decisions* (36 out of 228), accounting for 15.8%. It highlights the importance of maintaining the software over time. *Performance* with 34 *Implementation Decisions* (14.9%), performance is a key concern in quantum software development. *Compatibility* has 30 *Implementation Decisions* (13.2%), emphasizing the need for software to function well with other systems. *Consistency* holding 22 *Implementation Decisions* (9.6%), consistency is crucial for ensuring reliable and predictable software behavior. *Complexity* with 19 *Implementation Decisions* (8.3%), managing complexity is essential for developing efficient quantum software. 📖 These quality attributes (e.g., Maintainability, Performance, Compatibility) highlight critical research areas in quantum software where further research is needed. 🧍 Understanding these quality attributes is crucial, guiding practitioners to make suitable architectural decisions when developing quantum software systems.

Limitations and challenges: *Design Issues* appeared as the most prevalent challenge, accounting for 48 out of 228 *Implementation Decisions* (21.1%). This high frequency underscores the common occurrence of design-related challenges in quantum software development. *Performance Issues* with 20 *Implementation Decisions* (8.8%), performance issues are a significant challenge. *Implementation Issues* accounts for 19 *Implementation Decisions* (8.3%), indicating difficulties in the implementation process. *Complexity Issues* holding 11 *Implementation Decisions* (4.8%), complexity remains a notable challenge in quantum software development. *High Error Rate* challenge has 8 *Implementation Decisions* (3.5%), highlighting the issue of errors in quantum software. Addressing these challenges is crucial for advancing the field and developing robust quantum software systems. The challenges identified, such as *Design Issues* (21.1%) and *Performance Issues* (8.8%), provide a clear direction for future studies. 📖 Researchers can delve deeper into these areas to develop innovative solutions that address architecture related limitations and challenges, thereby advancing the state of quantum software development.

6. Threats to validity

6.1. Construct validity

Construct validity refers to whether our measurement tools accurately assess the intended properties [99]. In this study, we encountered two potential challenges to construct validity: search term selection and manual analysis. Search term selection: The retrieval of related issues from GitHub and related posts from Stack Exchange sites (including platforms such as Stack Overflow, Quantum Computing Stack Exchange, and Computer Science Stack Exchange) relies on keyword-based searches. When it came to selecting search keywords for finding GitHub issues and Stack Exchange posts, we faced challenges related to the construct validity of our data collection. One significant concern was whether the keywords we initially chose would capture all the relevant issues and posts or if some valuable information might be missed. To address this potential threat, we took a proactive approach by conducting pilot searches using alternative search terms before using the original terms discussed in Section 3.2. This iterative process allowed us to refine our keyword choices, ensuring that we cast a wider net and minimize the risk of incomplete data, thus safeguarding the construct validity of our study. Manual data handling: The stages of data handling in our study, which involve manual processes and thus risk subjectivity and bias, are as follows: (1) Labeling– To minimize bias in this stage, we implemented a two-step approach. Initially, we performed a pilot labeling on both GitHub issues and Stack Exchange posts. This was done prior to the formal labeling to identify potential biases and inconsistencies. Furthermore, we established the inclusion and exclusion criteria to objectively decide which posts or issues should be included in our study. (2) Extraction: The pilot data extraction process was a collaborative effort primarily between the first and second authors. To ensure accuracy and consistency, the first author conducted a thorough recheck of the extraction results. This step was crucial to maintain the integrity of the data extraction process. If any discrepancies were found, the third author participated in discussions with the first two authors to help reach a consensus; (3) Analysis: To mitigate the threat of data analysis, we used two qualitative techniques (predefined classification and the Constant Comparison method) to analyze the extracted data and answer the RQs. Moreover, we attempted to minimize this threat by performing a pilot data analysis before the formal data analysis. The first and second authors independently categorized content from filtered posts and issues, cross-referencing the filtered posts and issues with the corresponding data items in Table 5. The first author then consolidated these codes into higher-level concepts and categories, while the other authors reviewed and validated the pilot data analysis results, resolving disagreements through the negotiated agreement method [92] to enhance reliability.

6.2. External validity

External validity encompasses the extent to which the outcomes of a study can be applied and verified in diverse settings, thereby assessing the generalization of the findings. One primary factor influencing external validity is the selection of data sources. In our effort to mitigate this potential limitation, we deliberately opted for two widely recognized developer communities, namely GitHub and Stack Exchange as the primary sources for extracting architecture decisions in the realm of quantum software development. It is important to note that GitHub and Stack Exchange have consistently served as pivotal resources in empirical software engineering investigations. Previous research endeavors (e.g., [100,101]) have demonstrated that insights obtained from GitHub and Stack Exchange data enjoy a high degree of validation and resonance among practitioners. This strong alignment between research findings and real-world experiences bolsters our confidence in the representatives of these platforms for studying architecture decisions in quantum software development. Furthermore,

it is important to acknowledge that quantum computing is an emerging field still in its early stages compared to traditional software systems. As a result, the datasets available in this domain are often of limited sample size. For instance, Openja et al. [87] conducted an empirical study on technical debt and faults in open-source quantum software systems using a dataset of 118 projects, highlighting the constraints and realities of research in this emerging field. This situation further emphasizes the importance of our study, which expands the initial dataset and continues to contribute to the understanding of quantum software architecture. The specific nature of our dataset reflects the current landscape of quantum software systems, which presents unique challenges for empirical research on this topic. We also acknowledge that encompassing more data sources would enhance the study's external validity.

6.3. Reliability

Reliability, in the context of our research methodology, pertains to the degree of consistency exhibited by the chosen approach in yielding results. In order to mitigate the potential threat to reliability, a preliminary data labeling exercise was carried out independently by two authors on a sample consisting of 10 GitHub issues and 10 Stack Exchange posts. The assessment of inter-rater reliability was conducted through the calculation of Cohen's Kappa coefficient, yielding values of 0.80 for GitHub issues and 0.76 for Stack Exchange posts. Both of these coefficients exceed 0.7, indicating a satisfactory level of agreement between the participating authors. To further enhance the reliability of the process encompassing data labeling, data extraction, and data analysis, any disparities or inconsistencies that arose were resolved through constructive discussions involving the first author and other authors. To foster transparency and enable validation of our study findings, we have made the entire dataset, along with the corresponding data labeling outcomes for GitHub issues and Stack Exchange posts, readily available [53]. This provision ensures that other researchers can scrutinize our work and verify the consistency and reliability of our data handling and analysis procedures.

7. Related work

7.1. Quantum software development

In recent years, quantum software development has gained significant attention in the software engineering community, reflecting the growing importance of quantum computing. Khan et al. [38] conducted a systematic review to examine the software architecture for quantum computing systems. Their work provides a comprehensive overview of existing architectures of quantum software systems, highlighting the advancements and challenges in this domain. Truger et al. [102] explored the intersection of warm-starting and quantum computing through a systematic mapping study, shedding light on how quantum computing can benefit from optimizing initialization procedures. Haghparast et al. [103] took a developer's perspective to identify the challenges in quantum software engineering and mapped these challenges to a proposed workflow model, offering insights into the practical issues developers face in this field. Yue et al. [30] examined the challenges and opportunities in quantum software architecture, highlighting key considerations in designing quantum computing software. Felderer et al. [64] explored software engineering challenges in quantum computing through the First Working Seminar on Quantum Software Engineering (WSQSE 2022), providing valuable insights into this evolving domain. Finally, Zhao et al. [104] made a contribution to the field of quantum programming by introducing Bugs4Q, which is a novel benchmark suite comprising forty-two real and manually validated Qiskit bugs extracted from prominent platforms such as GitHub, StackOverflow, and Stack Exchange. These bugs are supplemented with test cases that facilitate reproducing erroneous behaviors, and these bug

scenarios are invaluable for systematically evaluating debugging and testing methodologies for quantum programs. These papers collectively contribute to the understanding of quantum software development, addressing various aspects such as software architecture, optimization, workflow models, challenges, and benchmarking in quantum computing systems.

7.2. Decisions in software engineering

Decisions are either explicitly presented in knowledge management tools or implied in various textual artifacts. Extensive research has been conducted to understand and analyze decisions and decision-making in software engineering. Li et al. [105] conducted an exploratory study on the Hibernate developer mailing list, analyzing 9006 posts to uncover insights into decision expressions, classification, rationale, approaches, related software artifacts, and the trend of decision-making over time. Their findings revealed that decisions in open-source software development are typically expressed as Information Giving, Solution Proposal, and Feature Request, with the main categories being Design Decision and Requirement Decision. In a separate study, Bi et al. [106] investigated how developers use architecture and design patterns concerning quality attributes and design contexts, revealing previously unknown relationships among these elements. Zhang et al. [88] conducted an empirical study on architecture decisions in AI-based systems development. They analyzed the data from Stack Overflow and GitHub, revealing that architecture decisions in AI are expressed through various linguistic patterns, with Solution Proposal and Information Giving being common. Their study identified Technology Decision, Component Decision, and Data Decision as primary decision types, with a focus on Performance as the key quality attribute. It also highlighted Design Issues and Data Issues as significant challenges in the architecture decision-making of AI-based systems. Waseem et al. [80] introduced decision models aimed at assisting microservices practitioners in selecting appropriate patterns and strategies for Microservices Architecture (MSA). These models cover four key MSA design areas, including microservices application decomposition, security, communication, and service discovery. To create these models, the authors conducted an extensive literature review and evaluated their effectiveness through interviews with 24 practitioners, and the results show that these decision models were found to be valuable tools for guiding microservices patterns and strategy selection. Shahin et al. [107] explored the visualization of architectural design decisions, focusing on the use of the Compendium tool to improve understandability and communication in architectural design. Lastly, Liu et al. [108] provided insights into the decision-making of students in requirements engineering course projects, emphasizing the importance of understanding the characteristics of decision-making during various stages of requirements engineering. These works collectively contribute to our understanding of decision-making in software engineering, encompassing diverse facets such as decision expression, architecture patterns, visualization, and decision-making in the software development process.

7.3. Architecture decisions in quantum software development

Khan et al. [38] conducted a systematic review to delve into the architectural aspects of quantum software development. Their study provides insights into how architectural processes, modeling notations, design patterns, tool support, and challenging factors impact quantum software architecture. The authors highlighted the transformation of quantum bits (Qubits) into architectural components, offering a fresh perspective on architectural decisions in quantum computing systems. Sodhi et al. [45] investigated Quantum Computing Platforms (QCPs) from a software engineering perspective. They proposed a QCPs architecture and a programming model and evaluated the impact of QCPs characteristics on Quality Attributes (QAs) and Software Development Life Cycle (SDLC) activities. Their findings suggest that QCPs

are best suited for specialized applications like scientific computing, with some QAs, such as maintainability and reliability, being negatively affected by certain QCPs characteristics (e.g., limitations in qubit state transmission, hard dependencies on quantum algorithms, and restrictions on copying and deletion of qubit states). Furthermore, Khan et al. [109] researched agile practices in quantum software development and delved into the challenges and applicability of agile practices in this domain. While their work primarily focuses on the adoption of agile practices, it also indirectly touches on limitations and challenges in quantum software development. Their study demonstrated that agile software development practices have the potential to address some of the challenges inherent in quantum software development, but it also highlighted new obstacles that impede their effective incorporation. This research offers valuable insights into the landscape of challenges faced when making architecture decisions in quantum software development.

Distinguished from the before mentioned research, our investigation delved into a comprehensive analysis of architectural decision-making within quantum software development. Specifically, our study encompassed an examination of the expressions and classifications of architecture decisions, the diverse application domains in which these decisions manifest, the critical quality attributes that factor into decision-making, as well as the formidable limitations and challenges encountered during this architectural process. To conduct this exploration, we conducted data mining across Stack Exchange and GitHub, amassing valuable insights into the intricacies of quantum software development's architectural landscape.

8. Conclusions

Quantum software architecture refers to the design and organization blueprint for quantum software systems that leverage the principles of quantum computing to solve specific problems or tasks. It focuses on addressing the challenges and limitations inherent to quantum computation, providing a structured approach to building applications that leverage the power of quantum technologies. Architecture decisions are essential for architects and developers to make informed design choices during the development of quantum software systems, and once a system is developed, it can be challenging and costly to change at the architecture level. This research aimed to explore the architecture decisions within the context of quantum software development. Specifically, we conducted an empirical study on GitHub issues and Stack Exchange posts to explore architecture decisions in quantum software development from practitioners' perspectives. We used a keyword-based search to collect data from GitHub and Stack Exchange. Finally, we got 385 GitHub issues and 70 Stack Exchange posts which include 33 Stack Overflow posts, 33, Quantum Computing Stack Exchange posts, and 4 Computer Science Stack Exchange posts. We manually extracted architecture decisions related to quantum software systems to investigate decision expressions, decision types, involved application domains, quality attributes considered, and limitations and challenges encountered in architecture decision-making of quantum software development. The main results are that:

- In quantum software development, architecture decisions are expressed through six distinct linguistic patterns, with *Solution Proposal* and *Information Giving* being the most prevalent. *Solution Proposal* constitutes 35.83%, while *Information Giving* accounts for 32.80% of the linguistic patterns used. Conversely, *Opinion Asking* and *Problem Discovery* are relatively less used in the linguistic patterns we collected.
- Architecture decisions primarily fall into two major categories: *Implementation Decision* and *Technology Decision*. Specifically, *Implementation Decision* represents 22.10%, and *Technology Decision* accounts for 18.81% of architecture decisions we collected.

- *Software Development Tools* stand out as the most common application domain among the twenty domains identified, representing 22.15% of the application domain we collected.
- Two important quality attributes considered in architectural decision-making of quantum software systems are *Maintainability* (18.53%) and *Performance* (15.68%) of the quality attributes we collected.
- Practitioners in quantum software development face significant limitations and challenges. *Design Issues* are prominent, accounting for 25.07%, while *High Error Rates* are a notable concern at 11.37% of the limitations and challenges we collected.

The findings of this study offer valuable insights to researchers and practitioners. By identifying common linguistic patterns, major decision categories, prevalent application domains, and critical quality attributes, this research equips professionals in quantum software systems with a deeper understanding of the made architectural decisions. Additionally, it highlights common challenges of making architecture decisions in quantum software development, such as design and performance issues, providing researchers and practitioners with the awareness needed to address these issues.

In the next step: (1) We intend to conduct a comparative study of architecture decisions on GitHub, Stack Exchange, and other platforms like developer mailing lists, aiming to reveal insights into the current focus of architecture decision-making in quantum software systems, including its advantages and deficiencies. (2) To provide a comprehensive perspective, we will validate and extend the proposed taxonomy of architecture decisions using a practitioner survey or interview, collecting insights from professionals actively engaged in quantum software development. (3) Finally, our overarching goal is to address the limitations and challenges identified in this study. We aim to develop decision models for selecting architecture patterns and strategies in architecting quantum software systems, contributing to more informed and effective architecture decision-making in quantum software development.

CRediT authorship contribution statement

Mst Shamima Aktar: Writing – original draft, Visualization, Validation, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Peng Liang:** Writing – review & editing, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Muhammad Waseem:** Writing – review & editing, Visualization, Validation, Supervision, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Amjed Tahir:** Writing – review & editing, Validation, Supervision, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Aakash Ahmad:** Writing – review & editing, Visualization, Validation, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Beiqi Zhang:** Writing – review & editing, Validation, Investigation, Formal analysis, Data curation. **Zengyang Li:** Writing – review & editing, Validation, Methodology, Investigation, Data curation, Conceptualization.

Declaration of competing interest

Data availability

We have shared the link to our data in the reference [53] of our submission.

Acknowledgments

References

[1] S.T. Marella, H.S.K. Parisa, Introduction to Quantum Computing, IntechOpen, 2020.

[2] M. Steffen, D.P. DiVincenzo, J.M. Chow, T.N. Theis, M.B. Ketchen, Quantum computing: An IBM perspective, IBM J. Res. Dev. 55 (5) (2011) 13:1–13:11.

[3] S. Lloyd, Ultimate physical limits to computation, Nature 406 (6799) (2000) 1047–1054.

[4] V. Mavroeidis, K. Vishi, M.D. Zych, A. Jøsang, The impact of quantum computing on present cryptography, 2018, arXiv preprint arXiv:1804.00200.

[5] J. Gruska, Quantum computing challenges, in: Mathematics Unlimited—2001 and Beyond, Springer, 2001, pp. 529–563.

[6] A. Ajagekar, F. You, Quantum computing for energy systems optimization: Challenges and opportunities, Energy 179 (2019) 76–89.

[7] S. Vyakaranal, S. Kengond, Performance analysis of symmetric key cryptographic algorithms, in: Proceedings of the 10th International Conference on Communication and Signal Processing, ICCSP, IEEE, Tamilnadu, India, 2018, pp. 0411–0415.

[8] W. Li, Quantum-accelerated big data analytics on cloud platforms: Leveraging quantum computing for large-scale data processing, J. Big-Data Anal. Cloud Comput. 9 (1) (2024) 14–24.

[9] R. Srivastava, I. Choi, T. Cook, N. Team, Commercial prospects for quantum computing, 2016.

[10] K. Svore, A. Cross, A. Aho, I. Chuang, I. Markov, Toward a software architecture for quantum computing design tools, in: Proceedings of the 2nd International Workshop on Quantum Programming Languages, QPL, Turku, Finland, 2004, pp. 145–162.

[11] A. Bhasin, M. Tripathi, Quantum computing at an inflection point: Are we ready for a new paradigm, IEEE Trans. Eng. Manage. 70 (7) (2023) 2546–2557.

[12] B. Weder, J. Barzen, F. Leymann, M. Zimmermann, Hybrid quantum applications need two orchestrations in superposition: A software architecture perspective, in: Proceedings of the 18th IEEE International Conference on Web Services, ICWS, IEEE, Chicago, IL, USA, 2021, pp. 1–13.

[13] A.J. McCaskey, E.F. Dumitrescu, D. Liakh, M. Chen, W.-C. Feng, T.S. Humble, A language and hardware independent approach to quantum–classical computing, SoftwareX 7 (2018) 245–254.

[14] J. Preskill, Quantum computing in the NISQ era and beyond, Quantum 2 (2018) 79–98.

[15] E. Strubell, An introduction to quantum algorithms, COS498 Chawathe Spring 13 (2011) 1–34.

[16] D. Willsch, M. Willsch, F. Jin, H. De Raedt, K. Michielsen, Large-scale simulation of shor's quantum factoring algorithm, 2023, arXiv preprint arXiv:2308.05047.

[17] A. Mandviwalla, K. Ohshiro, B. Ji, Implementing grover's algorithm on the IBM quantum computers, in: Proceedings of the 6th IEEE International Conference on Big Data, BigData, IEEE, Seattle, WA, USA, 2018, pp. 2531–2537.

[18] A. Montanaro, Quantum algorithms: an overview, NPJ Quant. Inf. 2 (1) (2016) 1–8.

[19] A.W. Harrow, A. Hassidim, S. Lloyd, Quantum algorithm for linear systems of equations, Phys. Rev. Lett. 103 (15) (2009) 150502.

[20] B. Heim, M. Soeken, S. Marshall, C. Granade, M. Roetteler, A. Geller, M. Troyer, K. Svore, Quantum programming languages, Nat. Rev. Phys. 2 (12) (2020) 709–722.

[21] J. Zhao, Quantum software engineering: Landscapes and horizons, 2020, arXiv preprint arXiv:2007.07047.

[22] S.M. Sundaram, T.R. Murgod, Quantum software engineering and technology, in: Technology Road Mapping for Quantum Computing and Engineering, IGI Global, 2022, pp. 102–116.

[23] N. Dey, M. Ghosh, A. Chakrabarti, LIQUi|>: A functional quantum software architecture and tool suite for quantum computing developed by microsoft research, 2020, arXiv preprint arXiv:2010.08053.

[24] B.K. Behera, T. Reza, A. Gupta, P.K. Panigrahi, Designing quantum router in IBM quantum computer, Quantum Inf. Process. 18 (2019) 1–13.

[25] R. Courtland, Google aims for quantum computing supremacy [news], IEEE Spectr. 54 (6) (2017) 9–10.

[26] C.G. Almudever, L. Lao, X. Fu, N. Khammassi, I. Ashraf, D. Iorga, S. Varsamopoulos, C. Eichler, A. Wallraff, L. Geck, A. Kruth, J. Knoch, H. Bluhm, K. Bertels, The engineering challenges in quantum computing, in: Proceedings of the 20th Design, Automation, and Test in Europe, DATE, IEEE, Lausanne, Switzerland, 2017, pp. 836–845.

[27] S. Ali, T. Yue, R. Abreu, When software engineering meets quantum computing, Commun. ACM 65 (4) (2022) 84–88.

[28] M. Piattini, G. Peterssen, R. Pérez-Castillo, Quantum computing: A new software engineering golden age, ACM SIGSOFT Softw. Eng. Not. 45 (3) (2021) 12–14.

[29] M.A. Serrano, J.A. Cruz-Lemus, R. Perez-Castillo, M. Piattini, Quantum software components and platforms: Overview and quality assessment, ACM Comput. Surv. 55 (8) (2022) 1–31.

[30] T. Yue, W. Mauerer, S. Ali, D. Taibi, Challenges and opportunities in quantum software architecture, in: Software Architecture: Recent Trends in Software Architecture, Springer, 2023, pp. 45–52.

[31] B. Kommadi, Quantum Computing Solutions: Solving Real-World Problems Using Quantum Computing and Algorithms, Springer, 2020.

[32] E. Moguel, J. Berrocal, J. García-Alonso, J.M. Murillo, A roadmap for quantum software engineering: Applying the lessons learned from the classics, in: Proceedings of the 1st International Workshop on Software Engineering & Technology Co-Located with the 1st IEEE International Conference on Quantum Computing and Engineering, Q-SET@QCE, Colorado, USA, 2020, pp. 5–13.

[33] M. Piattini, M. Serrano, R. Perez-Castillo, G. Petersen, J.L. Hevia, Toward a quantum software engineering, IT Prof. 23 (1) (2021) 62–66.

[34] A. García de la Barrera, I. García-Rodríguez de Guzmán, M. Polo, M. Piattini, Quantum software testing: State of the art, J. Softw. Evol. Process 35 (4) (2023) e2419.

[35] L. Zhang, A. Miranskyy, W. Rjaibi, G. Stager, M. Gray, J. Peck, Making existing software quantum safe: A case study on IBM Db2, Inf. Softw. Technol. 161 (2023) 107249.

[36] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, third ed., Addison-Wesley Professional, 2012.

[37] X. Zhao, X. Xu, L. Qi, X. Xia, M. Bilal, W. Gong, H. Kou, Unraveling quantum computing system architectures: An extensive survey of cutting-edge paradigms, Inf. Softw. Technol. 167 (2024) 107380.

[38] A.A. Khan, A. Ahmad, M. Waseem, P. Liang, M. Fahmideh, T. Mikkonen, P. Abrahamsson, Software architecture for quantum computing systems - A systematic review, J. Syst. Softw. 201 (2023) 111682.

[39] A. El Azzaoui, M.M. Salim, J.H. Park, Secure and reliable big-data-based decision making using quantum approach in IIoT systems, Sensors 23 (10) (2023) 4852.

[40] A. Jansen, J. Bosch, Software architecture as a set of architectural design decisions, in: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture, WICSA, IEEE, Pittsburgh, Pennsylvania, USA, 2005, pp. 109–120.

[41] I. Malavolta, H. Muccini, S.R. V, Enhancing architecture design decisions evolution with group decision making principles, in: Proceedings of the 6th International Workshop on Software Engineering for Resilient Systems, SERENE, Springer, Budapest, Hungary, 2014, pp. 9–23.

[42] Z. Li, P. Liang, P. Avgeriou, Architectural technical debt identification based on architecture decisions and change scenarios, in: Proeedings of the 12th Working IEEE/IFIP Conference on Software Architecture, WICSA, IEEE, Montreal, QC, Canada, 2015, pp. 65–74.

[43] D. Vietz, J. Barzen, F. Leymann, K. Wild, On decision support for quantum application developers: categorization, comparison, and analysis of existing technologies, in: Proceedings of the 21st International Conference on Computational Science, ICCS, Springer, Krakow, Poland, 2021, pp. 127–141.

[44] A. Ahmad, A.A. Khan, M. Waseem, M. Fahmideh, T. Mikkonen, Towards process centered architecting for quantum software systems, in: Proceedings of the 1st IEEE International Conference on Quantum Software, QSW, IEEE, Barcelona, Spain, 2022, pp. 26–31.

[45] B. Sodhi, R. Kapur, Quantum computing platforms: Assessing the impact on quality attributes and SDLC activities, in: Proceedings of the 18th IEEE International Conference on Software Architecture, ICSA, IEEE, Stuttgart, Germany, 2021, pp. 80–91.

[46] J. Barzen, F. Leymann, S. Feld, M. Wimmer, 2Nd workshop on quantum software architecture (QSA), in: Proceedings of the 19th IEEE International Conference on Software Architecture Companion, ICSA-C, IEEE, Los Alamitos, CA, USA, 2022, 128–128.

[47] Y. Naveh, Quantum software development is still in its infancy, 2021.

[48] U. Awan, L. Hannola, A. Tandon, R.K. Goyal, A. Dhir, Quantum computing challenges in the software industry. A fuzzy AHP-based approach, Inf. Softw. Technol. 147 (2022) 106896.

[49] I. Polian, A.G. Fowler, Design automation challenges for scalable quantum architectures, in: Proceedings of the 52nd Annual Design Automation Conference, DAC, ACM, San Francisco, CA, USA, 2015, pp. 1–6.

[50] P. Murali, N.M. Linke, M. Martonosi, A.J. Abhari, N.H. Nguyen, C.H. Alderete, Full-stack, real-system quantum computer studies: Architectural comparisons and design insights, in: Proceedings of the 46th International Symposium on Computer Architecture, ISCA, ACM, Phoenix, AZ, USA, 2019, pp. 527–540.

[51] A. Di Sorbo, S. Panichella, C.A. Visaggio, M. Di Penta, G. Canfora, H.C. Gall, Development emails content analyzer: Intention mining in developer discussions, in: Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering, ASE, IEEE, Lincoln, NE, USA, 2015, pp. 12–23.

[52] R.W. Grove, An analysis of the constant comparative method, Int. J. Qual. Stud. Educ. 1 (3) (1988) 273–279.

[53] M.S. Aktar, P. Liang, M. Waseem, A. Tahir, A. Ahmad, B. Zhang, Dataset for the paper: Architecture decisions in quantum software systems: An empirical study on stack exchange and GitHub, 2024, http://dx.doi.org/10.5281/zenodo.10050560.

[54] R.P. Feynman, Simulating physics with computers, Internat. J. Theoret. Phys. 21 (6/7) (2018).

[55] D.A. Sofge, A survey of quantum programming languages: History, methods, and tools, in: Proceedings of the 2nd International Conference on Quantum, Nano and Micro Technologies, ICQNM, IEEE, Sainte Luce, Martinique, France, 2008, pp. 66–71.

[56] A.M. Childs, D. Maslov, Y. Nam, N.J. Ross, Y. Su, Toward the first quantum simulation with quantum speedup, Proc. Natl. Acad. Sci. 115 (38) (2018) 9456–9461.

[57] B. Coecke, G. de Felice, K. Meichanetzidis, A. Toumi, Foundations for near-term quantum natural language processing (2020), 2020, arXiv preprint arXiv:2012.03755.

[58] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, S. Lloyd, Quantum machine learning, Nature 549 (7671) (2017) 195–202.

[59] D. Tosh, O. Galindo, V. Kreinovich, O. Kosheleva, Towards security of cyber-physical systems using quantum computing algorithms, in: Proceedings of the 15th International Conference of System of Systems Engineering, SoSE, IEEE, Budapest, Hungary, 2020, pp. 313–320.

[60] N. Quetschlich, L. Burgholzer, R. Wille, MQT bench: Benchmarking software and design automation tools for quantum computing, Quantum 7 (2023) 1062.

[61] F.T. Chong, D. Franklin, M. Martonosi, Programming languages and compiler design for realistic quantum hardware, Nature 549 (7671) (2017) 180–187.

[62] J.-F. Bobier, M. Langione, E. Tao, A. Gourevitch, What Happens When 'If' Turns to 'When' in Quantum Computing? Boston Consulting Group, 2021.

[63] H. Van Vliet, H. Van Vliet, J. Van Vliet, Software Engineering: Principles and Practice, vol. 13, John Wiley & Sons Hoboken, NJ, 2008.

[64] M. Felderer, D. Taibi, F. Palomba, M. Epping, M. Lochau, B. Weder, Software engineering challenges for quantum computing: Report from the first working seminar on quantum software engineering (WSQSE 22), ACM SIGSOFT Softw. Eng. Not. 48 (2) (2023) 29–32.

[65] M.A. Akbar, A.A. Khan, S. Mahmood, S. Rafi, Quantum software engineering: A new genre of computing, 2022, arXiv preprint arXiv:2211.13990.

[66] K. Dwivedi, M. Haghparast, T. Mikkonen, Quantum software engineering and quantum software development lifecycle: a survey, Cluster Comput. (2024) 1–19.

[67] M. De Stefano, F. Pecorelli, D. Di Nucci, F. Palomba, A. De Lucia, Software engineering for quantum programming: How far are we? J. Syst. Softw. 190 (2022) 111326.

[68] D. Wecker, K.M. Svore, LIQUi|>: A software design architecture and domain-specific language for quantum computing, 2014, arXiv preprint arXiv:1402.4467.

[69] R. Abreu, S. Ali, T. Yue, First international workshop on quantum software engineering (q-se 2020), ACM SIGSOFT Softwa. Eng. Not. 46 (2) (2021) 30–32.

[70] J.M. Murillo, J. Garcia-Alonso, E. Moguel, J. Barzen, F. Leymann, S. Ali, T. Yue, P. Arcaini, R. Pérez, I.G.R. de Guzmán, et al., Challenges of quantum software engineering for the next decade: The road ahead, 2024, arXiv preprint arXiv:2404.06825.

[71] H. Li, F. Khomh, M. Openja, et al., Understanding quantum software engineering challenges an empirical study on stack exchange forums and GitHub issues, in: Proceedings of the 37th IEEE International Conference on Software Maintenance and Evolution, ICSME, IEEE, Luxembourg, 2021, pp. 343–354.

[72] M.A. Akbar, A.A. Khan, S. Rafi, A systematic decision-making framework for tackling quantum software engineering challenges, Autom. Softw. Eng. 30 (2) (2023) 22.

[73] S. Ducasse, D. Pollet, Software architecture reconstruction: A process-oriented taxonomy, IEEE Trans. Softw. Eng. 35 (4) (2009) 573–591.

[74] H. Van Vliet, A. Tang, Decision making in software architecture, J. Syst. Softw. 117 (2016) 638–644.

[75] A. Ramírez, J.R. Romero, S. Ventura, A comparative study of many-objective evolutionary algorithms for the discovery of software architectures, Empir. Softw. Eng. 21 (2016) 2546–2600.

[76] D. Selva, B. Cameron, E. Crawley, Patterns in system architecture decisions, Syst. Eng. 19 (6) (2016) 477–497.

[77] L. Nallamothula, Selection of quantum computing architecture using a decision tree approach, in: Proceedings of the 3rd International Conference on Intelligent Sustainable Systems, ICISS, IEEE, Thoothukudi, India, 2020, pp. 644–649.

[78] A. Alreshidi, A. Ahmad, Architecting software for the internet of thing based systems, Future Internet 11 (7) (2019) 153.

[79] H. Muccini, K. Vaidhyanathan, Software architecture for ML-based systems: What exists and what lies ahead, in: Proceedings of 1st IEEE/ACM Workshop on AI Engineering - Software Engineering for AI, WAIN, IEEE, Madrid, Spain, 2021, pp. 121–128.

[80] M. Waseem, P. Liang, A. Ahmad, M. Shahin, A.A. Khan, G. Márquez, Decision models for selecting patterns and strategies in microservices systems and their evaluation by practitioners, in: Proceedings of the 44th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP, Pittsburgh, PA, USA, 2022, pp. 135–144.

[81] M. Shahin, P. Liang, M.R. Khayyambashi, Architectural design decision: Existing models and tools, in: Proceedings of the Joint 8th Working IEEE/IFIP Conference on Software Architecture and 3rd European Conference on Software Architecture, WICSA/ECSA, IEEE, Pittsburgh, PA, USA, 2009, pp. 293–296.

[82] A.A. Khan, M.A. Akbar, A. Ahmad, M. Fahmideh, M. Shameem, V. Lahtinen, M. Waseem, T. Mikkonen, Agile practices for quantum software development: Practitioners perspectives, 2022, arXiv preprint arXiv:2210.09825.

[83] V.R.B.G. Caldiera, H.D. Rombach, The goal question metric approach, Encyclopedia of Softw. Eng. (1994) 528–532.

[84] S. Easterbrook, J. Singer, M.-A. Storey, D. Damian, Selecting empirical methods for software engineering research, in: Guide to Advanced Empirical Software Engineering, Springer, 2008, pp. 285–311.

[85] V. Cosentino, J.L.C. Izquierdo, J. Cabot, A systematic mapping study of software development with GitHub, IEEE Access 5 (2017) 7173–7192.

[86] A. Barua, S.W. Thomas, A.E. Hassan, What are developers talking about? An analysis of topics and trends in stack overflow, Empir. Softw. Eng. 19 (2014) 619–654.

[87] M. Openja, M.M. Morovati, L. An, F. Khomh, M. Abidi, Technical debts and faults in open-source quantum software systems: An empirical study, J. Syst. Softw. 193 (2022) 111458.

[88] B. Zhang, T. Liu, P. Liang, C. Wang, M. Shahin, J. Yu, Architecture decisions in AI-based systems development: An empirical study, in: Proceedings of the 30th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER, IEEE, Taipa, Macao, 2023, pp. 616–626.

[89] M. Waseem, P. Liang, M. Shahin, A. Ahmad, A.R. Nassab, On the nature of issues in five open source microservices systems: An empirical study, in: Proceedings of 25th International Conference on Evaluation and Assessment in Software Engineering, EASE, ACM, Trondheim, Norway, 2021, pp. 201–210.

[90] J. Di Rocco, D. Di Ruscio, C. Di Sipio, P. Nguyen, R. Rubei, Topfilter: an approach to recommend relevant github topics, in: Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM, ACM, Bari, Italy, 2020, pp. 1–11.

[91] J. Cohen, A coefficient of agreement for nominal scales, Educ. Psychol. Meas. 20 (1) (1960) 37–46.

[92] J.L. Campbell, C. Quincy, J. Osserman, O.K. Pedersen, Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement, Sociol. Methods Res. 42 (3) (2013) 294–320.

[93] A.R. da Silva, D. Savić, Linguistic patterns and linguistic styles for requirements specification: Focus on data entities, Appl. Sci. 11 (9) (2021) 4119.

[94] K.-J. Stol, P. Ralph, B. Fitzgerald, Grounded theory in software engineering research: a critical review and guidelines, in: Proceedings of the 38th International Conference on Software Engineering, ICSE, IEEE, 2016, pp. 120–131.

[95] L.R. Hallberg, The "core category" of grounded theory: Making constant comparisons, Int. J. Qual. Stud. Health Well-Being 1 (3) (2006) 141–148.

[96] A. Koziolek, Automated Improvement of Software Architecture Models for Performance and Other Quality Attributes, vol. 7, KIT Scientific Publishing, 2014.

[97] A. Ahmad, M. Waseem, P. Liang, M. Fehmideh, A.A. Khan, D.G. Reichelt, T. Mikkonen, Engineering software systems for quantum computing as a service: A mapping study, 2023, arXiv preprint arXiv:2303.14713.

[98] C. Yang, P. Liang, P. Avgeriou, A systematic mapping study on the combination of software architecture and agile development, J. Syst. Softw. 111 (2016) 157–184.

[99] P. Ralph, E. Tempero, Construct validity in software engineering research and software metrics, in: Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering, EASE, ACM, Christchurch, New Zealand, 2018, pp. 13–23.

[100] A.R. Nasab, M. Shahin, S.A.H. Raviz, P. Liang, A. Mashmool, V. Lenarduzzi, An empirical study of security practices for microservices systems, J. Syst. Softw. 198 (2023) 111563.

[101] F. Tian, P. Liang, M.A. Babar, How developers discuss architecture smells? an exploratory study on stack overflow, in: Proceedings of the 16th IEEE International Conference on Software Architecture, ICSA, IEEE, Hamburg, Germany, 2019, pp. 91–100.

[102] F. Truger, J. Barzen, M. Bechtold, M. Beisel, F. Leymann, A. Mandl, V. Yussupov, Warm-starting and quantum computing: A systematic mapping study, 2023, arXiv preprint arXiv:2303.06133.

[103] M. Haghparast, T. Mikkonen, J.K. Nurminen, V. Stirbu, Quantum software engineering challenges from developers' perspective: Mapping research challenges to the proposed workflow model, 2023, arXiv preprint arXiv:2308.01141.

[104] P. Zhao, Z. Miao, S. Lan, J. Zhao, Bugs4Q: A benchmark of existing bugs to enable controlled testing and debugging studies for quantum programs, J. Syst. Softw. 205 (2023) 111805.

[105] X. Li, P. Liang, T. Liu, Decisions and their making in OSS development: An exploratory study using the hibernate developer mailing list, in: Proceedings of the 26th Asia-Pacific Software Engineering Conference, APSEC, IEEE, Putrajaya, Malaysia, 2019, pp. 323–330.

[106] T. Bi, P. Liang, A. Tang, Architecture patterns, quality attributes, and design contexts: How developers design with them, in: Proceedings of the 25th Asia-Pacific Software Engineering Conference, APSEC, IEEE, Nara, Japan, 2018, pp. 49–58.

[107] M. Shahin, P. Liang, M.R. Khayyambashi, Improving understandability of architecture design through visualization of architectural design decision, in: Proceedings of the 5th Workshop on Sharing and Reusing Architectural Knowledge, SHARK, ACM, Cape Town, South Africa, 2010, pp. 88–95.

[108] T. Liu, P. Liang, C. Yang, Z. Xiong, C. Wang, R. Li, Understanding the decision-making of students in requirements engineering course projects, in: Proceedings of the 1st Software Engineering Education Workshop (SEED) Co-Located with the 26th Asia-Pacific Software Engineering Conference, APSEC, CEUR-WS.org, Putrajaya, Malaysia, 2019, pp. 1–8.

[109] A.A. Khan, M.A. Akbar, A. Ahmad, M. Fahmideh, M. Shameem, V. Lahtinen, M. Waseem, T. Mikkonen, Agile practices for quantum software development: Practitioners' perspectives, in: Proceedings of the 2nd IEEE International Conference on Quantum Software, QSW, IEEE, Chicago, IL, USA, 2023, pp. 9–20.