Quantum Software Engineering: Roadmap and Challenges Ahead

JUAN MANUEL MURILLO, JOSE GARCIA-ALONSO, and ENRIQUE MOGUEL, University of Extremadura, Cáceres, Spain

JOHANNA BARZEN and FRANK LEYMANN, University of Stuttgart, Stuttgart, Germany

SHAUKAT ALI, Simula Research Laboratory, Oslo, Norway and Computer Science, Oslo Metropolitan University, Oslo, Norway

TAO YUE, Computer Science and Engineering, Beihang University, Beijing, China

PAOLO ARCAINI, National Institute of Informatics, Chiyoda-ku, Japan

RICARDO PÉREZ-CASTILLO, Faculty of Social Sciences and Information Technologies, University of Castilla-La Mancha, Talavera de la Reina, Spain

IGNACIO GARCÍA-RODRÍGUEZ DE GUZMÁN, Institute of Technologies and Information Systems, University of Castilla-La Mancha, Ciudad Real, Spain

MARIO PIATTINI, University of Castilla-La Mancha, Ciudad Real, Spain

ANTONIO RUIZ-CORTÉS, Computer Languages and Systems, University of Seville, Seville, Spain

ANTONIO BROGI, Department of Computer Science, University of Pisa, Pisa, Italy

JIANJUN ZHAO, Faculty of Information Science and Electrical Engineering, Kyushu University, Fukuoka, Japan

ANDRIY MIRANSKYY, Toronto Metropolitan University, Toronto, Ontario, Canada

MANUEL WIMMER, Johannes Kepler University Linz, Linz, Austria

As quantum computers advance, the complexity of the software they can execute increases as well. To ensure this software is efficient, maintainable, reusable, and cost-effective—key qualities of any industry-grade software—mature software engineering practices must be applied throughout its design, development, and operation. However, the significant differences between classical and quantum software make it challenging to directly apply classical software engineering methods to quantum systems. This challenge has led to the emergence of Quantum Software Engineering (QSE) as a distinct field within the broader software engineering landscape. In this work, a group of active researchers analyze in depth the current state of QSE research. From this analysis, the key areas of QSE are identified and explored in order to determine the most relevant open challenges that should be addressed in the next years. These challenges help identify necessary breakthroughs and future research directions for advancing QSE.

CCS Concepts: • **Software and its engineering**; • **Theory of computation Models of computation**;

Additional Key Words and Phrases: Quantum Software Engineering, open challenges, Quantum Computing, QSE

1 Introduction

Over the past two decades, quantum computing has rapidly transitioned from a theoretical concept to a burgeoning field of practical research and development. This transition has been driven by the advent of publicly accessible quantum computers, which have enabled researchers to experiment with algorithms that were previously confined to theoretical studies. Quantum algorithms have demonstrated the potential to solve certain problems in a reasonable timeframe that are beyond the capabilities of classical computers [160]. These algorithms also offer significant operational advantages, such as the perfect training of **Quantum Neural Networks (QNNs)** using only a few highly entangled training data points [97]. The immense potential of quantum computing has captivated industries, leading to increased investment and the commercialization of quantum computers by manufacturers such as D-Wave, IBM, IonQ, Rigetti, and Quantinuum.

The current focus in quantum computing is on developing more stable **Quantum Processing Units (QPUs)** to move beyond the **Noisy Intermediate-Scale Quantum (NISQ)** era [136]. NISQ computers, despite their limitations [88], have already demonstrated their utility by solving

of Seville, Seville, Spain; e-mail: aruiz@us.es; Antonio Brogi, Department of Computer Science, University of Pisa, Pisa, Italy; e-mail: antonio.brogi@unipi.it; Jianjun Zhao, Faculty of Information Science and Electrical Engineering, Kyushu University, Fukuoka, Japan; e-mail: zhao@ait.kyushu-u.ac.jp; Andriy Miranskyy, Toronto Metropolitan University, Toronto, Ontario, Canada; e-mail: avm@torontomu.ca; Manuel Wimmer, Johannes Kepler University Linz, Linz, Austria; e-mail: manuel.wimmer@jku.at.

problems through approximation methods, such as **Variational Quantum Algorithms (VQAs)** [25]. These advancements indicate that the era of industrialized quantum software is approaching, with the potential to revolutionize various fields by offering capabilities surpassing those of classical systems.

Nevertheless, historical lessons from software engineering reveal that the adoption of new technologies by the industry hinges on the ability to develop software in a repeatable, efficient, maintainable, reusable, and cost-effective manner [150]. Currently, quantum software lacks the development procedures and standards prevalent in classical computing. Therefore, integrating sound engineering principles into quantum software is crucial to bridging this gap [10, 133, 207].

Quantum Software Engineering (QSE) has been defined as "the use of sound engineering principles for the development, operation, and maintenance of quantum software[1]" [207]. One of the main challenges in QSE is to translate the knowledge and practices from classical software engineering to the quantum domain while also developing new methodologies tailored to match the unique requirements of quantum computing [168]. A key aspect of this challenge is the anticipated hybrid model of quantum software, which will blend classical and quantum computing [105, 193]. This hybrid approach necessitates the development of technologies that enable seamless interaction between different computing environments. Leveraging **Service-Oriented Computing (SOC)** principles to create interoperable interfaces and adopting Service Engineering methodologies will be critical for the effective design and management of quantum services.

In addition, testing quantum systems presents unique challenges due to the peculiarities of quantum states [9]. Developing specific practices for quantum software testing is essential to ensure reliability and correctness. Additionally, requirements engineering for quantum software, although currently less emphasized due to the limited number of real-world applications, will inevitably diverge from classical approaches [156, 201]. Understanding and addressing these differences will be crucial as quantum applications will become more prevalent.

In summary, the path to widespread industrial adoption of quantum computing is fraught with challenges, but the ongoing advancements in QSE indicate a promising avenue for overcoming these hurdles. By focusing on the development of sound engineering principles and practices tailored to quantum software, the research community can facilitate the transition from the NISQ era to a future where quantum computing will be an integral part of the technological landscape [15]. This journey will require concerted efforts in testing, requirements engineering, and the seamless integration of classical and quantum systems, ultimately leading to the realization of practical and scalable quantum software solutions.

This article aims to provide a comprehensive overview of the primary challenges arising in different areas of QSE. It highlights recent research developments in QSE and discusses the future challenges that researchers are likely to face in the coming years. By addressing these challenges, the field of QSE can advance significantly, paving the way for the development of robust, efficient, and maintainable quantum software. The future of quantum computing holds immense promise, and the progress in QSE will be instrumental in realizing the full potential of this revolutionary technology.

To this end, this article is structured as follows. Section 2 presents some key concepts of quantum computing. Section 3 provides a wide exploration of the research interest in the field of QSE. Section 4 shows a detailed review of the different research areas of QSE. Section 5 discusses the different key areas of software engineering and how they can influence quantum computing. Finally, Section 6 presents some implications and conclusions drawn from this research.

---

[1]In this work, "quantum software" refers to software that runs on a quantum computer. In a broader sense, because all computing on quantum computers involves classical parts, software that utilizes quantum computers is naturally hybrid.

2   Fundamentals of Quantum Computing

For those readers unfamiliar with quantum computing, this section introduces some basic concepts that will be useful for understanding the remaining sections. Quantum computing takes advantage of distinctive properties and processes rooted in the fundamental principles of quantum physics, setting it apart as a completely different paradigm from classical computing.

At the heart of quantum computing, there are key principles derived from quantum mechanics, including the use of quantum bits, or qubits, along with phenomena like superposition and entanglement. These quantum properties, along with many others, provide novel opportunities to enhance computational capabilities in ways that classical systems cannot achieve.

2.1   Qubit

In classical computing, the basic unit of information is the bit, which can take on the values of either 0 or 1. In quantum computing, however, we have the qubit (quantum bit), which serves as the fundamental unit of information.

A qubit can exist in a state of 0, 1, or a superposition of both simultaneously. This superposition enables quantum computers to perform complex computations more efficiently than classical computers. Mathematically, a qubit is often represented using Dirac notation, introduced by physicist Paul Dirac [69]. In this formalism, a qubit's state is expressed as a vector with two complex components, written as

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle. \tag{1}$$

In this equation, $|\psi\rangle$ is the state of the qubit, and α and β are complex numbers representing the probability amplitude of finding the qubit in the $|0\rangle$ and $|1\rangle$ basis states, respectively. These states are the computational basis of the qubit and are analogous to the classical 0 and 1.

In addition, the squared magnitude of the amplitudes α and β must sum to 1, which means that the probability of finding the qubit in one of the two states is 1. This can be represented as follows:

$$|\alpha|^2 + |\beta|^2 = 1. \tag{2}$$

2.2   Superposition

Superposition enables quantum computers to perform parallel computations in a single step, as a qubit in a superposition state can execute operations on all its possible states simultaneously [22]. For instance, if two qubits are in superposition, they can collectively represent four different states (00, 01, 10, and 11) simultaneously, whereas two classical bits could only represent one of these states at a time. As the number of qubits increases, the ability to represent multiple states grows exponentially, following the formula $States = 2^n$, where n is the number of qubits.

However, when a measurement is performed on a qubit in superposition, that is, when its value is observed, the superposition collapses. The qubit assumes one of its classical states (either 0 or 1), with the outcome determined by the probability amplitudes of each state. This phenomenon is a fundamental aspect of quantum computing: until a measurement is made, the qubit exists in a superposition of states, and only upon observation does it take on a definite value [85].

2.3   Entanglement

Quantum entanglement is a fundamental concept in quantum physics, crucial to the functioning of quantum computing. It describes a unique correlation between entangled particles, such as qubits, that are so deeply interconnected that the state of one qubit directly influences the state of the other, regardless of the distance between them. This phenomenon challenges classical notions of

independence between separate entities, where the state of one object does not affect another unless they are in direct interaction [71].

In this context, when two qubits become entangled, the state of one qubit is intrinsically linked to the state of its partner. Altering the state of one qubit immediately changes the state of the other, even if they are separated by vast distances. This non-locality enables the formation of entangled qubit pairs, which are critical for many quantum computational processes.

Entanglement also introduces the concept of non-local interactions, where measuring one qubit in an entangled pair instantly reveals information about the other qubit, regardless of their spatial separation. This property enhances the speed and efficiency of certain quantum algorithms to the point that, without entanglement, an exponential increase in speed cannot be achieved [81].

2.4 Quantum State Vector and Quantum Register

A quantum state vector (also simplified as a quantum state or state vector) is a mathematical representation that encapsulates the information about a quantum system. It contains all the information needed to predict the outcomes of measurements on the system.

In this context, a quantum state refers to the state of a quantum system as it evolves during the execution of a quantum algorithm. It describes the configuration of qubits at a particular point during the execution of the quantum program. To demonstrate these concepts, consider a quantum system consisting of two qubits. The state vector $|\psi\rangle$ for this system can be expressed as

$$|\psi\rangle = \begin{pmatrix} c_{00} \\ c_{01} \\ c_{10} \\ c_{11} \end{pmatrix} = \begin{pmatrix} c_{00} & c_{01} & c_{10} & c_{11} \end{pmatrix}^T.$$

In this case, each c_{ij} represents the amplitude corresponding to the basis state $|ij\rangle$. The basis states $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$ form a four-state orthonormal basis. For example, if the system is in the state $|\psi\rangle = (1,000)^T$, it means the amplitude for the state $|00\rangle$ is 1 and 0 for the remaining states $|01\rangle$, $|10\rangle$ and $|11\rangle$. Consequently, the probability of measuring each state on a computational basis is determined by the squared modulus of their corresponding amplitudes. In this case $|1|^2 = 1$ for state $|00\rangle$ and $|0|^2 = 0$ for the remaining states. Thus, the system has a 100% chance of being measured in the state $|00\rangle$ and no chance of being measured in the states $|01\rangle$, $|10\rangle$, or $|11\rangle$.

In more complex quantum systems, with more qubits involved, sets of qubits can be conceptually grouped into quantum registers. A quantum register is the quantum analogue of a classical register, playing a key role in storing and processing quantum information during a computation. In these complex systems, each quantum register can be represented by its own quantum state vector.

2.5 Circuit-Based Quantum Computing and Quantum Annealing

Currently, two different kinds of quantum computers are available for researchers interested in building quantum software: quantum annealers and circuit-based quantum computers. Quantum annealers specialize in solving optimization problems by finding the lowest energy state of a system. Circuit-based quantum computers are general-purpose quantum computers capable of running various algorithms. The following fundamental concepts are related to circuit-based quantum computers since, being general-purpose computers, they have gathered more interest from the QSE community. Nevertheless, quantum annealing works will also be analyzed and discussed during the rest of the article.
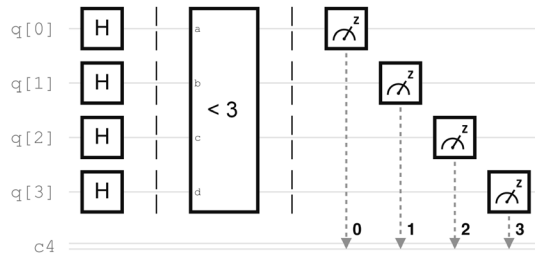
Fig. 1. Example of a gate-based quantum circuit.

2.6 Quantum Gate

Quantum gates are core components of quantum software, similar to classical logic gates. Unlike classical gates that work with bits, quantum gates manipulate qubits. They alter the quantum states of qubits by adjusting their probabilities of being measured in specific states or by creating entanglement between qubits. Mathematically, each gate is represented by a $2^n \times 2^n$ unitary matrix where n is the number of qubits affected by the gate. The action of a gate on a specific quantum system is determined by multiplying the state vector by the matrix representation of the gate.

Notable examples include the Hadamard gate for creating superposition (gate H in Figure 1), the Pauli-X gate (which flips qubit states), and the CNOT gate (for controlled operations creating entanglement between qubits). These gates are essential for executing quantum algorithms, enabling complex manipulations and transformations within a quantum system.

2.7 Quantum Circuit

A quantum circuit is a sequence of quantum gates arranged to manipulate qubits in a specific order, enabling the execution of quantum algorithms. It consists of multiple quantum gates, measurement operations, and potentially classical logic, all organized to perform a desired computation. Figure 1 illustrates an example of a gate-based quantum circuit. More specifically, the circuit shown in Figure 1 is a circuit that gives all numbers less than three for four qubits in a superposition state. Such a circuit yields a result (0000, 0001, and 0010).

Quantum circuits are graphically represented with qubits as horizontal lines, gates as symbols that indicate operations applied to these qubits, and modules encapsulating a set of gates like the oracle in the figure (oracle with the operation less than n). By altering the quantum states through operations like superposition and entanglement, a quantum circuit can perform complex computations that would be infeasible for classical circuits, making quantum circuits fundamental for implementing quantum algorithms.

To perform the computations they are designed for, quantum circuits are run (executed) on quantum machines. These quantum machines can be real quantum hardware or quantum simulators, and software systems can perform quantum computations using classical hardware. Quantum simulators can be ideal (meaning that quantum computations are executed perfectly) or include noise models to replicate the behavior of real quantum hardware. In any case, simulators are limited in terms of the number of qubits that can be simulated in classical hardware.

2.8 Quantum Algorithm

A quantum algorithm is a structured sequence of operations tailored to run on a quantum computer, leveraging quantum properties such as superposition or entanglement to solve computational problems that are infeasible or highly inefficient for classical computers. By manipulating qubits through quantum gates, these algorithms explore multiple solutions simultaneously and perform complex

operations in parallel, enabling them to tackle problems with significantly reduced computational complexity.

Quantum algorithms exploit the probabilistic nature of quantum mechanics to achieve speedups over classical methods for certain types of tasks. For example, Shor's algorithm [163] provides an exponential speedup for factoring large integers, reducing the time complexity from sub-exponential (classical) to polynomial (quantum). It does this by converting the problem into a period-finding task and using the Quantum Fourier Transform [194] to efficiently determine the periodicity of a function, a critical step in factorization. This capability threatens encryption systems like RSA, which rely on the classical difficulty of factoring large numbers, showcasing quantum computing's potential to disrupt current cryptographic standards.

Similarly, Grover's algorithm [63] achieves a quadratic speedup for unstructured search problems. Classical search methods require $O(N)$ time to find an item in an unsorted database, but Grover's algorithm reduces this to $O(\sqrt{N})$ using amplitude amplification. By encoding all possible solutions into quantum superpositions and iteratively amplifying the correct solution's amplitude, the algorithm identifies the solution in far fewer steps. Although the speedup is not exponential, it provides a meaningful advantage for large-scale search problems, highlighting quantum computing's utility in domains where classical approaches face scalability challenges.

Together, these algorithms illustrate the transformative potential of quantum computing in solving problems more efficiently than classical methods.

2.9 Quantum Oracle

An oracle is a piece of circuit performing a function that, applied to a quantum state, produces as output the same quantum states with some transformations. This output will be used as input for another circuit that implements some algorithm. Thus, oracles can be thought of as a black box performing a function that is used as an input by another algorithm constituting a pattern for quantum algorithms [87]. Oracles appear in many well-known quantum algorithms like Grover [64], Deutsch-Jozsa [37], Simon [164], or Bernstein-Vazirani [19].

2.10 Quantum Noise

Noise in quantum computers and simulators refers to any unwanted interaction or imperfection that disrupts the delicate quantum states used for computation. Quantum systems rely on phenomena like superposition and entanglement, which are inherently fragile and highly sensitive to external influences. Noise manifests as errors in quantum computations, leading to incorrect results or the loss of coherence in quantum states. The sources of noise in quantum systems can be broadly categorized as follows:

— *Environmental Noise*. Quantum systems are highly susceptible to interactions with their surroundings, such as electromagnetic radiation, thermal fluctuations, and vibrations. These external disturbances can cause decoherence, a process where qubits lose their quantum properties and revert to classical states.
— *Gate Errors*. Quantum gates are not perfect, and inaccuracies during their execution introduce gate errors. These arise due to hardware imperfections, such as imprecise control over qubits, timing issues, or limitations in the precision of control signals.
— *Measurement Errors*. Reading the state of a qubit (measurement) is another source of noise. Due to the probabilistic nature of quantum mechanics and imperfections in the measurement apparatus, the observed state may not accurately reflect the actual state of the qubit.
— *Cross-Talk*. In multi-qubit systems, operations on one qubit can inadvertently influence neighboring qubits due to unintended coupling or interference, leading to errors.

—*Qubit Relaxation and Dephasing*. Qubits have finite coherence times, meaning they can only maintain their quantum states for a limited duration. Relaxation occurs when a qubit spontaneously transitions to its ground state, while dephasing refers to the loss of relative phase information between superposed states.

On the other hand, in quantum simulators, noise can be artificially introduced to mimic the imperfections of real quantum hardware. This is essential for testing and developing error mitigation and correction strategies, which aim to counteract the effects of noise in practical quantum computing.

3 Research Interest in QSE

QSE has evolved into a distinct discipline within the broader field of software engineering. First mentioned in 2002, QSE was identified as one of the grand challenges in computer science research. Clark et al. described it as "the development of a full discipline of QSE, ready to exploit the full potential of commercial quantum computer hardware, once it arrives, projected to be around 2020" [29].

Although the early motivation for developing QSE was driven by the anticipated advancements in quantum hardware, it took several years before quantum hardware and simulators became available to the public. A significant milestone was reached in 2016 when IBM released its first gate-based quantum computer along with an online simulator [74]. This breakthrough was soon followed by other vendors, catalyzing the growth and development of QSE.

Since the early developments in QSE, several conference venues have emerged that significantly contribute to the growth of the field. While not all conferences focus solely on QSE, many include papers on the topic, reflecting the growing interest and research in this area.

One such notable event is the *International Workshop on Quantum Software Engineering*. This workshop, which celebrated its fifth edition this year, is collocated with the *ACM/IEEE International Conference on Software Engineering*. Another important conference is the *IEEE International Conference on Quantum Software*. Now in its third edition, this conference is organized under the umbrella of the *IEEE World Congress on Services*, providing a platform for discussing advancements in quantum software. Additionally, the *International Workshop on Quantum Software Engineering and Technology* is a significant event within the *IEEE Quantum Week*. This workshop focuses on advancing the field of QSE by bringing together researchers and practitioners to share their latest findings. The *IEEE International Conference on Quantum Computing and Engineering*, also part of *IEEE Quantum Week*, is another prominent venue that addresses various aspects of quantum computing, including software engineering.

These conferences and workshops provide essential platforms for researchers and practitioners to share advancements, discuss challenges, and foster collaboration in the field of QSE. Through these gatherings, the community can collectively push the boundaries of what is possible with quantum software, paving the way for future innovations.

On the other hand, the interest in QSE extends deeply into the realm of academic journals, where the importance of this emerging discipline is highlighted through dedicated issues and sections. Among these, the *IEEE Transactions on Quantum Engineering* and *ACM Transactions on Quantum Computing* are prominent journals that cover a broad spectrum of topics, including QSE. Furthermore, Elsevier *Journal of Systems and Software* and *Information and Software Technology* have recognized the growing significance of QSE by introducing special issues specifically dedicated to this area. Additionally, *ACM Transactions on Software Engineering and Methodology* features a Continuous Special Section on QSE.
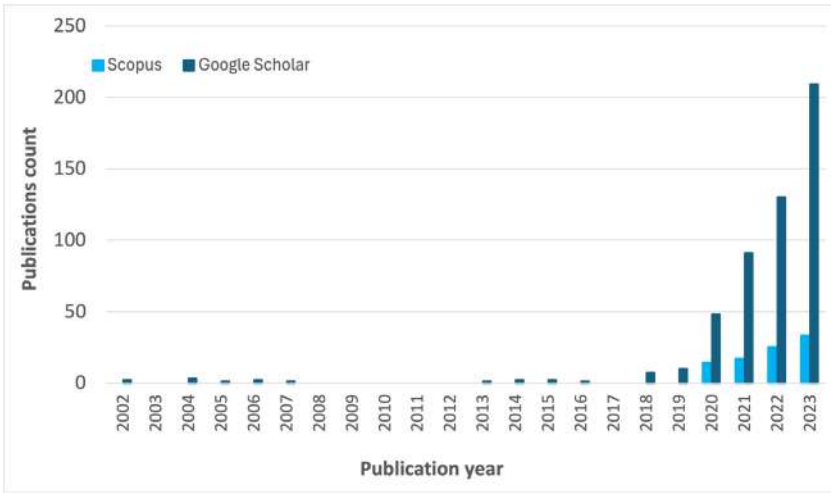
Fig. 2. Papers on QSE over time.

The inclusion of QSE in these prestigious journals underscores the field's growing importance and the need for rigorous academic inquiry. These publications serve as critical venues for researchers to share their findings, discuss emerging trends, and collaborate on addressing the complex challenges associated with developing robust and effective quantum software. Through these journals and special issues, the body of knowledge in QSE continues to expand, driving the field forward.

The rising interest in QSE can be indirectly measured by analyzing the number of publications containing the term "QSE," taking into account as inclusion criteria all types of papers, proceedings, books, technical reports, and so on. Figure 2 illustrates the annual number of such publications indexed in Scopus and Google Scholar.[2] Some initial publications appeared between 2002 and 2007, highlighting the early recognition of the need for QSE in the emerging quantum computing domain. A resurgence of publications began around 2013, primarily driven by the needs identified by quantum computing scientists during the development of quantum software. The availability of quantum simulators and platforms around 2020 led to a significant increase in QSE publications, with more than 200 publications recorded in 2023.

Focusing on the papers found in the Scopus database, probably more mature than its counterpart in Google Scholar, of the 102 papers identified up to 31 July 2024, 70 (68.6% of the total) were conference publications, 26 (25.5%) were journal articles, and 6 (5.9%) were book chapters. The data clearly indicates that conferences are the main venue for the publication of research papers in QSE, although it should be emphasized that of the 26 journal publications, 10 works (more than a third of the total) have appeared in the first half of 2024.

Once all these works have been analyzed and classified by the different areas of knowledge in the field of QSE, we obtain the results shown in Figure 3. It is noteworthy that the topics with the highest number of publications are "Quality Assurance" with 16 publications and "Service-Oriented Computing" with 12 publications. We also find the topics "Programming Paradigms" and "Software Development Processes" both with 9 works, "Model-Driven Engineering" with 8 publications, "Software Architectures" with 7 articles, and "Artificial Intelligence" with 3 publications. It is also important to note that from this analysis we have found 20 review publications (surveys, systematic literature reviews, systematic mapping studies, and other types of reviews). In addition,

---

[2]For replication purposes, the specific publications considered are listed in https://doi.org/10.5281/zenodo.13839576.
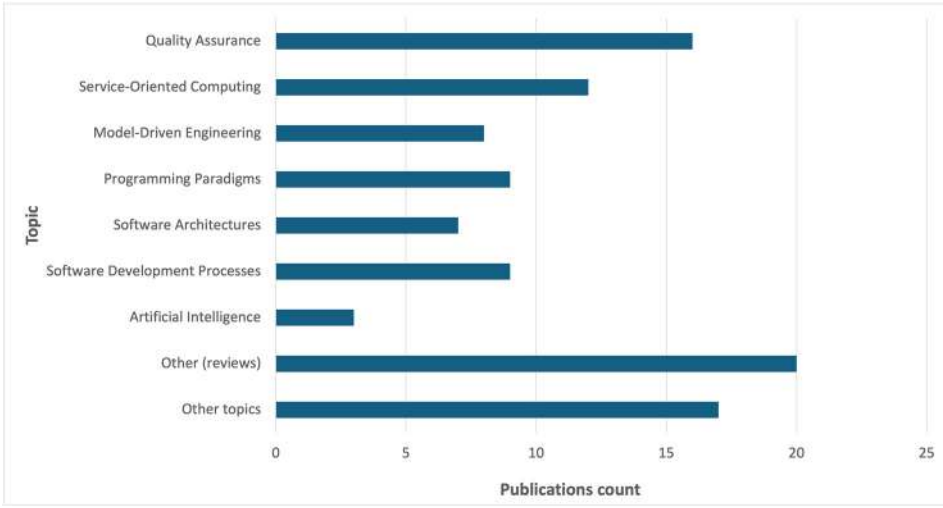
Fig. 3. Number of publications by topic.

we identified 17 papers on topics of less relevance to QSE such as communications, cybersecurity, or requirements engineering.

4 QSE Active Research Areas

This section examines some of the research conducted in different areas of QSE. We do not aim to provide a systematic analysis. Instead, we strive to gather QSE works and opinions from active researchers in the field regarding the most relevant challenges they anticipate in QSE for the upcoming years.

4.1 Quality Assurance: Testing and Debugging

Quantum software testing is about assessing the correctness of quantum software in delivering their intended functionalities, while debugging is about observing failures in quantum software that need to be diagnosed and fixed, especially when facing challenges such as quantum noise, limited quantum hardware and simulator resources, and limited observability [10].

Several groups have made notable contributions to test quantum software in different ways [94, 105–107, 199]. Various coverage criteria for assessing the quality of test cases have been proposed, including covering inputs and outputs of quantum programs [6, 184], coverage criteria based on equivalence class partitioning [94], and multi-granularity coverage criteria for QNNs [161]. In addition, several advanced techniques have been applied to test quantum programs including search-based testing [183, 185], combinatorial testing [182, 186], metamorphic testing [2], fuzzing [177], property-based testing [70], concolic testing [198], and also techniques based on the reversibility property of quantum circuits [33]. Moreover, techniques for testing quantum computing platforms have also emerged, such as those based on metamorphic testing and differential testing [125, 178]. Various assertion types have been suggested to check program output correctness, such as statistical assertions [72], projection-based assertions [89], and dynamic assertions [91, 92, 216]. Moreover, as quantum computer noise makes it difficult to distinguish actual bugs from program failures caused by noise, efforts have recently been made to mitigate noise in quantum computing to enhance testing reliability (e.g., see [116]). A testing framework [94] has also been developed to support

both unit and integration testing of quantum programs. Relations between quantum programs are also leveraged to address the test oracle[3] problem in quantum software testing [93].

Mutation testing is an efficient approach to assess the quality of a test suite. Different works have proposed mutation operators for quantum programs, like Muskit [100] and QmutPy [44–46]. Moreover, an empirical evaluation was conducted on over 700,000 mutants generated from more than 350 real quantum circuits [174], and the generation approach MutTG was proposed for creating minimal-sized test suites capable of killing a set of mutants [188].

An empirical analysis of software repositories containing quantum programs is underway, which leads to the identification of bug patterns (some of which are quantum-specific) affecting quantum software [80, 124, 211, 213, 214]. These patterns can be used by newly created static analyzers for quantum code, such as [84, 119, 126, 197, 212], to detect defects early in the development process. Additionally, they can be used to create benchmarks for testing and debugging [211]. Quantum software bug trackers reveal flaky tests (which is intuitive given quantum mechanics' probabilistic nature), but the root causes of flakiness differ from classical software [165, 203, 206]. Transformer-based models (ranging from CodeBERT [42] to **Large Language Models (LLMs)**) are being explored to classify bug reports and detect defects in quantum bug repositories by leveraging information from bug reports, change requests, and source code [165]. Quantum-specific technical debt [122] and code smells [26] are being analyzed and detected, and software repositories are used to validate the findings empirically. Lastly, software repositories can be used to identify a list of developers to be interviewed, e.g., to assess their background [162] or elicit code smells [26]. Formal methods, such as the formalization of **Quantum Intermediate Representation (QIR)**, also contribute to validating program safety and detecting unsafe operations [95, 96]. Moreover, an investigation into automatic quantum program repair has begun, demonstrating the effectiveness of LLMs, specifically with OpenAI's GPT-3, in repairing some quantum programs [66]. Li et al. [90] proposed a framework for repairing quantum programs using unitary operations. By systematically generating and validating patches, the framework addresses defects in quantum programs while ensuring correctness through algebraic principles. Additionally, Yu and Zhao [200] introduced the Quantum Program Dependence Graph, which effectively models both quantum-specific and classical dependencies in quantum programs. This framework facilitates slicing and dependency analysis, aiding in the debugging, testing, and understanding of quantum programs.

Debugging involves analyzing programs to identify and correct errors through causal analysis between bugs and detected errors, crucial for both classical and quantum programming [18, 27]. Miranskyy et al. [106, 107] examined quantum program debugging tactics, exploring the adaptation of classical strategies—brute force, backtracking, and cause elimination—to quantum contexts. Metwalli and Meter [101] proposed a debugging framework for quantum circuits, concentrating on amplitude permutation, phase modulation, and amplitude redistribution circuit blocks, addressing the need for specialized approaches for each quantum circuit type, contributing toward robust quantum computing systems. Sato and Katsube [157] introduced four considerations for locating bugs in quantum programs and an efficient bug-locating method integrating cost-based binary search, early determination, finalization, and retrospection, validated by experimental results. Recently, delta debugging has been explored in the context of regression testing of quantum programs [135] with a specific focus on property-based testing.

In quantum program debugging, implementing quantum assertions often involves duplicating quantum variables and repeated measurements. Huang and Martonosi [72] used hypothesis testing to partially reconstruct quantum register information from measurements for assertions, noting a lack of run-time assertion support. To overcome this, Liu et al. [92] proposed using additional

---

[3]This should not be confused with the definition of oracle seen in Section 2.9 above.

qubits to capture assertion-target qubits' information. Furthermore, Li et al. [89] introduced Proq, a projection-based run-time assertion tool, utilizing projection measurements to enable assertions without destroying quantum variables. Additionally, Jin and Zhao proposed ScaffML [79], a behavioral interface specification language for the quantum programming language Scaffold [1], which specifies pre- and post-conditions for Scaffold modules to support the debugging and testing of Scaffold programs. The researchers have also realized the need to compress program specifications often used to support automated testing. For instance, Oldfield et al. [121] proposed an algorithm to compress program specifications to improve testing efficiency.

Given the noise on current quantum computers, testing quantum programs on real quantum computers or their corresponding noisy simulators poses additional challenges. For instance, a tester may be unable to reliably determine whether a test case failed a program or was due to noise in quantum computations. To this end, a set of approaches were recently proposed [113–116]. Muqeet et al. [115, 116] proposed two machine learning-based approaches to learn noise patterns as machine learning models for a given quantum computer or noisy simulator. Such machine learning models are then used to filter out noise from the outputs of quantum programs, thereby improving testing reliability. Moreover, to enable the use of existing error mitigation techniques implemented on IBM's quantum computers, an approach has been proposed by Muqeet et al. [114]. Naturally, the approach can be used to support testing on quantum computers. Finally, Muqeet et al. [113] proposed an approach that employs genetic programming to develop noise models for quantum computers; such models can be used for testing quantum programs in simulators.

Though all the above work is contributing to good results in software testing, we can identify open challenges and opportunities that will need to be addressed in the next years, some of which are given below:

Challenges in Quantum Software Testing

Ch-ST-1. Efficient test oracles.
Ch-ST-2. Test scalability.
Ch-ST-3. From simulators to real quantum computers.
Ch-ST-4. (Quantum) AI and (quantum) software testing.

Efficient Test Oracles. Though some test oracles exist to assess whether test cases pass or fail, most of them are expensive to compute [6, 72, 89, 93, 184]. In addition, sometimes the test oracle must be generated for each given input-output, which increases the execution cost of the testing process [34]. Given the scarce and expensive quantum computing resources, such test oracles do not scale. Thus, more efficient test oracles are needed. Theoretically, one could use state vectors instead of multiple executions, but one cannot easily obtain such state vectors from real quantum computers. Quantum state tomography and partial tomography can give us an approximation of the state vectors, but they are computationally expensive [21, 167]. In addition, there is also no efficient support (adding dynamic assertions is cumbersome) for checking quantum program states at various points during execution. Such checks, like in classical computing, are important, also for debugging and repair. Projection-based oracle [89] is one approach, but it still requires reading quantum programs, albeit it aims at reducing the times reading is required. Near future research directions include conducting empirical studies to devise guidelines for conducting experiments and selecting appropriate statistical techniques to keep the balance of required runs and the reliability of results. Another research direction aligns with classical test optimization: prioritizing testing on critical test cases. In this sense, the use of metrics and the development of quality models may be of great use in improving various aspects of the quantum software testing process [30, 41, 208].

Additionally, exploring the conversion of testing tasks into common relations between quantum states or programs offers a valuable avenue. For example, many testing tasks can be transformed into checking the equivalence of two quantum programs or verifying whether the target quantum program is identity or unitarity [93]. In this regard, discovering more topical relations between quantum states and programs would be valuable.

Test Scalability. Most existing test data generation approaches are based on initial classical states of quantum programs (e.g., [184, 185]). QuraTest [199], and the work presented in [94], goes beyond classical test input generation. However, the current works do not scale to complex quantum software and have challenges in efficiently covering the high-dimensional test input space determined by the number of qubits, entangled and superposition states, and so on. Thus, we foresee advanced test data generation methods that lead to the efficient detection of faults in non-trivial quantum programs (e.g., faulty superpositioned states, wrong gates applied), considering practical constraints such as limited quantum computing resources and the diversity and representatives of the test data as we typically do in classical software testing.

From Simulators to Real Quantum Computers. Currently, quantum computers are noisy; hence, test results are unreliable. Most current works test on ideal simulators (e.g., [2, 70, 93, 94, 177, 183, 185, 186, 199]). Thus, we foresee the need for noise reduction techniques to be integrated as part of quantum software testing to increase the reliability of test results. Moreover, due to limited access to real quantum computers, existing testing solutions mainly rely on simulators. However, in the future, when real quantum computers become more accessible, it is crucial to test quantum programs on real quantum computers. Thus, to be cost-effective, test optimization would be needed [40]. Depending on the scale of testing, either classical test optimization or quantum test optimization approaches could be developed to minimize or prioritize tests. To this end, several preliminary works apply quantum annealing and **Quantum Approximate Optimization Algorithms (QAOAs)** to optimize test cases for classical software [181, 187], which, we believe, is equally applicable to test optimization of quantum software.

(Quantum) AI and (Quantum) Software Testing. There is a potential to use classical AI for quantum software testing and apply quantum AI to support classical software testing (for more details on the intersection between AI and QSE, also see the *Ch-AI* challenges in Section 4.7). Classical AI techniques can support various quantum software testing and debugging. For example, there is potential to explore whether LLMs can be used to assist in repairing quantum programs by providing patch recommendations [66]. Moreover, LLMs can also be used during mutation analysis to obtain suggestions for generating realistic mutants for a given quantum program. Furthermore, classical AI techniques such as LLMs can be explored to help in determining appropriate test oracles and generating test cases by leveraging their ability to comprehend the semantics of quantum programs. For further discussion, see Section 4.7.

Quantum AI, on the other hand, has the potential to be used to empower classical software testing. For instance, quantum extreme learning machines have been explored to improve the efficiency of regression testing of industrial elevators compared to classical machine learning [180]. QAOA has also recently been applied to solve test case optimization problems [181], in addition to quantum annealing for classical test case minimization [187].

These applications highlight the potential of applying quantum algorithms, in general, to improve various classical software engineering tasks (e.g., Grover's algorithm [64] has been utilized to speed up dynamic software testing for classical computers [102]). The key of such applications is to ensure that a practical software engineering task can be formulated into a mathematical problem that can be efficiently solved on a quantum computer [103]. Numerous use cases span the entire software development lifecycle, from planning and requirements engineering to testing and maintenance [103].

Other Aspects of Testing. Additionally, other aspects of quantum software testing are worth discussing. Concerning mutation testing, different problems remain open. First, it is necessary to understand whether artificial faults really represent real faults. Empirical studies are also needed to discover existing subsumption relations among mutation operators due to the large number of mutants that can be produced. Further research is also needed to identify similarities and differences between classical and quantum software artifacts, guide the development of new tools or reuse existing ones, identify common pain points, and devise best practices through mining software repositories [35], and so on. Refactoring has been proposed to improve code maintainability in quantum software. Recent work introduced methods for applying refactorings specific to quantum programs, focusing on managing quantum-specific dependencies and optimizing circuit modularity [209]. Moreover, for quantum software, mapping testing techniques and activities to specific test phases is challenging [107]. Formalizing this mapping for unit and integration testing has been performed [94], but other phases require more work. To guide testing practices, seven testing principles have been proposed to guide quantum software testing [94]. These principles emphasize representative and verifiable quantum input states, systematic partitioning of input variables by state types, and criterion-based input selection for comprehensive testing coverage. Additionally, ensuring clear methods for input preparation and output verification, maintaining endian mode consistency, and providing tools with key functionalities such as subroutine composition and result analysis are critical for effective testing. In addition, it is necessary to apply techniques that assist in the replicability and verifiability of quantum software experiments. To this end, there are already proposals for reporting guidelines and delineating a laboratory package structure adapted to quantum computing experiments [109].

4.2 SOC

In classical software engineering, SOC is defined as a paradigm to support the development of interoperable, distributed applications where services are regarded as autonomous entities [127]. In the last few years, research in this field has covered a range of topics, including the provisioning and aggregation of basic services [175], the composition of complex systems through the coordination of services [13], and the management and monitoring of services deployed in the cloud or on-premise [144].

Quantum hardware and software offered in the cloud allow users to access quantum computing resources over the Internet without the need to own quantum computing hardware. Several companies and institutions provide cloud-based quantum computing services like IBM already mentioned above, or others like Amazon Braket or Microsoft through Azure Quantum. This makes it easier for a broader audience to explore and develop quantum software. These platforms offer access to a variety of quantum processors and simulators, along with tools and environments to create, run, and test quantum algorithms.

Employing classical SOC principles in this scenario may look easy. Nevertheless, given the incipient state of quantum hardware technology, classical SOC approaches are not directly transferable to quantum software development [147]. The distinct and complex interaction model required for quantum computing challenges the straightforward application of conventional SOC frameworks, so it is necessary to re-evaluate how these principles can be adapted or redefined for the field of quantum computing [110]. To address these difficulties, several research groups have recently focused on **Quantum Service-Oriented Computing (QSOC)**, intending to bring its benefits to QSE [108].

Wild et al. [196] presented one of the first approaches in this regard. In this work, the authors presented TOSCA4QC, introducing two deployment modeling styles based on the Topology and Orchestration Specification for Cloud Applications standard for automating the deployment and

orchestration of quantum applications. From there, this research group has provided several contributions in the Quantum SOC domain. Also, the work done by Weder et al. [191] focuses on the need for additional mechanisms for orchestration in hybrid quantum applications over classical applications. On the other hand, Beisel et al. [17] provide a service ecosystem for the execution, based on workflows, of VQA. VQA is a hybrid quantum-classical approach that uses parameterized quantum circuits and classical optimization techniques to solve complex optimization and machine learning problems on near-term quantum devices [25].

Also, Weder et al. [189] introduce a method to detect parts of a hybrid workflow that can be optimized at runtime. These works, as a whole, provide a formalized and complete toolkit to bring the benefits of working with workflows for the deployment and execution of hybrid quantum applications.

From a different perspective, the work done by Rojo et al. [143] analyzes the problems and difficulties of developing hybrid quantum applications using a service-oriented approach. From there, this research group has also provided several contributions in this domain. Also, Garcia-Alonso et al. [12, 53] proposed adapting the API Gateway pattern to be combined with quantum software. And Romero-Álvarez et al. [145] extend the Open API standard to develop quantum services. Another work by Romero-Álvarez et al. [146] focuses on using DevOps techniques to enable the continuous deployment of quantum software. Finally, Bisicchia et al. [20] propose a technique for distributing quantum computations between different computers by splitting the shots required for a given quantum task among different QPUs. All these works provide an approach that brings the development of hybrid quantum applications closer to the tools and techniques usually employed in SOC.

These examples show the potential benefits of applying the principles of SOC to QSE. However, to exploit the full potential of service orientation, several challenges must be addressed first. More specifically:

Challenges in QSOC

Ch-SoC-1. Interoperability.
Ch-SoC-2. Platform independence.
Ch-SoC-3. Demand and capacity management.
Ch-SoC-4. Workforce training.

Interoperability. Facing this challenge is crucial as the quantum computing landscape continues to expand, with various technologies and platforms emerging across the industry. The lack of standardized frameworks hinders seamless interaction between different QPUs, software stacks, and development environments. The need for universally accepted standards, either through industry initiatives or research-driven proposals, is becoming increasingly evident.

One notable example is OpenQASM (Quantum Assembly Language), standardized by several companies, highlighting the benefits of unifying quantum programming languages. OpenQASM allows developers to write instructions compatible with multiple quantum computers, demonstrating how standardization can simplify the development process and reduce fragmentation. However, despite this advancement, there are still numerous assembly-level languages in use, each tied to specific hardware or platforms, creating a barrier to widespread quantum software development.

To address this challenge, ongoing efforts are focusing on creating intermediate representations, such as the QIR, which can act as a bridge between high-level programming languages and low-level hardware-specific instructions. QIR allows quantum programs to be hardware-agnostic, enabling

code to be executed across different quantum platforms without the need for reimplementation. This not only facilitates portability but also enhances scalability as quantum hardware evolves.

In addition to programming languages, there is a pressing need to create standardized APIs to interface with QPUs. A consistent set of APIs would enable developers to build quantum applications without needing to understand the specifics of each quantum device deeply. This would significantly improve the interoperability of quantum systems and enable the development of tools and solutions aligned with QSOC. Standard APIs could allow services to interact seamlessly across different QPU architectures, fostering collaboration and allowing for more sophisticated quantum cloud services, cross-platform quantum software development, and distributed quantum computing. By establishing these standards, the quantum computing community can ensure that future quantum technologies are accessible, interoperable, and scalable across diverse platforms.

Platform Independence. This challenge is a key issue in the rapidly evolving quantum computing ecosystem, where the capabilities of QPUs vary widely across different vendors. The environments through which these QPUs are accessed (often via cloud-based platforms) introduce further variability. Some platforms offer hybrid runtimes that integrate classical and quantum computing, while others provide software optimizations and performance enhancements that are specific to their own architecture. This diversity in features, while beneficial in some ways, creates challenges for quantum software developers, who must often tailor their applications to specific platforms. As a result, there is a significant dependency on the platform for which the software is designed, increasing the risk of vendor lock-in where developers are constrained to a particular hardware provider, limiting flexibility and innovation.

In classical SOC, platform independence has been a driving force for the scalability and success of cloud-based services. By abstracting the underlying hardware, developers can deploy applications on a variety of systems without having to modify their code for each platform. Achieving a similar level of platform independence in quantum computing is crucial to fostering a more open and competitive ecosystem where developers can choose the best tools and services without being tied to a specific vendor's hardware or cloud environment.

To realize platform independence in quantum computing, intermediate layers that abstract away the details of specific QPUs are needed. This could involve the development of universal quantum compilers that translate high-level quantum programs into intermediate representations compatible with multiple hardware platforms. Standards such as QIR (already mentioned above) as well as efforts to create vendor-neutral quantum **Software Development Kits (SDKs)** could serve as key building blocks for this. These SDKs would enable developers to write quantum applications that run across various quantum hardware without the need for significant reconfiguration or optimization.

Demand and Capacity Management. This challenge in quantum computing is complex, particularly when supporting hybrid workflows that integrate both classical and quantum resources. Effective coordination between these systems is essential, as quantum computations often rely on classical pre-processing and post-processing steps. Additionally, optimizing data transfer and communication between classical and quantum systems is crucial for maintaining efficiency, as quantum computers may be physically remote and subject to network latency or bandwidth constraints. These factors make it imperative to design infrastructure that seamlessly manages the interaction between classical and quantum resources to avoid bottlenecks.

Quantum hardware introduces further complexity into capacity management due to its inherent limitations, such as coherence times (the duration for which a qubit remains in a superposition), gate fidelities (the accuracy of quantum operations), and qubit connectivity (the ability of qubits to interact with one another within a quantum processor). Capacity planning for quantum services must consider these constraints, which directly impact the performance and scalability of quantum

computations. The number of available qubits, their quality, and the complexity of quantum circuits that can be executed in a single run are key factors that must be continuously monitored and optimized.

A significant challenge lies in adapting existing demand and capacity management strategies, particularly from microservice architectures, to quantum computing. In traditional microservice-based systems, capacity is often analyzed and managed with third-party services in mind [48]. In scenarios where these services are quantum, however, new considerations emerge—such as the limited availability of quantum processing time, the need for queuing systems to manage access to quantum hardware, and the potential cost implications of quantum computing resources. For example, the latency and reliability of third-party quantum services will differ drastically from classical services, necessitating new models for assessing service-level agreements and ensuring that capacity is dynamically allocated to meet fluctuating demand.

Moreover, the role of pricing plans [49] and capacity limitations in the API industry [52] will take on new significance as quantum services become commercially available. Pricing models for quantum computing are typically based on the amount of time spent on a quantum processor, the number of qubits used, and the complexity of the quantum circuits. These factors introduce new challenges for capacity management, as developers must balance computational requirements with cost efficiency. As quantum APIs become more integrated into broader service architectures, organizations will need to develop sophisticated pricing and capacity strategies that account for the unique characteristics of quantum computing. This will likely include tiered pricing models based on quantum hardware capabilities, the development of more granular usage metrics, and dynamic capacity allocation based on real-time demand.

Workforce Training. This challenge is critical as the industry transitions toward quantum and hybrid software development. As such, this challenge could be included in many other research areas discussed in this section. It has been included here due to the pervasive presence of SOC in industry projects. While the principles of SOC are well-understood and widely practiced by a large community of developers, introducing quantum computing into this paradigm requires a substantial shift in both mindset and skills. Developers accustomed to classical **Service-Oriented Architecture (SOA)** must be retrained to navigate the complexities of QSOC. The challenge here is multifaceted, requiring not only new technical knowledge but also the ability to integrate quantum principles with existing classical architectures.

Addressing this challenge involves a two-pronged approach. The first step is to define comprehensive training strategies that facilitate the smooth transition from classical to quantum development. These strategies should include specialized educational programs, certifications, and hands-on experience with quantum technologies. Training initiatives must be designed to bridge the gap between classical and quantum computing by focusing on core quantum concepts such as qubits, superposition, entanglement, and quantum algorithms while also teaching developers how these concepts integrate with classical computing frameworks. Moreover, training should be structured progressively, beginning with foundational quantum mechanics and programming languages like Qiskit or Cirq and moving toward more advanced topics like hybrid quantum-classical workflows and quantum cloud services. The end goal is to make quantum technologies accessible to traditional developers, easing the learning curve and fostering broader adoption across industries.

The second aspect of addressing workforce training is the development of QSE methodologies and tools that streamline the transition from classical SOC to its quantum counterpart. These methodologies should provide a systematic framework for building, testing, and deploying quantum services within hybrid systems. For instance, tools that abstract the complexities of quantum hardware and offer familiar service-oriented interfaces will be crucial in helping developers adopt quantum technologies without needing deep expertise in quantum mechanics. By focusing on

interoperability between classical and quantum components, these tools can reduce the cognitive load on developers, allowing them to build quantum-enhanced applications with minimal disruption to their existing workflows.

Additionally, collaborative environments and quantum development platforms should be created to foster hands-on learning. Quantum simulators, cloud-based quantum platforms, and **Integrated Development Environments (IDEs)** that support hybrid systems will be key to giving developers practical experience with quantum technologies. These platforms can provide sandbox environments where developers can experiment with quantum services, test quantum circuits, and integrate quantum capabilities into classical service-oriented applications.

Beyond technical training, there must also be an emphasis on reshaping the development culture to accommodate quantum thinking. In classical SOC, concepts like modularity, scalability, and interoperability are well established, but in quantum computing, additional considerations such as quantum error correction, decoherence, and probabilistic outcomes are crucial. Developers need to be educated not only in the technical aspects of quantum computing but also in the unique challenges and opportunities it presents. Workshops, conferences, and community-driven learning initiatives can be valuable in fostering a collaborative culture where knowledge sharing and continuous learning are prioritized.

4.3 Model-Driven Engineering (MDE)

MDE [158] has been historically significant in the development of classical software by providing methodologies and tools to manage complexity through high-level abstractions and automation. MDE involves creating abstract models that represent the system's structure, behavior, and functionality, allowing for a clear and concise specification of requirements and design. These models are often defined using **Domain-Specific Modeling Languages (DSMLs)** tailored to particular domains, enhancing expressiveness and reducing ambiguity. MDE facilitates model validation and verification through simulation and formal methods, ensuring that the software adheres to the specified requirements before implementation. One of MDE's key strengths is automatic code generation, where high-level models are transformed into executable code, reducing manual coding effort and minimizing errors. This automation extends to various phases of software development, including analysis, design, implementation, and testing. By promoting a model-centric approach, MDE improves the consistency, traceability, and maintainability of software systems, enabling better alignment with business goals and more efficient handling of complex large-scale projects.

As can be seen, MDE is transversal to the other sections and topics addressed in this article, as it can contribute significantly to all areas. However, this section focuses more specifically on abstract modeling and automatic transformation of quantum and hybrid software models.

One of the most relevant challenges of current QSE is that existing languages and methodologies in quantum software development operate at a notably lower level of abstraction compared to the typical scope addressed by MDE techniques for classical software. The fundamental differences between classical and quantum software complicate the development and reuse of MDE techniques for this kind of application (e.g., Boolean algebra vs. quantum mechanics principles, well-known architectures and patterns vs. algorithm complexity, deterministic vs. stochastic, low dependency and high compatibility with hardware vs. the opposite, high scalability vs. NISQ limitations). Furthermore, the design of hybrid software is rarely modeled at a higher abstraction level, ignoring irrelevant low-level technical details and focusing on architectural details (see Section 4.5). To address these challenges, several research groups have started working on Model-Driven QSE and have already produced relevant contributions in the area.

Ali and Yue [7] offer a preliminary exploration of how MDE can be used to support quantum code generation or quantum verification and validation. Similarly, Gemeinhardt et al. [57] propose

an initial roadmap of research questions that should be addressed to bring the benefits of MDE to quantum software. From this starting point, the authors have developed several additional contributions. Gemeinhardt et al. [56], in a different work, analyze how model-driven optimization techniques can be applied in the context of quantum software. Furthermore, the same authors propose a modeling language and a design framework for quantum circuits that support the definition of composite operators [58]. This allows developers to raise the abstraction level of quantum algorithm design alongside the provided code generator. Similarly, Ammermann et al. [14] propose a view-based quantum development approach based on a single underlying model. That proposal is supported by a quantum IDE to model quantum software at a higher abstraction level by considering different specific views for various stakeholders.

With a different goal but using similar model-driven principles, software modernization efforts have been tailored to tackle the issues associated with migrations of these hybrid software systems [141]. The software modernization process, which integrates traditional reengineering with MDE principles, has been progressively developed by Pérez-Castillo et al. over the last few years. Thus, reverse engineering techniques to abstract different quantum programming languages have been proposed [130]; the restructuring and transformation of different high-level representations have been addressed [77]; or preliminary code generation techniques from high-level designs of hybrid software have been proposed [138].

One of the most recurrent challenges covered is how to model quantum/hybrid software in an abstract way. In this sense, Pérez-Delgado and Perez-Gonzalez [131] outlined certain principles for designing modeling languages for quantum software. Similarly, Pérez-Castillo et al. [139], propose a UML profile that covers the analysis and design of hybrid software. In addition, Ali and Yue [7] discuss some ideas for obtaining new metamodels for modeling quantum programs as extensions of UML. Apart from UML, other authors have focused on other existing standards. Weder et al. [192] introduce a BPMN-based modeling approach to facilitate the integration of quantum computations with classical applications and quantum circuits, aiming to simplify orchestration tasks and ensure portability. Zhao [210] proposes a foundation for developing **Architecture Description Languages (ADLs)** specifically designed for hybrid quantum-classical software systems. The research aims to establish a formal framework for describing the architecture of such systems by capturing both quantum and classical components and their interactions at the architectural level. In addition, some of the works mentioned [77, 130] use the extension of the Knowledge Discovery Metamodel to support the maintainability of quantum software.

In contrast to extensions for existing modeling standards, some authors have explored using DSMLs, as discussed by Gemeinhardt et al. [57]. DSMLs, such as SimuQ [129] and Quingo [50], cater to specific needs within quantum computing. Polat et al. [134] provide MDE4QP, a framework built upon the existing MDE tools in classical platforms with which to define optimization problems in a Platform-Independent Model, and then provides automatic transformations for different Platform-Specific Models such as gate-based or annealing programming models.

Another relevant DSML is QIR, which has been developed by Microsoft [55], QIR serves as a DSML built on the LLVM intermediate language, to provide a unified interface between quantum programming languages and platforms. There exist other similar libraries and frameworks that make the code hardware-independent. For example, Xanadu Pennylane software and, to some extent, AWS Braket software can convert their code to multiple hardware vendors. Finally, other examples of DSMLs have been proposed for Quantum Machine Learning [111] and for modeling quantum circuits derived from satisfiability problems [11].

Those are only some examples of the work in the intersection between MDE and QSE. However, additional research efforts are still needed to address the remaining challenges in this domain. Specifically:

> **Challenges in Quantum MDE**
>
> **Ch-MDE-1.** Modeling quantum-specific constructs.
> **Ch-MDE-2.** Development of high-level design methodologies.
> **Ch-MDE-3.** Scalable quantum software maintenance and evolution.
> **Ch-MDE-4.** Intelligent code generation and orchestration.

Modeling Quantum-Specific Constructs. An open challenge for the future in the application of MDE to quantum software lies in developing quantum-specific modeling constructs. Unlike classical software, quantum software requires precise representations of quantum gates, circuits, and states, which necessitate specialized MDE languages, techniques, and tools capable of accurately modeling these components. Additionally, the inherently probabilistic nature of quantum computations, including aspects such as measurement probabilities, must be effectively incorporated into MDE models. In this sense, there already exists some approximations for modeling uncertainty [172], which should be further explored for quantum software. Addressing these challenges is crucial for advancing the reliability and accuracy of quantum software development using MDE methodologies.

Developing high-level design methodologies for hybrid software systems is crucial for bridging the gap between classical and quantum computing paradigms. This involves creating abstract modeling frameworks that encapsulate the complexity of hybrid quantum-classical interactions, providing a unified view that enhances comprehensibility and facilitates design decisions [210]. Future research could focus on developing DSMLs that offer intuitive abstractions for quantum-classical integration, enabling software engineers to design hybrid applications without delving into the low-level technical aspects of quantum computing.

Scalable Quantum Software Maintenance and Evolution. As quantum software becomes more complex and widespread, maintaining and evolving these systems will pose significant challenges. Future research could explore MDE approaches to predict the impact of changes in quantum software components also involving sophisticated quantum software metric models, ensuring compatibility and optimizing performance across versions. Techniques such as model-based regression testing and automated refactoring tools tailored to quantum software could be developed to support scalable maintenance processes.

Intelligent code generation and orchestration are crucial to improving the productivity and efficiency of QSE. By automating the generation of quantum code from high-level models, developers can focus more on problem-solving rather than the intricacies of quantum programming languages. Future work in this domain could look at developing sophisticated code generation engines, e.g., based on model-driven optimization, that translates models into executable quantum code and optimizes this code for specific quantum hardware, considering factors such as qubit connectivity and gate fidelity. Orchestration, on the other hand, involves managing the execution of quantum and classical components in hybrid systems, ensuring they operate seamlessly together to achieve desired outcomes. Furthermore, the research could aim to create intelligent orchestration tools that dynamically manage the execution of hybrid applications, optimizing resource allocation and execution in order to improve performance and reliability.

4.4 Programming Paradigms

Programming consists of instructing computers with algorithms (strategies) to achieve an objective (produce a result) using programming languages.[4] The typical approach to formulating strategies

---

[4]By programming language here we mean any special or dedicated language (including graphical or natural languages) with enough precision to express algorithms that finally can instruct computers.

involves breaking them down into increasingly simpler steps until the level of instruction provided by programming languages is reached. The process of breaking down occurs at different levels of abstraction. Thus, the way strategies are approached is conditioned by the final set of operations available for the computation model, be it classical or quantum.

Since 1996 to date, when the metalanguage lambda-q calculus was proposed [98], many quantum programming languages have been created. Most of them are languages that follow the imperative programming paradigm. Some examples of them are QASM [123] based on Assembly, Ket [32] based on Python or Q# [170] based on C#. A few of them are functional, like QML [61] and Quipper [62] based on Haskell, or even declarative languages, like Forest [166] based on Python.

All of the above languages are designed to compose quantum circuits that will compute on a QPU. Programming circuit-based quantum computers presents a significant challenge for classical programmers approaching it [8]. While many factors contribute to it, we focus here on two of them.

On the one hand, the conception of strategies for classical and quantum programs differs significantly. A classical program encodes a strategy for composing a result.[5] Each step in a classical program transforms the machine's state. Eventually, the result is produced, and the program is deemed correct because it successfully constructs the result. On the contrary, a quantum program encodes a strategy for discovering the result. In a quantum program, calculations usually begin with initialization operations on a quantum register, thus generating a quantum state in which different values co-exist in superposition. Each of these values has an associated probability amplitude collected in the corresponding state vector associated with the state. The program ends by collapsing the quantum state. The resulting value after collapse is one of the values of the quantum state in superposition. Therefore, this result already exists in the quantum state. The program strategy is then determined by all manipulations of the quantum state after its initialization and until its collapse. Such manipulations are aimed at increasing the probability amplitude of the value(s) that constitute a solution to the problem solved by the program.

On the other hand, quantum programming languages provide a low level of abstraction. Although there are many languages for quantum programming, such as QASM, Quil, qibo, and Qiskit, all of them constrain the abstraction level to that provided by the primitives of the language (quantum gates). Those primitives operate directly over the phase and amplitude of qubits representing the quantum state. So, quantum program strategies must be considered in terms of phase and amplitude manipulation. Compared to classical programming, it is like composing strategies in terms of rotation, shifting, addition, or carrying operations over binary registers. This stage has long been superseded in classical software engineering, and history has shown its benefits.

In recent years, some initiatives have been developed that seek to make quantum programming more affordable by raising the level of abstraction and providing abstractions that facilitate the design of discovery strategies. One of the significant advances in achieving comprehensibility, reliability, and simplicity of classical software was the introduction of basic data types like integer, float, or character, along with simple operations on them. Now, inspired by that [152, 153] and [151] propose a kind of Oracle-Based Quantum Programming. The idea is to treat quantum registers as data type encodings. Such types are complemented with oracles [60, 87] that implement basic operations on them. The feasibility of this approach has been explored considering quantum registers encoding integers. Then, a set of reusable and composable oracles implementing simple operations [151] have been developed.

---

[5]Strategies to produce results are not only for imperative languages. Logic or declarative languages also express strategies to produce results with instructions conceived for their different paradigms

To provide quantum programmers with a higher abstraction level, other authors also exploit the idea of providing Quantum Types [176]. Programmers can define their own types and thus create new abstractions on which to build their algorithms. Some other works are also focused on providing different means to encode specific types and operations in quantum states [31, 73, 159, 195].

The concern of raising the level of abstraction to make the programming of quantum systems more accessible is not unique to circuit-based systems. Also, researchers in the field of annealing quantum computing aim to achieve this objective. Programming a QUBO is not an easy task. Thus, some researchers [142] try to approach that problem by enabling programmers to formulate their algorithms in the scope of constraint programming [99] and then translating constraints to a QUBO.

The above work reveals some of the challenges that will have to be addressed over the next years:

Challenges in Quantum Programming Paradigms

Ch-PP-1. Complexity of circuits.
Ch-PP-2. Composable and reusable quantum software.
Ch-PP-3. Abstractions for quantum software.

Complexity of Circuits. This is one of the core challenges in quantum computing, particularly when it comes to implementing algorithms that require sophisticated operations. Oracles, which serve as black-box subroutines, have shown their potential as a pattern for encapsulating complex operations. These oracles are essential in many quantum algorithms, such as Grover's search algorithm, where the amplitude amplification technique is encoded as a subroutine that can be reused in larger computations [64]. The ability to encapsulate such operations is invaluable because it abstracts the complexity, allowing developers to focus on higher-level algorithmic design. However, as with any complex quantum operation, implementing these oracles can result in circuits that are both wide (involving many qubits) and deep (requiring numerous sequential gate operations), which are challenging for current quantum hardware.

The wide and deep nature of these circuits presents serious constraints, particularly with NISQ devices, which are limited by coherence times, gate fidelities, and qubit connectivity. A wide circuit implies a large number of qubits, which is difficult to achieve with current hardware, while deep circuits require a high number of gates, which increases the likelihood of errors due to noise and decoherence. Moreover, deep circuits can be inefficient for near-term quantum computers, as long-running computations exceed the coherence time of qubits, leading to a loss of quantum information. Consequently, optimizing quantum circuits to minimize both the number of qubits and the number of sequential gate operations is crucial for making complex algorithms feasible on today's hardware.

Researchers are actively exploring optimization techniques to address these challenges. One promising direction involves quantum gate synthesis, where gate sequences are optimized to reduce circuit depth [75]. Similarly, techniques such as circuit compression aim to minimize the number of gates required for a given quantum operation, while qubit recycling strategies reuse qubits within a circuit to reduce the overall qubit count [155]. Another approach is hardware-aware optimization, where the specific architecture of a quantum device, such as qubit connectivity or gate fidelity, is considered during the design of the circuit [120]. This allows circuits to be mapped onto hardware in a way that reduces the need for costly operations like SWAP gates, which arise from poor qubit connectivity. These advancements in circuit optimization are critical for the practical implementation of oracles and other complex quantum subroutines on real-world devices.

Composable and Reusable Quantum Software. This challenge is essential for reducing the complexity of designing quantum algorithms and circuits. Developing oracles or other complex quantum circuits often requires a deep understanding of quantum mechanics and quantum states, making the process daunting for many developers. To make quantum development more accessible and efficient, it's crucial that these circuits are designed to be as reusable as possible. By creating quantum circuits that can be reused across multiple algorithms and applications, the overall effort required to develop new quantum solutions is significantly reduced. Reusability in quantum software not only saves time and resources but also enables developers to build upon existing, well-optimized circuits rather than starting from scratch each time a new algorithm is needed.

For circuits to be reusable, they must also be composable, meaning they can be integrated with other circuits seamlessly. This is where the current landscape of quantum development faces a challenge. The tools and frameworks available to support circuit composition and reuse are still in their infancy, lacking the sophistication seen in classical software engineering. In classical computing, developers have access to mature libraries, modular components, and standardized interfaces that enable the efficient reuse and integration of code. In contrast, quantum computing is still developing such frameworks, and the process of reusing quantum circuits is far more complex due to the unique nature of quantum states, entanglement, and superposition. Furthermore, quantum circuits often need to maintain delicate quantum properties like coherence and phase relationships, which adds additional constraints when trying to compose them with other circuits.

Initial efforts to create tools and techniques for the documentation, reuse, and composition of quantum software have highlighted just how different the quantum domain is from classical software engineering [154]. Unlike classical software, where code can be modularized and reused with relative ease, quantum circuits require more specialized handling to preserve their quantum characteristics when integrated with other circuits. This makes the task of developing reusable quantum components much more challenging. Over the next years, the development of techniques for better documenting quantum circuits, creating modular quantum software architectures, and enabling the flexible composition of quantum components will be crucial. Such advancements will help quantum developers reuse complex quantum logic efficiently, thereby accelerating the development of new quantum applications and making quantum software development more scalable and accessible for a broader range of developers.

Abstractions for Quantum Software. This challenge is essential for elevating the usability and flexibility of quantum programming languages. At present, the process of encoding data types in quantum states and designing corresponding operations on these states is one of the foundational strategies for increasing the abstraction level in quantum computing. This strategy opens up the potential for defining new sets of operations that enrich the primitive operations currently provided by quantum programming languages. These operations, in turn, can offer developers more versatile and powerful tools for designing quantum algorithms. However, the types and operations that have been proposed so far in the literature often mirror those used in classical computing, limiting the scope of what quantum computers can achieve. For instance, some researchers have attempted to define quantum states that encode prime numbers [54], but these kinds of abstractions, while useful, might not fully exploit the unique capabilities of quantum systems.

Quantum computing's true potential lies in its ability to model and simulate complex quantum systems, such as those found in chemistry, physics, and other fields where quantum effects dominate. Therefore, a more promising direction for developing quantum abstractions would be to focus on encoding quantum-native data types that reflect the kinds of problems quantum computers are inherently suited to solve. For example, instead of abstracting classical concepts like numbers or strings, quantum states could be used to represent more complex entities like molecules, atomic structures, or even quantum fields. Operations on these quantum states could simulate interactions

between molecules or particles, allowing quantum algorithms to directly engage with the types of calculations that classical computers struggle with, such as simulating chemical reactions or quantum systems with many-body interactions. This approach would not only enhance the level of abstraction in quantum programming but also align more closely with the strengths of quantum computing as envisioned by pioneers like Richard Feynman [43], who famously proposed that quantum computers would excel in simulating physical processes.

4.5 Software Architectures

Software architecture involves creating structures essential for understanding and creating a software system. These structures consist of software elements, their relationships, and their properties. We can use quantum computers to make our classical software tackle problems that were previously out of reach. Consequently, quantum systems should not operate independently but should co-exist and collaborate with classical systems [24, 141, 191, 210].

Tools and methodologies are needed to integrate quantum layers and stakeholders with those in classical information systems. The motivation for such integration of different systems should not just be the fact that they reside in different computational paradigms. Instead, information systems should be able to be designed as a whole and integrated based on the functionalities they provide independently of the hardware architectures in which they reside.

Software architects play a crucial role in achieving seamless integration while designing systems that effectively meet businesses' requirements. Thus, the systematic review by Khan et al. [82] investigates the software architecture for quantum computing systems. Further challenges and opportunities are discussed in detail by Yue et al. [202]. In a similar study, aspects involved in various software architectures are analyzed since "the software architecture of quantum computing systems plays a pivotal role in determining their ultimate success and usability" [215]. In the work "Architecture Decisions in Quantum Software Systems" [5], the authors conduct empirical research to examine and analyze architectural decisions while creating quantum software systems. A foundation for developing ADLs has also been discussed by Zhao [210].

Some particular design patterns have also been proposed by Frank Leymann [87] and Buhler [23], which mainly focused on the design of quantum circuits. In the study [137], authors perform a preliminary exploration of the usage in the practice of some of those design patterns. Preliminary work in extending the definition of patterns for hybrid software systems is proposed in [193]. Guo et al. [65] specifically focused on quantum circuit ansatzes, a type of specialized design patterns commonly used for VQA [25]. Rather than introducing new ansatz patterns, their work aimed to create a comprehensive catalog of existing quantum circuit ansatzes, facilitating abstraction and reuse in the practical design and implementation of quantum algorithms.

In summary, the research prospects on quantum software architecture might be the following:

Challenges in Quantum Software Architectures

Ch-SA-1. Architectural decisions in quantum software.
Ch-SA-2. Design patterns for hybrid software systems.
Ch-SA-3. Empirical evidence for the application of design patterns.
Ch-SA-4. Evolution of hybrid software architectures.

Investigating all the factors influencing architectural decisions in quantum software and system design requires a comprehensive understanding of both the implementation details and the broader technical choices that shape the development of a robust quantum computing system. Exploring these factors should consider a wide range of dimensions, each playing a critical role in guiding

the decision-making process. By systematically evaluating these considerations, architects can build systems that not only meet immediate functional requirements but also ensure long-term sustainability and adaptability in an evolving quantum computing landscape.

One of the key factors is performance optimization, which involves balancing the limitations and capabilities of quantum hardware, such as gate fidelities, qubit coherence times, and error rates, with the computational goals of the software. Architectural decisions must ensure that quantum algorithms are executed as efficiently as possible, considering the constraints of today's NISQ devices. This includes minimizing the depth of quantum circuits, optimizing qubit layout to reduce the need for qubit-swapping operations, and employing error correction techniques that don't overly degrade performance. Performance decisions must also account for how well the quantum system integrates with classical components in hybrid architectures, as seamless interaction between quantum and classical processes is crucial for many near-term applications.

Another essential factor is compatibility and interoperability (see also *Ch-SoC-1*). As quantum software and systems are developed, architects must ensure that they can function across different quantum hardware platforms and work in tandem with classical systems. This includes making decisions about standardizing communication protocols, programming languages, and intermediate representations like QIR, ensuring that the quantum architecture remains flexible and capable of adapting to future advancements in quantum hardware. Interoperability also involves considering how the system can support multi-vendor environments, where different components may come from various providers, and how they can collaborate in a seamless manner. This factor is essential for creating scalable, vendor-agnostic solutions that avoid lock-in and provide more flexibility for developers and users.

Cost implications are another critical dimension to consider. Quantum computing resources are still expensive, and usage costs can vary based on the number of qubits, the depth of circuits, and the length of time the quantum hardware is in use. Architects must balance these costs with the need for performance, scalability, and experimentation, ensuring that the architecture is not only cost-effective in the short term but also scalable as more affordable quantum technologies become available. Additionally, scalability itself is a significant concern, as quantum systems need to accommodate the expected growth in qubit numbers and system complexity over time. This means planning for architectures that can support larger, more powerful quantum computers in the future, as well as hybrid solutions that evolve with advancements in both quantum and classical computing.

Defining the design patterns for building hybrid software systems that integrate quantum and classical computing is essential for ensuring both scalability and flexibility. At the low-level software component level, the focus should be establishing patterns promoting modularity, reusability, and efficiency. This includes designing compilation units that translate high-level quantum algorithms into executable instructions optimized for various quantum processors. Patterns such as the factory or builder patterns can be employed to dynamically create and configure these components, ensuring that quantum circuits and classical routines can be easily reused across different algorithms and applications. Additionally, functions and modules should be encapsulated in a way that allows for the seamless integration of classical and quantum operations while maintaining independence from the underlying hardware (see also *Ch-SoC-2*). This ensures that developers can swap or upgrade components without extensive refactoring, promoting code reuse and adaptability in evolving quantum-classical environments.

At a high level, architectural design patterns need to be developed to manage the orchestration of services and the execution of workflows that span both quantum and classical systems. This requires patterns that can coordinate the interaction between classical software services and quantum computing tasks [210]. Patterns such as SOA or microservices architecture can be adapted to

the quantum domain to manage the distributed nature of hybrid workflows. In these architectures, quantum services act as specialized computational units that can be called upon when needed, and orchestration mechanisms ensure that dataflow smoothly between quantum and classical resources. For example, mediator and adapter patterns can help manage the communication between QPUs and classical backends, allowing the orchestration layer to interface with multiple quantum platforms without needing to change the core business logic. This pattern-driven approach enables quantum and classical services to operate in tandem, ensuring that workloads are balanced, computational tasks are optimized, and the hybrid system is responsive to varying performance and resource needs.

Moreover, workflow management in hybrid systems requires additional patterns that can handle task scheduling, resource allocation, and error management in complex, distributed environments. Patterns like chain of responsibility or observer can be implemented to track the execution of quantum tasks and monitor their status in real-time, allowing the system to react to errors or changes in quantum state coherence. These patterns provide the flexibility needed to manage the dynamic nature of hybrid computations, where quantum resources are expensive and limited, and classical resources are used to support the optimization of quantum workloads. By clearly defining these high-level patterns, developers can ensure that hybrid systems are not only efficient but also modular, scalable, and maintainable, capable of evolving alongside quantum technology advancements. This systematic use of both low-level and high-level design patterns will enable the seamless integration of quantum and classical systems, laying the foundation for scalable, high-performing hybrid computing solutions (cf. Section 4.2).

Research focused on acquiring substantial empirical evidence for the application of design patterns in industrial quantum and hybrid computing scenarios is vital for bridging the gap between theoretical advancements and practical implementation. Such research should aim to provide a deeper understanding of how both current and future design patterns and architectural decisions impact the development, deployment, and scalability of quantum systems in real-world settings. The quantum domain, with its rapid evolution and nascent adoption in industry, presents unique challenges that classical software patterns may not fully address. Therefore, empirical studies are essential to validate and adapt these patterns to quantum computing's distinct requirements, such as error rates, qubit coherence, and hybrid quantum-classical interaction.

In the immediate term, empirical research should focus on how existing design patterns, adapted from classical computing, function within industrial quantum applications. This involves studying patterns like modularity, reusability, and service orchestration to determine their effectiveness in managing the unique complexities of quantum systems. For example, the research could investigate how well low-level patterns (such as the factory or adapter patterns) handle the translation of quantum algorithms across different hardware platforms and whether these patterns enable effective hardware abstraction. Similarly, high-level patterns related to service orchestration and workflow management should be evaluated for their ability to integrate quantum tasks into existing classical infrastructures seamlessly. Real-world case studies involving industries like finance, pharmaceuticals, and logistics, where quantum computing has begun to make an impact, could serve as valuable sources of data for analyzing how these patterns contribute to system efficiency, scalability, and error management.

Looking ahead, research should also explore new design patterns and architectural models specifically tailored to quantum computing's future needs. As quantum hardware matures and larger-scale quantum computers become available, new patterns will be required to manage more complex computations and workflows. For example, quantum-native design patterns that account for quantum entanglement, superposition, and error correction should be developed to enhance

the effectiveness of hybrid systems. These new patterns might focus on optimizing quantum-classical interactions, reducing quantum circuit depth, or managing qubit connectivity in a way that classical computing patterns cannot address. Research should seek empirical evidence on how these emerging patterns perform in future industrial scenarios, testing them across different quantum hardware platforms, industries, and use cases. Furthermore, this research could identify best practices for adopting these new patterns and inform standardization efforts in the QSE field, ensuring that quantum solutions are not only powerful but also maintainable and scalable for industrial applications.

In addition to examining the direct application of design patterns, research should also consider the broader architectural decisions that guide the development of quantum systems in industry. This includes exploring the impact of architectural choices on performance, cost, and flexibility, as well as the tradeoffs between different architectural approaches in hybrid quantum-classical systems. For instance, empirical studies could investigate whether certain architectures are better suited for specific quantum algorithms or whether certain patterns enable more efficient resource allocation and error management in NISQ environments. Understanding how design patterns influence these architectural decisions in practice will help industries make more informed choices about how to integrate quantum computing into their operations. By acquiring empirical evidence on these issues, researchers can provide industries with actionable insights that reduce development risks, streamline quantum adoption, and maximize the potential of quantum computing technologies in real-world scenarios.

Investigating the evolution of hybrid software architectures over time is crucial for understanding the challenges and complexities that arise as quantum and classical systems become more integrated. As these architectures evolve, new patterns, tools, and technologies emerge, which can introduce both improvements and complications. Hybrid quantum-classical systems, in particular, are subject to rapid advancements in quantum hardware and software, which can lead to significant architectural changes. These changes, if not carefully managed, can accumulate technical debt, a situation where shortcuts in design, implementation, or maintenance lead to long-term inefficiencies or scalability issues. Therefore, it is essential to not only track the evolution of these hybrid systems but also develop maintenance strategies that prevent architectural decay and ensure clean, effective, and future-proof designs.

One of the key areas to investigate is how the integration of new quantum technologies impacts the long-term health of hybrid architectures. As quantum computing hardware advances, the need to adapt existing architectures to accommodate new qubit types, error correction techniques, or quantum algorithms becomes inevitable. These adaptations may introduce complexity, leading to bloated or inefficient architectures if not managed properly. Researchers should focus on identifying the points at which architectural refactoring is necessary and developing techniques to modernize or restructure hybrid systems without disrupting their core functionality. For example, automated tools could be designed to refactor hybrid architectures by recognizing and replacing outdated components, optimizing circuit depth, or improving the interoperability between quantum and classical services. By tracking how hybrid architectures evolve in response to technological shifts, developers can anticipate points of potential technical debt and address them proactively.

In addition to refactoring, hybrid architectures must be continuously maintained to prevent technical debt from accumulating over time. A key strategy for managing technical debt in these systems is the development of automated maintenance solutions. These solutions should be capable of analyzing hybrid architectures for inefficiencies, code duplication, and unused or outdated components, providing developers with actionable insights on how to keep the system clean and effective. For example, maintenance tools could automatically identify quantum circuits that are no longer optimized for the latest hardware or pinpoint classical components that are redundant

due to new quantum capabilities. Furthermore, automated testing frameworks that continuously verify the performance, accuracy, and stability of both quantum and classical components could be crucial for preventing the slow buildup of inefficiencies. These tools would help developers mitigate technical debt by providing regular, data-driven feedback on system health, ultimately leading to more sustainable and efficient architectures.

Finally, the evolution of hybrid software architectures also demands attention to documentation and version control as a means of ensuring long-term maintainability. As quantum and classical systems evolve together, maintaining accurate and up-to-date documentation becomes increasingly important for managing architectural complexity. Documentation should cover both the high-level architecture of the system and the details of how quantum and classical components interact, ensuring that future developers can easily understand and extend the system without introducing additional complexity. Version control systems must also be adapted to manage the distinct versions of quantum software, classical software, and their integration points, ensuring that any changes to the system are well-tracked and reversible if needed. Establishing best practices for maintaining clear and consistent documentation, along with robust versioning strategies, will be essential to keeping hybrid architectures maintainable as they evolve over time, thereby reducing technical debt and enhancing long-term system effectiveness.

4.6 Software Development Processes

Akbar et al. [3] categorize various QSE challenges. Specifically, two challenges suggest that specific quantum/hybrid software development processes are needed: the 14th challenge: "Integration with classical computing," and the 22nd challenge: "Project management issues." With regard to the integration of classical quantum software, quantum software cannot be developed and operated in isolation. "A key challenge is to fully integrate these types of the system into a unified classical and quantum software development lifecycle" [24]. Thus, there are some issues, for example, the interpretation of the results by the classical counterpart, since quantum computations are stochastic [67]. In addition, new architectural paradigms and design patterns will be necessary to develop quantum software effectively [82]. Regarding project management, some issues, such as risk management, are specifically framed in the development of hybrid software systems. For example, the limited availability of real quantum hardware, which leads to testing quantum software in simulators, sometimes leads to different results when the quantum software is executed in the production environment. Other risks with a high impact on the success of quantum software development projects are the lack of standardized tools and frameworks, or scalability [4].

There is some preliminary research on the holistic quantum software development lifecycle. Thus, Zhao [207] proposed a slight adaptation of the waterfall model to support the design and construction of quantum software systems, covering phases such as requirements analysis, design, implementation, testing, and maintenance. In a similar approach based on the waterfall model, Dey et al. [38] proposed some alternative stages.

Since these models can inherit all the weaknesses of the classical waterfall lifecycle, other proposals are based on iterative models. Weder et al. [190] proposed an iterative model based on "quantum data provenance," which serves in different phases of the lifecycle. In addition, Pérez-Castillo et al. [140] propose an adaptation of the Incremental Commitment Spiral Model, which is also an iterative model that suggests risk management during software development.

Other proposals examine how agile good practices can be integrated into quantum software development. In this context, proposals have emerged that address the integration of the DevOps paradigm within the domain of quantum software development [59, 146, 169]. In a more analytical way, an interview-based study [83] shows the most important challenges in agile-quantum software development, such as sustainable scaling and the need for mature tool ecosystems and

standard agile specifications. The significance of these challenges is likely to be paramount in the coming years.

The open research topics for the next years in this domain might be the following:

> **Challenges in Quantum Software Development Processes**
>
> **Ch-DP-1.** Iterative development of hybrid software.
> **Ch-DP-2.** Risk management.
> **Ch-DP-3.** Project management.

Managing iterative development of hybrid software systems through the whole lifecycle. Iterative development models, which have proven effective in managing complexity and enhancing flexibility in classical software projects, must be adapted to hybrid software systems development. This includes models that can accommodate rapid changes in quantum technology and provide frameworks for the co-development of classical and quantum components (see also *Ch-SA-4*). This, in turn, will introduce unique challenges, such as the need for quantum-specific design patterns, integration testing frameworks, and development tools that can handle quantum-classical software systems.

A survey conducted by Jiménez-Navajas et al. [78] shows that there is a certain dissatisfaction with specific tools, indicating a gap between the available tools and the developers' needs. In particular, that survey indicates that current modeling tools are insufficient for quantum and hybrid quantum-classical software. While classical software has more robust modeling support, there is a significant need for better tools to model quantum and hybrid software systems effectively. This is aligned with MDE challenges presented in Section 4.3.

Risk management within the specialized realm of quantum software, coupled with sustainable scalability strategies, is essential for developing progressively larger and more intricate hybrid software systems. Thus, adaptive risk management strategies, which can evolve with the quantum computing landscape, will be necessary. This might involve dynamic resource allocation models, contingency planning for quantum hardware failures, and methodologies for evaluating the reliability of quantum algorithms.

In addition to the risks involved in quantum computing *per se* [47], there are some specific risks for quantum software development [68]. Table 1 summarizes most of the risks mentioned above, pointing out some approaches that can be used to mitigate them. As we can see, good practices in software and systems engineering (MDE, SOA, ADM, metrics, etc.) can be used to mitigate some of these risks.

One of the most crucial risk mitigation strategies is adopting an agnostic architecture, which helps safeguard the investments that users have made in developing quantum software assets.

We are experiencing a continuous and often rapid evolution in the disruptive field of quantum computing. Given the current state of development, making precise predictions regarding the widespread adoption and deployment of general-purpose quantum computing remains difficult [36]. As a result, it is even more critical in quantum computing than in other areas of IT to conduct thorough risk assessments. The responsibility will fall to the "Quantum Business Strategist" [86] to develop a risk management strategy grounded in an integrated approach, allowing organizations to understand quantum risks comprehensively.

Project management, particularly in its focus on the operational aspects of quantum software, requires the integration of DevOps or similar agile paradigms into the software development lifecycle. Hence, future research lines could focus on developing or adapting agile toolchains, including version control systems, **Continuous Integration/Continuous Deployment (CI/CD)** pipelines, and quantum-aware agile project management tools. This research line could investigate

Table 1. Main Quantum Software Risks and Mitigation Approaches Adapted from [68]

Risk	Mitigation Approach
Lack of preparation	Training, automation (AQSE)
Difficulty in use	User-friendly, GUIs
Algorithm complexity	Training, algorithm libraries
Variety of types of quantum computers	Technology agnosticism
Changes in the same quantum system	Full portability of quantum software
Diversity of programming languages	MDE, low code techniques
Integration of classical and quantum IT	SOA, standardized APIs
Hybrid information systems construction	New hybrid software lifecycles
Migration of classical software	Quantum software modernization (ADM, Architecture-Driven Modernization)
Poor quality of quantum software	New testing techniques/new software quality characteristics and metrics
Problems in execution and results/ environment management	An integrated environment for design and execution
Lack of community	Create quantum software networks and interest groups

how quantum computing as a service can be integrated into software development projects, enabling agile teams to leverage quantum computing resources efficiently.

In addition, another critical point during project management is the impact of automated code generation tools tailored to QSE. The use of LLMs to assist in software development has shown promising results in classical software, and it is expected that similar techniques could be applied to quantum software, as demonstrated in some preliminary works [39, 66]. LLMs could assist developers in writing, refactoring, and optimizing quantum code, potentially lowering the barrier to entry for quantum software development and accelerating the development process (see Section 4.7). However, challenges in using LLMs for software development include bias and inaccuracy in generated code, security risks from suggesting vulnerable patterns, and limitations in contextual understanding, which can lead to integration issues within complex projects. Additionally, there are ethical and legal concerns around intellectual property, especially when LLMs may reproduce licensed code. Developers also face a learning curve in effectively using these tools, and integration into existing development workflows, such as CI/CD pipelines, remains a challenge. Finally, trust in AI-generated code requires careful human oversight to ensure reliability.

4.7 AI

As in almost every software engineering domain, QSE has been increasingly influenced by the recent advances in AI. The intersection between AI and QSE is a promising and rapidly growing area of research. On the one hand, applying AI techniques to enhance the processes and methodologies within QSE is starting to yield significant benefits. AI can assist in optimizing quantum software development and automating tasks like quantum circuit optimization, error correction, and hybrid system orchestration. On the other hand, quantum computing itself holds immense potential to revolutionize AI, particularly by developing quantum algorithms designed to enhance machine learning and deep learning processes, leading to the exploration of quantum AI systems.

In terms of using AI for QSE, several research teams are actively exploring how machine learning models can help design, test, and optimize quantum algorithms. For instance, AI can automatically suggest optimized quantum circuits or identify more efficient qubit allocations, thereby improving the overall performance of quantum computations. AI-driven tools can also aid in error mitigation by predicting and compensating for quantum noise, which is especially important in NISQ devices. Additionally, AI-enhanced simulators could provide more accurate predictions of quantum system behaviors, helping developers better understand the limitations of their algorithms and hardware and adjust accordingly. AI can also be integrated into quantum programming environments to assist with debugging, verification, and testing of quantum circuits, areas that are notoriously complex in quantum software.

Works such as the one proposed by Salm et al. [148] propose using machine learning approaches to optimize the selection of compiled quantum circuits for the available quantum computers. This work is further developed in [149], where several machine learning algorithms are used to improve such selection. A similar approach was also followed by Garcia-Alonso et al. [53], where a predictive model is used to determine the minimal waiting time for executing a quantum circuit.

Several works have also been developed in the domain of VQAs that try to leverage the benefits of AI. For example, Furutanpey et al. [51] propose using QNNs to help system designers distribute hybrid quantum applications through the computing continuum. Similarly, in [173], the authors address the problem of hyper-parameter selection in warm-starting quantum optimization problems.

In addition, neural networks have also been used for the automated synthesis of quantum circuits [117]. This work proposes AutoQC, a tool for synthesizing quantum circuits using the neural network from input and output pairs. In the same line, Dupuis et al. [39] focus on training code LLMs to generate quantum computing code, addressing challenges such as limited datasets and evolving technology, and demonstrating superior performance in tasks using the Qiskit library. From a different point of view, Quantum Machine Learning approaches have been empirically evaluated from a software engineering perspective in order to study the bugs present in such approaches. Also, Zhao et al. [213] try to ensure the correctness and robustness of Quantum Machine Learning approaches by inspecting almost 400 real-world bugs collected from more than 20 repositories of popular QML frameworks. And, of course, the nowadays ubiquitous ChatGPT has also been analyzed in terms of its ability to repair quantum programs [66].

Conversely, QSE for AI is another burgeoning field, with research teams investigating how quantum computing can be leveraged to improve AI algorithms. Quantum computing's ability to process vast amounts of data in parallel through superposition and entanglement has the potential to enhance machine learning algorithms significantly. Quantum versions of AI algorithms, such as QNNs and quantum support vector machines, are expected to be more efficient than their classical counterparts in processing large datasets and solving complex optimization problems (although loading classical data presents a challenge [104]). The unique characteristics of quantum computing, such as quantum entanglement and superposition, may enable more powerful AI models that can learn faster and handle more complex patterns than classical AI systems. As a result, QSE is playing an essential role in developing the tools, methodologies, and frameworks necessary to design and implement quantum-enhanced AI systems, making it a critical enabler of the next generation of AI technology.

Overall, the synergy between AI and QSE opens up exciting possibilities for innovation in both fields. By combining AI's capacity to learn, optimize, and automate with quantum computing's unparalleled computational power, researchers are beginning to push the boundaries of what is achievable in both quantum software development and AI. This intersection promises to drive progress in solving some of the most complex problems in computing, from optimizing quantum hardware to creating more intelligent and efficient AI models. As research continues to expand

in this direction, the integration of AI and QSE will likely become a cornerstone of technological advancements in both quantum computing and AI, unlocking new applications and accelerating progress in these cutting-edge domains.

These are just some examples of the initial impact of the relationship between QSE and AI on the research community. Nevertheless, additional challenges need to be addressed in this domain. Specifically:

> **Challenges in Quantum AI**
>
> **Ch-AI-1.** Quantum circuit optimization.
> **Ch-AI-2.** Developing hybrid AI-quantum workflows.
> **Ch-AI-3.** Error mitigation and correction.
> **Ch-AI-4.** Scalability of AI-assisted quantum software development.

Quantum Circuit Optimization with AI. Quantum circuits are often highly complex and require optimization to run efficiently on current quantum hardware, with limitations such as qubit coherence time and gate fidelities. Although AI techniques, such as reinforcement learning and evolutionary algorithms, have shown potential in optimizing classical software, applying these techniques to quantum circuits presents significant challenges. AI-driven optimizations must account for quantum-specific issues like gate depth, noise resilience, and the limited number of qubits. The research community must focus on developing AI models that can handle quantum-specific characteristics while ensuring the optimized circuits are scalable and executable on real quantum devices. Moreover, techniques that reduce the time it takes for AI systems to learn and improve quantum circuit designs need to be explored, making the process both time-efficient and cost-efficient.

Developing Hybrid AI-Quantum Workflows. Quantum computing's current state requires hybrid workflows involving both classical and quantum components. The challenge lies in integrating AI techniques within these hybrid quantum-classical systems. Designing efficient workflows where AI can dynamically decide which parts of a computation should be offloaded to quantum systems and which should remain classical is not trivial. AI algorithms capable of predicting the best partitioning of tasks between quantum and classical systems based on available resources and computational goals are needed. Additionally, these workflows must optimize communication between quantum and classical systems, balancing latency and data transfer speeds. The research community must investigate how AI can be effectively integrated into quantum software frameworks to create adaptive, efficient, and seamless hybrid systems.

AI for Quantum Error Mitigation and Correction. Quantum systems are inherently noisy, and error correction is one of the most significant challenges facing quantum computing. While AI has successfully automated certain aspects of error detection and correction in classical computing, the unique nature of quantum errors, such as decoherence and gate errors, requires new approaches. AI models could potentially be trained to predict quantum system errors and suggest real-time corrections. However, building AI systems that can learn and adapt to the specific noise models of quantum processors and account for the probabilistic nature of quantum measurements presents a significant research challenge. The scientific community must explore how AI techniques can be tailored to quantum error mitigation strategies while ensuring they work efficiently within the constraints of current quantum hardware.

Scalability of AI-Assisted Quantum Software Development. As quantum computing systems scale, so must the software and algorithms that control them. AI has the potential to accelerate quantum software development, but ensuring that AI-driven development processes scale effectively is a

major challenge. QSE currently lacks mature development environments, automated tools, and debugging support at the scale needed for widespread industry use. Developing AI systems that assist in code generation, automated testing, and debugging for quantum systems is essential, but these systems must scale as quantum computers grow in power and complexity. The research community should focus on building robust AI-assisted frameworks that can handle larger quantum systems, addressing issues such as increased circuit complexity, debugging for hybrid systems, and the integration of AI across multiple layers of quantum software.

4.8 Summary of the Different Key Challenges of QSE

A summary table of the different key challenges of QSE seen throughout this section is provided below:

Summary

- **Challenges in Quantum Software Testing (4.1)**
  - **Ch-ST-1.** Efficient test oracles.
  - **Ch-ST-2.** Test scalability.
  - **Ch-ST-3.** From simulators to real quantum computers.
  - **Ch-ST-4.** (Quantum) AI and (quantum) software testing.
- **Challenges in QSOC (4.2)**
  - **Ch-SoC-1.** Interoperability.
  - **Ch-SoC-2.** Platform independence.
  - **Ch-SoC-3.** Demand and capacity management.
  - **Ch-SoC-4.** Workforce training.
- **Challenges in Quantum MDE (4.3)**
  - **Ch-MDE-1.** Modeling quantum-specific constructs.
  - **Ch-MDE-2.** Development of high-level design methodologies.
  - **Ch-MDE-3.** Scalable quantum software maintenance and evolution.
  - **Ch-MDE-4.** Intelligent code generation and orchestration.
- **Challenges in Quantum Programming Paradigms (4.4)**
  - **Ch-PP-1.** Complexity of circuits.
  - **Ch-PP-2.** Composable and reusable quantum software.
  - **Ch-PP-3.** Abstractions for quantum software.
- **Challenges in Quantum Software Architectures (4.5)**
  - **Ch-SA-1.** Architectural decisions in quantum software.
  - **Ch-SA-2.** Design patterns for hybrid software systems.
  - **Ch-SA-3.** Empirical evidence for the application of design patterns.
  - **Ch-SA-4.** Evolution of hybrid software architectures.
- **Challenges in Quantum Software Development Processes (4.6)**
  - **Ch-DP-1.** Iterative development of hybrid software.
  - **Ch-DP-2.** Risk management.
  - **Ch-DP-3.** Project management.
- **Challenges in Quantum AI (4.7)**
  - **Ch-AI-1.** Quantum circuit optimization.
  - **Ch-AI-2.** Developing hybrid AI-quantum workflows.
  - **Ch-AI-3.** Error mitigation and correction.
  - **Ch-AI-4.** Scalability of AI-assisted quantum software development.

---

5 Software Engineering Key Areas and Challenges Regarding Quantum Computing

The landscape of software engineering has undergone a profound transformation in recent years, driven by the rapid evolution of new technologies, methodologies, and computing paradigms. As a result, there is a growing need for a new roadmap reflecting the future research direction in this dynamic field. A key example of this shift is the emergence of QSE, which presents unique challenges and demands fundamental changes in the core principles of software engineering [207]. Quantum computing, as we have already seen, introduces novel concepts such as superposition, entanglement, and probabilistic behavior, all requiring rethinking traditional software engineering paradigms to accommodate the distinct characteristics of quantum systems. The impact of these changes is expected to reshape not only QSE but the broader landscape of software development as a whole.

Already, researchers and thought leaders are actively considering what the short-term to medium-term future of software engineering will look like, both in general and specifically within the realm of QSE [118]. A notable event in this context is the *2030 Software Engineering Roadmap workshop* that was co-located with *ACM SIGSOFT FSE Foundations of Software Engineering* on July 15th and 16th, 2024 in *Porto de Galinhas, Brazil.*[6] This workshop brought together researchers worldwide to discuss the key challenges facing software engineering in the current decade. During the event, participants explored recent shifts in software engineering practices, shared their vision of the field's future evolution, and collaborated on designing a roadmap to guide the research community toward addressing these emerging challenges.

Drawing on the insights from the workshop [132], six key areas were identified as the most pressing challenges for software engineering research going forward. These areas reflect both the broader concerns within traditional software engineering and the specialized issues that arise in the context of quantum computing. In this section, we will outline these six key areas in detail, focusing on how they intersect with the field of QSE and how advancements in quantum computing may shape the future of software engineering as a whole.

AI for Software Engineering (AI4SE). The recent breakthroughs in machine learning, generative AI, and autonomous systems represent the most profound transformation in software engineering research and practice since the advent of the Internet in the latter half of the 20th century [171]. The software engineering community has never experienced such a rapid and dominant rise in new research directions, with topics like machine learning in software engineering and the challenges of engineering AI-driven systems becoming central themes in leading conferences and journals [76].

The relationship between AI and Quantum Computing offers immense potential to transform both fields, particularly in the context of software engineering (see also *Ch-AI*). As AI becomes more integrated into software development practices, it can significantly impact the evolution of quantum computing by providing advanced tools for algorithm design, circuit optimization, and error management. AI can help automate the design of quantum circuits, optimize qubit usage, and even predict and correct quantum errors in real-time, all of which are essential for improving the efficiency and reliability of quantum computations. This synergy is especially important in hybrid quantum-classical systems, where AI can manage the distribution of tasks between classical and quantum processors, ensuring the efficient orchestration of computational resources. Using AI to enhance the development process, quantum computing systems can become more scalable and accessible to a broader range of developers and industries.

On the other hand, quantum computing offers the potential to revolutionize AI itself, particularly in the domain of AI4SE. Quantum computing's ability to process large amounts of data simultaneously through quantum superposition and entanglement could dramatically speed up

---

[6]2030 Software Engineering Roadmap workshop. https://conf.researchr.org/home/2030-se

key machine learning algorithms, allowing AI models to be trained faster and more efficiently. This would enable AI systems to handle larger datasets and solve more complex optimization problems, making AI-driven tasks more powerful and accurate. Furthermore, quantum computing could allow AI models to better understand and manage complex software systems by analyzing multiple variables and configurations at once, leading to deeper insights into system behavior and improved software reliability.

Software Engineering by and for Humans (SE4H). Machine learning, generative AI, and autonomous systems are reshaping the landscape of software engineering, fundamentally altering the traditional concept of software artifacts [28]. These emerging technologies introduce complex ethical, fairness, and technical challenges that software engineers must now navigate. Humans are no longer just users of software systems, they have become integral components of expansive cyber-physical ecosystems. As a result, the scope of software engineering research must expand beyond the narrow view of human users and embrace a broader perspective that considers humans as essential elements within these interconnected systems [179].

SE4H strongly emphasizes creating software systems that are ethical, user-friendly, and aligned with human values. This approach can significantly influence the development of Quantum Computing by shaping how quantum systems are designed, implemented, and deployed. Quantum computing has the potential to solve complex problems that classical computing cannot handle efficiently, but the power and complexity of these systems also raise significant concerns related to transparency, accessibility, and usability. SE4H can drive the development of quantum systems that prioritize human-centric design, ensuring that quantum applications are not just powerful but also understandable, ethical, and responsive to user needs. For instance, SE4H principles can guide the creation of quantum software interfaces that simplify interactions between users and quantum machines, making quantum computing more approachable for non-experts while maintaining the necessary levels of control and oversight.

Conversely, quantum computing can profoundly influence software engineering for humans by introducing new capabilities for solving problems that directly impact human-centered applications. Quantum computing could revolutionize areas like healthcare, environmental modeling, and cryptography, enabling more efficient solutions to problems that have been computationally intractable using classical methods. These breakthroughs can enhance software systems designed for human use by providing faster, more accurate, and scalable solutions. However, as quantum computing introduces novel algorithms and computing paradigms, SE4H will need to evolve to ensure that quantum software is developed with consideration for human ethics, usability, and societal impact. Quantum computing also requires new models for how humans interact with software, given the complexity of quantum mechanics, making SE4H crucial in ensuring that these systems are accessible and understandable to diverse user groups.

Automatic Programming. Machine learning, particularly through deep neural networks and LLMs, represents the most significant amplification of human productivity in software engineering since its early days. These technologies are pushing the boundaries of what is possible by enabling new frontiers in automatic programming and transforming how software is written, tested, and maintained. They are also reshaping the landscape of quality and security, raising critical questions about how we ensure robust, secure, and reliable systems in the face of AI-generated code. These advancements also introduce new societal and legal issues, such as accountability and the ethical implications of automation in software development.

Automatic programming, using AI to generate code with minimal human input, has the potential to influence quantum computing development significantly. Quantum programming is inherently more complex than classical programming due to the nature of quantum mechanics, involving concepts like superposition, entanglement, and probabilistic outcomes. These factors make quantum

software development highly specialized and challenging. Automatic programming can help bridge this gap by generating quantum code from high-level descriptions, allowing developers who may not be experts in quantum computing to leverage the power of quantum computing. AI-driven tools can automate the creation and optimization of quantum circuits, reduce human error, and speed up the development process, making quantum computing more accessible to a broader range of developers and industries (see also *Ch-AI*).

On the other hand, quantum computing can influence automatic programming by enhancing the performance of AI algorithms used to generate code. Quantum computers have the potential to dramatically improve the efficiency of machine learning models that underlie automatic programming systems. Quantum algorithms could be used to optimize the search spaces involved in code generation, enabling faster and more efficient solutions to programming challenges. Additionally, quantum computing could allow automatic programming systems to tackle more complex problems, such as large-scale software optimization or real-time bug fixing, that are currently too resource-intensive for classical computing.

Software Security. The widespread integration of machine learning in software quality and security brings new societal and legal considerations [112]. As software systems become increasingly complex and large-scale, novel security challenges arise, underscoring the need for advanced methods in secure software engineering and cybersecurity [128].

Software security plays a vital role in the development of quantum computing, as the rise of quantum technologies introduces new vulnerabilities and security risks. With quantum computers having the potential to break widely used encryption algorithms, particularly those based on classical public-key cryptography, securing quantum software becomes an urgent priority. As quantum systems evolve, the integration of security measures, such as quantum-resistant cryptographic algorithms [16, 204, 205] and secure quantum communication protocols, will be crucial in protecting data and applications from breaches. Software security methodologies, including secure coding practices, vulnerability detection, and real-time threat monitoring, will need to adapt to the unique characteristics of quantum environments (see also *Ch-ST*), ensuring that quantum applications and their interactions with classical systems are secure from new types of cyber threats.

On the other hand, quantum computing can transform software security by enabling more advanced cryptographic techniques and faster detection of vulnerabilities. Quantum algorithms, such as Shor's algorithm, have the ability to factor large integers exponentially faster than classical algorithms, which poses a threat to current encryption methods. However, quantum cryptography, particularly quantum key distribution, offers an unprecedented level of security by leveraging the principles of quantum mechanics to detect eavesdropping and ensure secure communication channels. Additionally, quantum computing could enhance the speed and accuracy of security tools, allowing for the rapid identification of software vulnerabilities and the development of more robust defense mechanisms against emerging cyber threats.

Validation and Verification (V&V). Machine learning and AI are transforming the landscape of validation and verification in software engineering. The adaptive and evolving nature of AI-driven systems changes the traditional concepts of test input and oracles, as these systems can continuously learn and modify their behavior. Simultaneously, machine learning and generative AI open up new possibilities for automating the testing process, offering innovative approaches to test generation and analysis. There is a pressing need for a new conceptual framework that addresses the unique challenges of testing and analyzing AI-driven software systems, as well as further research into how ML and generative AI can be harnessed to improve testing and analysis.

V&V are critical aspects of software development, ensuring that systems perform as intended and meet specified requirements. In quantum computing, V&V methodologies are essential for managing the inherent complexity and probabilistic nature of quantum algorithms. As quantum programs

deal with qubits, superposition, and entanglement, traditional V&V methods must evolve to account for the unique characteristics of quantum systems. This includes developing tools for verifying the correctness of quantum circuits, identifying quantum-specific bugs, and ensuring that quantum systems behave reliably across different execution environments (see also *Ch-ST*). Enhanced V&V processes will play a crucial role in building trust in quantum applications, particularly as they become integrated into high-stakes domains like cryptography, pharmaceuticals, and optimization.

Conversely, quantum computing has the potential to revolutionize validation and verification by providing unprecedented computational power to handle complex verification tasks. Quantum computers can process vast amounts of data in parallel, potentially allowing them to verify intricate software systems far more efficiently than classical methods. For example, quantum algorithms could dramatically accelerate the analysis of software models, enabling faster identification of bugs, inconsistencies, or security vulnerabilities in both quantum and classical software. This could lead to the development of quantum-enhanced verification tools capable of validating complex configurations and simulations that are computationally intractable for classical systems. In this way, quantum computing can push the boundaries of what is possible in V&V, enabling deeper and more comprehensive analysis of large-scale software systems.

Sustainable Software Engineering. The concept of sustainable development now extends beyond traditional environmental concerns and encompasses software systems operating within cyber-physical spaces. Achieving sustainability in these systems requires innovative approaches to design, development, deployment, and maintenance that prioritize reducing ecological impact, improving resource efficiency, and promoting social responsibility.

Sustainable software engineering can influence the development of quantum computing by encouraging the design of quantum systems that minimize energy consumption and optimize resource usage. As quantum computing evolves, it is essential to address its potential environmental impact, particularly in the context of large-scale quantum processors that may require significant amounts of energy for cooling and operation. By applying principles of sustainability, researchers can develop quantum software and hardware that prioritize energy efficiency and minimize the ecological footprint. This could involve optimizing quantum algorithms to run on fewer qubits or for shorter durations, reducing the overall power consumption of quantum systems and contributing to more environmentally responsible computing practices (see also *Ch-PP* and *Ch-DP*).

Conversely, quantum computing has the potential to reshape sustainable software engineering by providing solutions to problems that are currently too complex or resource-intensive for classical computing. Quantum computing's ability to solve optimization problems more efficiently can lead to breakthroughs in areas such as energy distribution, climate modeling, and resource management. In this way, quantum computing could directly support the goals of sustainable development by providing more efficient solutions to critical sustainability challenges.

6 Conclusion and Challenges

In this work, a group of active researchers is currently addressing the challenges of QSE and analyzing the most recent advances in this field. This analysis reveals some certainties, which can be summarized as follows. First, no matter the discipline one focuses on, quantum software development requires new techniques compared to classical software development. These techniques have already begun to form the body of QSE. Second, quantum computers are reaching a development stage attracting industry attention. Therefore, empirical software engineering methods and techniques need to be revisited to meet industry expectations regarding quantum computing, making the development of QSE a priority.

To achieve this development, QSE's most noteworthy challenges have been identified. Next, it is necessary to identify synergies and dependencies between them further. This could help researchers

in the QSE domain to focus on the most needed aspects. By successfully addressing these challenges, QSE will be able to support the development of hybrid software systems beyond the NISQ era. The roadmap does not end here: quantum technology will continue to evolve, posing new challenges and opportunities for the QSE community.

References

[1] Ali J. Abhari, Arvin Faruque, Mohammad J. Dousti, Lukas Svec, Oana Catu, Amlan Chakrabati, Chen-Fu Chiang, Seth Vanderwilt, John Black, and Fred Chong. 2012. *Scaffold: Quantum Programming Language*. Technical Report. Department of Computer Science, Princeton University. Retrieved from https://www.cs.princeton.edu/research/techreps/TR-934-12

[2] Rui Abreu, João Paulo Fernandes, Luis Llana, and Guilherme Tavares. 2023. Metamorphic testing of oracle quantum programs. In *Proceedings of the 3rd International Workshop on Quantum Software Engineering (Q-SE '22)*. ACM, New York, NY, 16–23. DOI: https://doi.org/10.1145/3528230.3529189

[3] Muhammad Azeem Akbar, Arif Ali Khan, and Saima Rafi. 2023. A systematic decision-making framework for tackling quantum software engineering challenges. *Automated Software Engineering* 30, 2 (2023), 22. DOI: https://doi.org/10.1007/s10515-023-00389-7

[4] Muhammad Azeem Akbar, Arif Ali Khan, Mohammad Shameem, and Mohammad Nadeem. 2024. Genetic model-based success probability prediction of quantum software development projects. *Information and Software Technology* 165 (2024), 107352. DOI: https://doi.org/10.1016/j.infsof.2023.107352

[5] Mst Shamima Aktar, Peng Liang, Muhammad Waseem, Amjed Tahir, Aakash Ahmad, Beiqi Zhang, and Zengyang Li. 2025. Architecture decisions in quantum software systems: An empirical study on Stack Exchange and GitHub. *Information and Software Technology* 177 (2025), 107587. DOI: https://doi.org/10.1016/j.infsof.2024.107587

[6] Shaukat Ali, Paolo Arcaini, Xinyi Wang, and Tao Yue. 2021. Assessing the effectiveness of input and output coverage criteria for testing quantum programs. In *Proceedings of the 2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST '21)*. IEEE, 13–23. DOI: https://doi.org/10.1109/ICST49551.2021.00014

[7] Shaukat Ali and Tao Yue. 2020. Modeling quantum programs: Challenges, initial results, and research directions. In *Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software (APEQS '20)*. ACM, New York, NY, 14–21. DOI: https://doi.org/10.1145/3412451.3428499

[8] Shaukat Ali and Tao Yue. 2023. On the need of quantum-oriented paradigm. In *Proceedings of the 2nd International Workshop on Quantum Programming for Software Engineering (QP4SE '23)*. ACM, New York, NY, 17–20. DOI: https://doi.org/10.1145/3617570.3617868

[9] Shaukat Ali and Tao Yue. 2023. Quantum software testing: A brief introduction. In *Companion Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE-Companion '23)*. IEEE, 332–333. DOI: https://doi.org/10.1109/ICSE-Companion58688.2023.00093

[10] Shaukat Ali, Tao Yue, and Rui Abreu. 2022. When software engineering meets quantum computing. *Communications of the ACM* 65, 4 (2022), 84–88. DOI: https://doi.org/10.1145/3512340

[11] Diego Alonso, Pedro Sánchez, and Francisco Sánchez-Rubio. 2022. Engineering the development of quantum programs: Application to the Boolean satisfiability problem. In *Advances in Engineering Software*, Vol. 173, 103216. DOI: https://doi.org/10.1016/j.advengsoft.2022.103216

[12] Jaime Alvarado-Valiente, Javier Romero-Álvarez, Enrique Moguel, José García-Alonso, and Juan M. Murillo. 2024. Technological diversity of quantum computing providers: A comparative study and a proposal for API Gateway integration. *Software Quality Journal* 32, 1 (2024), 53–73. DOI: https://doi.org/10.1007/s11219-023-09633-5

[13] Jaime Alvarado-Valiente, Javier Romero-Álvarez, Enrique Moguel, Jose García-Alonso, and Juan M. Murillo. 2024. Orchestration for quantum services: The power of load balancing across multiple service providers. *Science of Computer Programming* 237 (2024), 103139. DOI: https://doi.org/10.1016/j.scico.2024.103139

[14] Joshua Ammermann, Wolfgang Mauerer, and Ina Schaefer. 2024. Towards view-based development of quantum software. arXiv:2406.18363. Retrieved from https://arxiv.org/abs/2406.18363

[15] Álvaro M. Aparicio-Morales, Enrique Moguel, Luis Mariano Bibbo, Alejandro Fernandez, Jose Garcia-Alonso, and Juan M. Murillo. 2024. An overview of quantum software engineering in Latin America. *Quantum Information Processing* 23, 11 (2024), 380. DOI: https://doi.org/10.1007/s11128-024-04586-5

[16] Johanna Barzen and Frank Leymann. 2024. Post-quantum security: Origin, fundamentals, and adoption. arXiv:2405.11885. Retrieved from https://arxiv.org/abs/2405.11885

[17] Martin Beisel, Johanna Barzen, Simon Garhofer, Frank Leymann, Felix Truger, Benjamin Weder, and Vladimir Yussupov. 2022. Quokka: A service ecosystem for workflow-based execution of variational quantum algorithms. In *Proceedings of the International Conference on Service-Oriented Computing*. Springer, 369–373. DOI: https://doi.org/10.1007/978-3-031-26507-5_35

[18] Jon Bentley. 1985. Programming pearls: Confessions of a coder. *Communications of the ACM* 28, 7 (1985), 671–679. DOI: https://doi.org/10.1145/3894.315112

[19] Ethan Bernstein and Umesh Vazirani. 1997. Quantum complexity theory. *SIAM Journal on Computing* 26, 5 (1997), 1411–1473. DOI: https://doi.org/10.1137/S0097539796300921

[20] Giuseppe Bisicchia, Jose García-Alonso, Juan M. Murillo, and Antonio Brogi. 2023. Distributing quantum computations, by shots. In *Proceedings of the International Conference on Service-Oriented Computing*. Springer, 363–377. DOI: https://doi.org/10.1007/978-3-031-48421-6_25

[21] Xavier Bonet-Monroig, Ryan Babbush, and Thomas E. O'Brien. 2020. Nearly optimal measurement scheduling for partial tomography of quantum states. *Physical Review X* 10, 3 (2020), 031064. DOI: https://doi.org/10.1103/PhysRevX.10.031064

[22] Dirk Bouwmeester and Anton Zeilinger. 2000. The physics of quantum information: Basic concepts. In *The Physics of Quantum Information: Quantum Cryptography, Quantum Teleportation, Quantum Computation*. Springer, 1–14. DOI: https://doi.org/10.1007/978-3-662-04209-0_1

[23] Fabian Bühler, Johanna Barzen, Martin Beisel, Daniel Georg, Frank Leymann, and Karoline Wild. 2023. Patterns for quantum software development. In *Proceedings of the 15th International Conference on Pervasive Patterns and Applications (PATTERNS '23)*, 30–39. DOI: https://doi.org/10.1145/3665870.3665871

[24] Anita D. Carleton, Erin Harper, John E. Robert, Mark H. Klein, Dionisio De Niz, Edward Desautels, John B. Goodenough, Charles Holland, Ipek Ozkaya, and Douglas Schmidt. 2021. *Architecting the Future of Software Engineering: A National Agenda for Software Engineering Research and Development*. Report. Software Engineering Institute, Carnegie Mellon University. Retrieved from https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=741193

[25] Marco Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C. Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, et al. 2021. Variational quantum algorithms. *Nature Reviews Physics* 3, 9 (2021), 625–644. DOI: https://doi.org/10.1038/s42254-021-00348-9

[26] Qihong Chen, Rúben Câmara, José Campos, André Souto, and Iftekhar Ahmed. 2023. The smelly eight: An empirical study on the prevalence of code smells in quantum computing. In *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE '23)*. IEEE, 358–370. DOI: https://doi.org/10.1109/ICSE48619.2023.00041

[27] Frederic T. Chong, Diana Franklin, and Margaret Martonosi. 2017. Programming languages and compiler design for realistic quantum hardware. *Nature* 549, 7671 (2017), 180–187. DOI: https://doi.org/10.1038/nature23459

[28] Matteo Ciniselli, Niccolò Puccinelli, Ketai Qiu, and Luca Di Grazia. 2024. From today's code to tomorrow's symphony: The AI transformation of developer's routine by 2030. arXiv:2405.12731. Retrieved from https://arxiv.org/abs/2405.12731

[29] John Clark and Susan Stepney. 2002. Proposed "Grand Challenge for Computing Research" Quantum Software Engineering. Retrieved from https://www.cs.york.ac.uk/quantum/sig/021108/qsegc.pdf

[30] José A. Cruz-Lemus, Luis A. Marcelo, and Mario Piattini. 2021. Towards a set of metrics for quantum circuits understandability. In *Proceedings of the Quality of Information and Communications Technology - 14th International Conference (QUATIC '21)*. Ana C. R. Paiva, Ana Rosa Cavalli, Paula Ventura Martins, and RicardoPérez-Castillo (Eds.), Communications in Computer and Information Science, Vol. 1439, Springer, 239–249. DOI: https://doi.org/10.1007/978-3-030-85347-1_18

[31] Steven A. Cuccaro, Thomas G. Draper, Samuel A. Kutin, and David Petrie Moulton. 2004. A new quantum ripple-carry addition circuit. arXiv:quant-ph/0410184. Retrieved from https://arxiv.org/abs/quant-ph/0410184

[32] Evandro Chagas Ribeiro Da Rosa and Rafael De Santiago. 2021. Ket quantum programming. *ACM Journal on Emerging Technologies in Computing Systems* 18, 1, Article 12 (Oct. 2021), 25 pages. DOI: https://doi.org/10.1145/3474224

[33] Antonio García de la Barrera, Ignacio García Rodríguez de Guzmán, Macario Polo, and Mario Piattini. 2023. Quantum software testing: State of the art. *Journal of Software: Evolution and Process* 35, 4 (2023). DOI: https://doi.org/10.1002/SMR.2419

[34] Antonio García de la Barrera Amo, Manuel A. Serrano, Ignacio García Rodríguez de Guzmán, Macario Polo, and Mario Piattini. 2022. Automatic generation of test circuits for the verification of Quantum deterministic algorithms. In *Proceedings of the 1st International Workshop on Quantum Programming for Software Engineering (QP4SE '22)*. Fabiano Pecorelli, Vita Santa Barletta, and Manuel A. Serrano (Eds.), ACM, New York, NY, 1–6. DOI: https://doi.org/10.1145/3549036.3562055

[35] Manuel De Stefano, Fabiano Pecorelli, Dario Di Nucci, Fabio Palomba, and Andrea De Lucia. 2022. Software engineering for quantum programming: How far are we? *Journal of Systems and Software* 190 (2022), 111326. DOI: https://doi.org/10.1016/j.jss.2022.111326

[36] A. Deshpande. 2022. Assessing the quantum-computing landscape. *Communications of the ACM* 65, 10 (2022), 57–65. DOI: https://doi.org/10.1145/3524109

[37] David Deutsch and Richard Jozsa. 1992. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* 439 (1992), 553–558. DOI: https://doi.org/10.1098/rspa.1992.0167

[38] Nivedita Dey, Mrityunjay Ghosh, Subhra Samir kundu, and Amlan Chakrabarti. 2020. QDLC – The quantum development life cycle. arXiv:2010.08053. Retrieved from https://arxiv.org/abs/2010.08053

[39] Nicolas Dupuis, Luca Buratti, Sanjay Vishwakarma, Aitana Viudes Forrat, David Kremer, Ismael Faro, Ruchir Puri, and Juan Cruz-Benito. 2024. Qiskit code assistant: Training LLMs for generating quantum computing code. arXiv:2405.19495. Retrieved from https://arxiv.org/abs/2405.19495

[40] Ana Díaz, Jaime Alvarado-Valiente, Javier Romero-Álvarez, Enrique Moguel, Jose Garcia-Alonso, Moisés Rodríguez, Ignacio García-Rodríguez, and Juan M. Murillo. 2024. Service engineering for quantum computing: Ensuring high-quality quantum services. *Information and Software Technology* (2024), 107643. DOI: https://doi.org/10.1016/j.infsof.2024.107643

[41] Ana Díaz-Muñoz, Moisés Rodríguez, and Mario Piattini. 2024. Towards a set of metrics for hybrid (quantum/classical) systems maintainability. *Journal of Universal Computer Science* 30, 1 (2024), 25–48. DOI: https://doi.org/10.3897/jucs.99348

[42] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP*, 1536–1547. DOI: https://doi.org/10.18653/V1/2020.FINDINGS-EMNLP.139

[43] Richard P. Feynman. 2018. Simulating physics with computers. In *Feynman and Computation*. CRC Press, 133–153. DOI: https://doi.org/10.1007/BF02650179

[44] Daniel Fortunato, José Campos, and Rui Abreu. 2022. Mutation testing of quantum programs: A case study with Qiskit. *IEEE Transactions on Quantum Engineering* 3 (2022), 1–17. DOI: https://doi.org/10.1109/TQE.2022.3195061

[45] Daniel Fortunato, José Campos, and Rui Abreu. 2022. Mutation testing of quantum programs written in QISKit. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings (ICSE '22)*. ACM, New York, NY, 358–359. DOI: https://doi.org/10.1145/3510454.3528649

[46] Daniel Fortunato, José Campos, and Rui Abreu. 2022. QMutPy: A mutation testing tool for quantum algorithms and applications in Qiskit. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '22)*. ACM, New York, NY, 797–800. DOI: https://doi.org/10.1145/3533767.3543296

[47] World Economic Forum. 2022. Quantum computing governance principles. Retrieved from https://www3.weforum.org/docs/WEF_Quantum_Computing_2022.pdf

[48] Rafael Fresno-Aranda, Pablo Fernández, Amador Durán, and Antonio Ruiz-Cortés. 2022. Semi-automated capacity analysis of limitation-aware microservices architectures. In *Proceedings of the International Conference on the Economics of Grids, Clouds, Systems, and Services*. Springer, 75–88. DOI: https://doi.org/10.1007/978-3-031-29315-3_7

[49] Rafael Fresno-Aranda, Pablo Fernandez, Antonio Gamez-Diaz, Amador Duran, and Antonio Ruiz-Cortes. 2025. Pricing4APIs: A rigorous model for RESTful API pricings. *Computer Standards & Interfaces* 91 (2025), 103878. DOI: https://doi.org/10.1016/j.csi.2024.103878

[50] X. Fu, Jintao Yu, Xing Su, Hanru Jiang, Hua Wu, Fucheng Cheng, Xi Deng, Jinrong Zhang, Lei Jin, Yihang Yang, et al. 2021. Quingo: A programming framework for heterogeneous quantum-classical computing with NISQ features. *ACM Transactions on Quantum Computing* 2, 4, Article 19 (Dec. 2021), 37 pages. DOI: https://doi.org/10.1145/3483528

[51] Alireza Furutanpey, Johanna Barzen, Marvin Bechtold, Schahram Dustdar, Frank Leymann, Philipp Raith, and Felix Truger. 2023. Architectural vision for quantum computing in the edge-cloud continuum. In *Proceedings of the 2023 IEEE International Conference on Quantum Software (QSW '23)*. DOI: https://doi.org/10.1109/QSW59989.2023.00021

[52] Antonio Gamez-Diaz, Pablo Fernandez, Antonio Ruiz-Cortés, Pedro J. Molina, Nikhil Kolekar, Prithpal Bhogill, Madhurranjan Mohaan, and Francisco Méndez. 2019. The role of limitations and SLAs in the API industry. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19)*, 1006–1014. DOI: https://doi.org/10.1145/3338906.3340445

[53] Jose Garcia-Alonso, Javier Rojo, David Valencia, Enrique Moguel, Javier Berrocal, and Juan Manuel Murillo. 2021. Quantum software as a service through a quantum API gateway. *IEEE Internet Computing* 26, 1 (2021), 34–41. DOI: https://doi.org/10.1109/MIC.2021.3132688

[54] D. García-Martín, E. Ribas, S. Carrazza, J. I. Latorre, and G. Sierra. 2020. The prime state and its quantum relatives. *Quantum* 4 (Dec. 2020), 371. DOI: https://doi.org/10.22331/q-2020-12-11-371

[55] Alan Geller. 2020. Introducing quantum intermediate representation (QIR). *Q# Blog*. Retrieved September 2020 from https://devblogs.microsoft.com/qsharp/introducing-quantum-intermediate-representation-qir

[56] Felix Gemeinhardt, Martin Eisenberg, Stefan Klikovits, and Manuel Wimmer. 2023. Model-driven optimization for quantum program synthesis with MOMoT. In *Proceedings of the 2023 ACM/IEEE International Conference on Model*

Driven Engineering Languages and Systems Companion (MODELS-C '23). IEEE, 614–621. DOI: https://doi.org/10.1109/MODELS-C59198.2023.00100

[57] Felix Gemeinhardt, Antonio Garmendia, and Manuel Wimmer. 2021. Towards model-driven quantum software engineering. In *Proceedings of the 2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE '21)*. IEEE, 13–15. DOI: https://doi.org/10.1109/Q-SE52541.2021.00010

[58] Felix Gemeinhardt, Antonio Garmendia, Manuel Wimmer, and Robert Wille. 2024. A model-driven framework for composition-based quantum circuit design. *ACM Transactions on Quantum Computing* (2024). DOI: https://doi.org/10.1145/3688856

[59] Ilie-Daniel Gheorghe-Pop, Nikolay Tcholtchev, Tom Ritter, and Manfred Hauswirth. 2020. Quantum DevOps: Towards reliable and applicable NISQ quantum computing. In *Proceedings of the 2020 IEEE GLOBECOM Workshops (GC Wkshps '20)*, 1–6. DOI: https://doi.org/10.1109/GCWkshps50303.2020.9367411

[60] András Gilyén, Srinivasan Arunachalam, and Nathan Wiebe. 2019. Optimizing quantum optimization algorithms via faster quantum gradient computation. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 1425–1444. DOI: https://doi.org/10.1137/1.9781611975482.87

[61] Jonathan Grattage. 2011. An overview of QML with a concrete implementation in Haskell. *Electronic Notes in Theoretical Computer Science* 270, 1 (2011), 165–174. DOI: https://doi.org/10.1016/j.entcs.2011.01.015

[62] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. 2013. Quipper: A scalable quantum programming language. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '13)*, Hans-Juergen Boehm and Cormac Flanagan (Eds.), ACM, New York, NY, 333–342. DOI: https://doi.org/10.1145/2491956.2462177

[63] Lov K. Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, 212–219. DOI: https://doi.org/10.48550/arXiv.quant-ph/9605043

[64] Lov K. Grover. 1998. Quantum computers can search rapidly by using almost any transformation. *Physical Review Letters* 80, 19 (May 1998), 4329–4332. DOI: https://doi.org/10.1103/physrevlett.80.4329

[65] Xiaoyu Guo, Takahiro Muta, and Jianjun Zhao. 2024. Quantum circuit Ansatz: Patterns of abstraction and reuse of quantum algorithm design. In *Proceedings of the 2024 IEEE International Conference on Quantum Software (QSW '24)*. IEEE, 69–80. DOI: https://doi.org/10.1109/QSW62656.2024.00021

[66] Xiaoyu Guo, Jianjun Zhao, and Pengzhan Zhao. 2024. On repairing quantum programs using ChatGPT. In *Proceedings of the 2024 IEEE/ACM 5th International Workshop on Quantum Software Engineering (Q-SE '24)*. ACM, New York, NY. DOI: https://doi.org/10.1145/3643667.3648223

[67] Majid Haghparast, Tommi Mikkonen, Jukka K. Nurminen, and Vlad Stirbu. 2023. Quantum software engineering challenges from developers' perspective: Mapping research challenges to the proposed workflow model. In *Proceedings of the 2023 IEEE International Conference on Quantum Computing and Engineering (QCE '23)*, 173–176. DOI: https://doi.org/10.1109/QCE57702.2023.10204

[68] Jose Luis Hevia, Guido Peterssen, and Mario Piattini. 2024. Quantum software development risks. *Quantum Information and Computation* 24, 5&6 (2024), 455–467. DOI: https://doi.org/10.26421/QIC24.5-6-5

[69] Jack D. Hidary. 2021. Dirac notation. In *Quantum Computing: An Applied Approach*, 377–381. DOI: https://doi.org/10.1007/978-3-030-83274-2_14

[70] Shahin Honarvar, Mohammad Reza Mousavi, and Rajagopal Nagarajan. 2020. Property-based testing of quantum programs in Q#. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW '20)*. ACM, New York, NY, 430–435. DOI: https://doi.org/10.1145/3387940.3391459

[71] Ryszard Horodecki, Paweł Horodecki, Michał Horodecki, and Karol Horodecki. 2009. Quantum entanglement. *Reviews of Modern Physics* 81, 2 (2009), 865. DOI: https://doi.org/10.1103/RevModPhys.81.865

[72] Yipeng Huang and Margaret Martonosi. 2019. Statistical assertions for validating patterns and finding bugs in quantum programs. In *Proceedings of the 46th International Symposium on Computer Architecture (ISCA '19)*. ACM, New York, NY, 541–553. DOI: https://doi.org/10.1145/3307650.3322213

[73] Thomas Häner, Mathias Soeken, Martin Roetteler, and Krysta M. Svore. 2018. Quantum circuits for floating-point arithmetic. arXiv:1807.02023. Retrieved from https://arxiv.org/abs/1807.02023

[74] IBM. 2016. IBM makes quantum computing available on IBM cloud to accelerate innovation. Retrieved from https://uk.newsroom.ibm.com/2016-May-04-IBM-Makes-Quantum-Computing-Available-on-IBM-Cloud-to-Accelerate-Innovation

[75] Shahab Iranmanesh, Hossein Aghababa, and Kazim Fouladi. 2024. Gate optimization of NEQR quantum circuits via PPRM transformation. arXiv:2409.14629. Retrieved from https://arxiv.org/abs/2409.14629

[76] Victoria Jackson, Bogdan Vasilescu, Daniel Russo, Paul Ralph, Maliheh Izadi, Rafael Prikladnicki, Sarah D'Angelo, Sarah Inman, Anielle Lisboa, and Andre van der Hoek. 2024. Creativity, generative AI, and software development: A research agenda. arXiv:2406.01966. Retrieved from https://arxiv.org/abs/2406.01966

[77] Luis Jiménez-Navajas, Ricardo Pérez-Castillo, and Mario Piattini. 2021. KDM to UML model transformation for quantum software modernization. In *Proceedings of the International Conference on the Quality of Information and Communications Technology*. Springer, 211–224. DOI: https://doi.org/10.1007/978-3-030-85347-1_16

[78] Luis Jiménez-Navajas, Fabian Bühler, Frank Leymann, Ricardo Pérez-Castillo, Mario Piattini, and Daniel Vietz. 2024. Quantum software development: A survey. *Quantum Information and Computation* 24, 7&8 (2024), 609–642. DOI: https://doi.org/10.26421/QIC24.7-8-4

[79] Tiancheng Jin and Jianjun Zhao. 2023. ScaffML: A quantum behavioral interface specification language for Scaffold. In *Proceedings of the 2023 IEEE International Conference on Quantum Software (QSW '23)*. IEEE, 128–137. DOI: https://doi.org/10.48550/arXiv.2306.06468

[80] Jose Jose Campos and Andre Souto. 2021. QBugs: A collection of reproducible bugs in quantum algorithms and a supporting infrastructure to enable controlled quantum software testing and debugging experiments. In *Proceedings of the 2nd International Workshop on Quantum Software Engineering (Q-SE '21)*, 28–32. DOI: https://doi.org/10.1109/Q-SE52541.2021.00013

[81] Richard Jozsa and Noah Linden. 2003. On the role of entanglement in quantum-computational speed-up. *Proceedings of the Royal Society of London. Series A. Mathematical, Physical and Engineering Sciences* 459 (2003), 2011–2032. Retrieved from https://www.jstor.org/stable/3560059

[82] Arif Ali Khan, Aakash Ahmad, Muhammad Waseem, Peng Liang, Mahdi Fahmideh, Tommi Mikkonen, and Pekka Abrahamsson. 2023. Software architecture for quantum computing systems — A systematic review. *Journal of Systems and Software* 201 (2023), 111682. DOI: https://doi.org/10.1016/j.jss.2023.111682

[83] Arif Ali Khan, Muhammad Azeem Akbar, Aakash Ahmad, Mahdi Fahmideh, Mohammad Shameem, Valtteri Lahtinen, Muhammad Waseem, and Tommi Mikkonen. 2022. Agile practices for quantum software development: Practitioners perspectives. arXiv:2210.09825. Retrieved from https://arxiv.org/abs/2210.09825

[84] Krishn Vishwas Kher, Ishan Joshi, Bharat Chandra Mukkavalli, Lei Zhang, and M. V. Panduranga Rao. 2023. Automatic diagnosis of quantum software bug fix motifs. In *Proceedings of the 35th International Conference on Software Engineering and Knowledge Engineering (SEKE '23)*, 97–102. DOI: https://doi.org/10.18293/SEKE2023-196

[85] Ray LaPierre. 2021. *Quantum Parallelism and Computational Complexity*. Springer International Publishing, Cham, 139–148. DOI: https://doi.org/10.1007/978-3-030-69318-3_10

[86] B. Lenahan. 2021. Quantum adoption: Lessons learned from a quantum strategist. Retrieved from https://quantumstrategyinstitute.com/2021/10/24/quantum-adoption-lessons-learned-from-a-quantum-strategist/QuantumStrategyInstitute

[87] Frank Leymann. 2019. Towards a pattern language for quantum algorithms. In *Quantum Technology and Optimization Problems*, Lecture Notes in Computer Science (LNCS), Vol. 11413, Springer International Publishing, Cham, 218–230. DOI: https://doi.org/10.1007/978-3-030-14082-3_19

[88] Frank Leymann and Johanna Barzen. 2020. The bitter truth about gate-based quantum algorithms in the NISQ era. *Quantum Science and Technology* 5, 4 (Sep. 2020), 044007. DOI: https://doi.org/10.1088/2058-9565/abae7d

[89] Gushu Li, Li Zhou, Nengkun Yu, Yufei Ding, Mingsheng Ying, and Yuan Xie. 2020. Projection-based runtime assertions for testing and debugging quantum programs. *Proceedings of the ACM on Programming Languages* 4, OOPSLA, Article 150 (Nov. 2020), 29 pages. DOI: https://doi.org/10.1145/3428218

[90] Yuechen Li, Hanyu Pei, Linzhi Huang, Beibei Yin, and Kai-Yuan Cai. 2024. Automatic repair of quantum programs via unitary operation. *ACM Transactions on Software Engineering and Methodology* 33, 6 (2024), 1–43.

[91] Yangjia Li and Mingsheng Ying. 2014. Debugging quantum processes using monitoring measurements. *Physical Review A* 89, 4 (2014), 042338. DOI: https://doi.org/10.1103/PhysRevA.89.042338

[92] Ji Liu, Gregory T. Byrd, and Huiyang Zhou. 2020. Quantum circuits for dynamic runtime assertions in quantum computation. In *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1017–1030. DOI: https://doi.org/10.1109/LCA.2019.2935049

[93] Peixun Long and Jianjun Zhao. 2024. Equivalence, identity, and unitarity checking in black-box testing of quantum programs. *Journal of Systems and Software* 211 (2024), 112000. DOI: https://doi.org/10.1016/j.jss.2024.112000

[94] Peixun Long and Jianjun Zhao. 2024. Testing multi-subroutine quantum programs: From unit testing to integration testing. *ACM Transactions on Software Engineering and Methodology* 33, 6 (2024), 1–6. DOI: https://doi.org/10.1145/3656339

[95] Junjie Luo and Jianjun Zhao. 2023. Enhancing code safety in quantum intermediate representation. In *Proceedings of the 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE '23)*. IEEE, 1771–1775. DOI: https://doi.org/10.1109/ASE56229.2023.00195

[96] Junjie Luo and Jianjun Zhao. 2025. Formalization of quantum intermediate representations for code safety. *Journal of Systems and Software* 219 (2025), 112236.

[97] Alexander Mandl, Johanna Barzen, Marvin Bechtold, Michael Keckeisen, Frank Leymann, and Patrick K. S. Vaudrevange. 2024. Linear structure of training samples in quantum neural network applications. In *Service-Oriented*

Computing – ICSOC 2023 Workshops. Flavia Monti, Pierluigi Plebani, Naouel Moha, Hye-young Paik, Johanna Barzen, Gowri Ramachandran, Devis Bianchini, Damian A. Tamburri, and Massimo Mecella (Eds.), Springer Nature, Singapore, 150–161. DOI: https://doi.org/10.1007/978-981-97-0989-2_12

[98] Philip Maymin. 1997. Extending the Lambda calculus to express randomized and quantumized algorithms. arXiv:quant-ph/9612052. Retrieved from https://arxiv.org/abs/quant-ph/9612052

[99] B. Mayoh, E. Tyugu, and J. Penjam. 2013. *Constraint Programming*. Springer, Berlin. Retrieved from https://books.google.es/books?id=B0aqCAAAQBAJ

[100] Eñaut Mendiluze, Shaukat Ali, Paolo Arcaini, and Tao Yue. 2022. Muskit: A mutation analysis tool for quantum software testing. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE '21)*. IEEE Press, 1266–1270. DOI: https://doi.org/10.1109/ASE51524.2021.9678563

[101] Sara Ayman Metwalli and Rodney Van Meter. 2024. Testing and debugging quantum circuits. *IEEE Transactions on Quantum Engineering* 5 (2024). DOI: https://doi.org/10.1109/TQE.2024.3374879

[102] Andriy Miranskyy. 2022. Using quantum computers to speed up dynamic testing of software. In *Proceedings of the 1st International Workshop on Quantum Programming for Software Engineering*, 26–31. DOI: https://doi.org/10.1145/3549036.3562061

[103] Andriy Miranskyy, Mushahid Khan, Jean Paul Latyr Faye, and Udson C. Mendes. 2022. Quantum computing for software engineering: Prospects. In *Proceedings of the 1st International Workshop on Quantum Programming for Software Engineering*, 22–25. DOI: https://doi.org/10.1145/3549036.3562060

[104] Andriy Miranskyy, Mushahid Khan, and Udson Mendes. 2024. Comparing algorithms for loading classical datasets into quantum memory. In *Proceedings of the IEEE International Conference on Quantum Computing and Engineering (QCE '24)*, Vol. 2. DOI: https://doi.org/10.1109/QCE60285.2024.10284

[105] Andriy Miranskyy and Lei Zhang. 2019. On testing quantum programs. In *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER '19)*. IEEE Press, 57–60. DOI: https://doi.org/10.1109/ICSE-NIER.2019.00023

[106] Andriy Miranskyy, Lei Zhang, and Javad Doliskani. 2020. Is your quantum program bug-free? In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER '20)*. ACM, New York, NY, 29–32. DOI: https://doi.org/10.1145/3377816.3381731

[107] Andriy Miranskyy, Lei Zhang, and Javad Doliskani. 2021. On testing and debugging quantum software. arXiv:2103.09172. Retrieved from https://arxiv.org/abs/2103.09172

[108] Enrique Moguel, Javier Berrocal, Jose García-Alonso, and Juan Manuel Murillo. 2020. A roadmap for quantum software engineering: Applying the lessons learned from the classics. Retrieved from https://ceur-ws.org/Vol-2705/short1.pdf

[109] Enrique Moguel, José Antonio Parejo, Antonio Ruiz-Cortés, Jose Garcia-Alonso, and Juan Manuel Murillo. 2024. Quantum software experiments: A reporting and laboratory package structure guidelines. arXiv:2405.04192. Retrieved from https://arxiv.org/abs/2405.04192

[110] Enrique Moguel, Javier Rojo, David Valencia, Javier Berrocal, Jose Garcia-Alonso, and Juan M. Murillo. 2022. Quantum service-oriented computing: Current landscape and challenges. *Software Quality Journal* 30, 4 (2022), 983–1002. DOI: https://doi.org/10.1007/s11219-022-09589-y

[111] Armin Moin, Moharram Challenger, Atta Badii, and Stephan Günnemann. 2021. MDE4QAI: Towards model-driven engineering for quantum artificial intelligence. arXiv:2107.06708. Retrieved from https://arxiv.org/abs/2107.06708

[112] Facundo Molina and Alessandra Gorla. 2024. Test Oracle Automation in the era of LLMs. arXiv:2405.12766. Retrieved from https://arxiv.org/abs/2405.12766

[113] Asmar Muqeet, Shaukat Ali, and Paolo Arcaini. 2024. Approximating stochastic quantum noise through genetic programming. In *Search-based software engineering*. Gunel Jahangirova and Foutse Khomh (Eds.), Springer Nature, Cham, 56–62. DOI: https://doi.org/10.1007/978-3-031-64573-0_5

[114] Asmar Muqeet, Shaukat Ali, and Paolo Arcaini. 2024. Quantum program testing through commuting Pauli strings on IBM's quantum computers. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24)*. ACM, New York, NY, 2130–2141. DOI: https://doi.org/10.1145/3691620.3695275

[115] Asmar Muqeet, Shaukat Ali, Tao Yue, and Paolo Arcaini. 2024. A machine learning-based error mitigation approach for reliable software development on IBM's quantum computers. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE '24)*. ACM, New York, NY, 80–91. DOI: https://doi.org/10.1145/3663529.3663830

[116] Asmar Muqeet, Tao Yue, Shaukat Ali, and Paolo Arcaini. 2024. Mitigating noise in quantum software testing using machine learning. *IEEE Transactions on Software Engineering* (2024), 1–15. DOI: https://doi.org/10.1109/TSE.2024.3462974

[117] Kentaro Murakami and Jianjun Zhao. 2022. Automated synthesis of quantum circuits using neural network. In *Proceedings of the 2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS '22)*. IEEE, 694–702. DOI: https://doi.org/10.1109/QRS57517.2022.00075

[118] Juan M. Murillo, Jose Garcia-Alonso, Enrique Moguel, Johanna Barzen, Frank Leymann, Shaukat Ali, Tao Yue, Paolo Arcaini, Ricardo Pérez Castillo, Ignacio García Rodríguez de Guzmán, et al. 2024. Challenges of quantum software engineering for the next decade: The road ahead. arXiv:2404.06825. Retrieved from https://arxiv.org/abs/2404.06825

[119] Pranav K. Nayak, Krishn V. Kher, M. Bharat Chandra, M. V. Panduranga Rao, and Lei Zhang. 2023. Q-PAC: Automated detection of quantum bug-fix patterns. arXiv:2311.17705. Retrieved from https://arxiv.org/abs/2311.17705

[120] Siyuan Niu, Adrien Suau, Gabriel Staffelbach, and Aida Todri-Sanial. 2020. A hardware-aware heuristic for the qubit mapping problem in the NISQ era. *IEEE Transactions on Quantum Engineering* 1 (2020), 1–14. DOI: https://doi.org/10.1109/TQE.2020.3026544

[121] Noah H. Oldfield, Christoph Laaber, Tao Yue, and Shaukat Ali. 2024. Faster and better quantum software testing through specification reduction and projective measurements. arXiv:2405.15450. Retrieved from https://arxiv.org/abs/2405.15450

[122] Moses Openja, Mohammad Mehdi Morovati, Le An, Foutse Khomh, and Mouna Abidi. 2022. Technical debts and faults in open-source quantum software systems: An empirical study. *Journal of Systems and Software* 193 (2022), 111458. DOI: https://doi.org/10.1016/J.JSS.2022.111458

[123] Scott Pakin. 2016. A quantum macro assembler. In *Proceedings of the 2016 IEEE High Performance Extreme Computing Conference (HPEC '16)*, 1–8. DOI: https://doi.org/10.1109/HPEC.2016.7761637

[124] Matteo Paltenghi and Michael Pradel. 2022. Bugs in quantum computing platforms: An empirical study. *Proceedings of the ACM on Programming Languages* 6, OOPSLA1 (2022), 1–27. DOI: https://doi.org/10.1145/3527330

[125] Matteo Paltenghi and Michael Pradel. 2023. MorphQ: Metamorphic testing of the Qiskit quantum computing platform. In *Proceedings of the 45th International Conference on Software Engineering (ICSE '23)*. IEEE Press, 2413–2424. DOI: https://doi.org/10.1109/ICSE48619.2023.00202

[126] Matteo Paltenghi and Michael Pradel. 2024. Analyzing quantum programs with LintQ: A static analysis framework for Qiskit. *Proceedings of the ACM on Software Engineering* 1, FSE, Article 95 (Jul. 2024), 23 pages. DOI: https://doi.org/10.1145/3660802

[127] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. 2007. Service-oriented computing: State of the art and research challenges. *Computer* 40, 11 (2007), 38–45. DOI: https://doi.org/10.1109/MC.2007.400

[128] Nikhil Patnaik, Joseph Hallett, and Awais Rashid. 2024. Saltzer & Schroeder for 2030: Security engineering principles in a world of AI. arXiv:2407.05710. Retrieved from https://arxiv.org/abs/2407.05710

[129] Yuxiang Peng, Jacob Young, Pengyu Liu, and Xiaodi Wu. 2024. SimuQ: A framework for programming quantum Hamiltonian simulation with analog compilation. *Proceedings of the ACM on Programming Languages* 8, POPL, Article 81 (Jan. 2024), 31 pages. DOI: https://doi.org/10.1145/3632923

[130] Ricardo Pérez-Castillo, Luis Jiménez-Navajas, and Mario Piattini. 2022. QRev: Migrating quantum code towards hybrid information systems. *Software Quality Journal* 30, 2 (2022), 551–580. DOI: https://doi.org/10.1007/s11219-021-09574-x

[131] Carlos A. Pérez-Delgado and Hector G. Perez-Gonzalez. 2020. Towards a quantum software modeling language. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW '20)*. ACM, New York, NY, 442–444. DOI: https://doi.org/10.1145/3387940.3392183

[132] Mauro Pezzè, Matteo Ciniselli, Luca Di Grazia, Niccolò Puccinelli, and Ketai Qiu. 2024. The trailer of the ACM 2030 roadmap for software engineering. In *Proceedings of the 2030 Software Engineering Roadmap Workshop*. Retrieved from https://www.inf.usi.ch/faculty/pezze/media/SE2030SENreport.pdf

[133] Mario Piattini, Manuel Serrano, Ricardo Perez-Castillo, Guido Petersen, and Jose Luis Hevia. 2021. Toward a quantum software engineering. *IT Professional* 23, 1 (2021), 62–66. DOI: https://doi.org/10.1109/MITP.2020.3019522

[134] Furkan Polat, Hasan Tuncer, Armin Moin, and Moharram Challenger. 2024. Model-driven engineering for quantum programming: A case study on ground state energy calculation. arXiv:2405.17065. Retrieved from https://arxiv.org/abs/2405.17065

[135] Gabriel Joseph Pontolillo and Mohammad Reza Mousavi. 2024. Delta debugging for property-based regression testing of quantum programs. In *Proceedings of the 5th ACM/IEEE International Workshop on Quantum Software Engineering (Q-SE '24)*. ACM, New York, NY, 1–8. DOI: https://doi.org/10.1145/3643667.3648219

[136] John Preskill. 2018. Quantum computing in the NISQ era and beyond. *Quantum* 2 (2018), 79. DOI: https://doi.org/10.22331/q-2018-08-06-79

[137] Ricardo Pérez-Castillo, Miriam Fernández-Osuna, Jose Antonio Cruz-Lemus, and Mario Piattini. 2024. A preliminary study of the usage of design patterns in quantum software. In *Proceedings of the 2024 IEEE/ACM 5th International Workshop on Quantum Software Engineering (Q-SE '24)*. Retrieved from https://ieeexplore.ieee.org/document/10649787

[138] Ricardo Pérez-Castillo, Luis Jiménez-Navajas, Iván Cantalejo, and Mario Piattini. 2023. Generation of classical-quantum code from UML models. In *Proceedings of the 2023 IEEE International Conference on Quantum Computing and Engineering (QCE '23)*, 165–168. DOI: https://doi.org/10.1109/QCE57702.2023.10202

[139] Ricardo Pérez-Castillo, Luis Jiménez-Navajas, and Mario Piattini. 2021. Modelling quantum circuits with UML. In *Proceedings of the 2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE '23)*, 7–12. DOI: https://doi.org/10.1109/Q-SE52541.2021.00009

[140] Ricardo Pérez-Castillo, Manuel A. Serrano, José A. Cruz-Lemus, and Mario Piattini. 2024. Guidelines to use the incremental commitment spiral model for developing quantum-classical systems. *Quantum Information and Computation* 24, 1&2 (2024), 71–88. Retrieved from https://www.rintonpress.com/xxqic24/qic-24-12/0071-0088.pdf

[141] Ricardo Pérez-Castillo, Manuel A. Serrano, and Mario Piattini. 2021. Software modernization to embrace quantum technology. In *Advances in Engineering Software*, Vol. 151, 102933. DOI: https://doi.org/10.1016/j.advengsoft.2020.102933

[142] Florian Richoux, Jean-François Baffier, and Philippe Codognet. 2023. Learning QUBO models for quantum annealing: A constraint-based approach. In *Computational Science – ICCS 2023*. Jiří Mikyška, Clélia de Mulatier, Maciej Paszynski, Valeria V. Krzhizhanovskaya, Jack J. Dongarra, and Peter M.A. Sloot (Eds.), Springer Nature, Cham, 153–167. DOI: https://doi.org/10.1007/978-3-031-36030-5_12

[143] Javier Rojo, David Valencia, Javier Berrocal, Enrique Moguel, Jose Garcia-Alonso, and Juan Manuel Murillo Rodriguez. 2021. Trials and tribulations of developing hybrid quantum-classical microservices systems. arXiv:2105.04421. Retrieved from https://arxiv.org/abs/2105.04421

[144] Javier Romero-Álvarez, Jaime Alvarado-Valiente, Enrique Moguel, and Jose Garcia-Alonso. 2023. Quantum web services: Development and deployment. In *Web Engineering*. Irene Garrigós, Juan Manuel Murillo Rodríguez, and Manuel Wimmer (Eds.), Springer Nature, Cham, 421–423. DOI: https://doi.org/10.1007/978-3-031-34444-2_39

[145] Javier Romero-Álvarez, Jaime Alvarado-Valiente, Enrique Moguel, José García-Alonso, and Juan M. Murillo. 2022. Using open API for the development of hybrid classical-quantum services. In *Proceedings of the International Conference on Service-Oriented Computing*. Springer, 364–368. DOI: https://doi.org/10.1007/978-3-031-26507-5_34

[146] Javier Romero-Álvarez, Jaime Alvarado-Valiente, Enrique Moguel, Jose Garcia-Alonso, and Juan M. Murillo. 2024. Enabling continuous deployment techniques for quantum services. *Software: Practice and Experience* 54, 8 (2024), 1491–1515. DOI: https://doi.org/10.1002/spe.3326

[147] Javier Romero-Álvarez, Jaime Alvarado-Valiente, Enrique Moguel, José Garcia-Alonso, and Juan M. Murillo. 2024. *Quantum Service-Oriented Computing: A Proposal for Quantum Software as a Service*. River Publishers.

[148] Marie Salm, Johanna Barzen, Frank Leymann, and Philipp Wundrack. 2022. Optimizing the prioritization of compiled quantum circuits by machine learning approaches. In *Symposium and Summer School on Service-Oriented Computing*. Springer, 161–181. DOI: https://doi.org/10.1007/978-3-031-18304-1_9

[149] Marie Salm, Johanna Barzen, Frank Leymann, and Philipp Wundrack. 2023. How to select quantum compilers and quantum computers before compilation. In *Proceedings of the 13th International Conference on Cloud Computing and Services Science (CLOSER '23)*, 172–183. DOI: https://doi.org/10.5220/0011775300003488

[150] Johannes Sametinger. 1997. *Software Engineering with Reusable Components*. Springer. DOI: https://doi.org/10.1007/978-3-662-03345-6

[151] Javier Sanchez-Rivero, Daniel Talaván, Jose Garcia-Alonso, Antonio Ruiz-Cortés, and Juan Manuel Murillo. 2023. Operating with quantum integers: An efficient 'multiples of' oracle. In *Service-Oriented Computing*. Marco Aiello, Johanna Barzen, Schahram Dustdar, and Frank Leymann (Eds.), Springer Nature, Cham, 105–124. DOI: https://doi.org/10.1007/978-3-031-45728-9_7

[152] Javier Sanchez-Rivero, Daniel Talaván, Jose Garcia-Alonso, Antonio Ruiz-Cortés, and Juan Manuel Murillo. 2023. Automatic generation of an efficient less-than oracle for quantum amplitude amplification. In *Proceedings of the 2023 IEEE/ACM 4th International Workshop on Quantum Software Engineering (Q-SE '23)*, 26–33. DOI: https://doi.org/10.1109/Q-SE59154.2023.00011

[153] Javier Sanchez-Rivero, Daniel Talaván, Jose Garcia-Alonso, Antonio Ruiz-Cortés, and Juan Manuel Murillo. 2023. Automatic generation of efficient oracles: The less-than case. SSRN. DOI: https://doi.org/10.2139/ssrn.4594664

[154] Javier Sanchez-Rivero, Daniel Talaván, Jose Garcia-Alonso, Antonio Ruiz-Cortés, and Juan Manuel Murillo. 2023. Some initial guidelines for building reusable quantum oracles. In *Services and Quantum Software - 21st International Conference on Service-Oriented Computing*. DOI: https://doi.org/10.1007/978-981-97-0989-2_16

[155] Yuki Sano, Kosuke Mitarai, Naoki Yamamoto, and Naoki Ishikawa. 2024. Accelerating Grover adaptive search: Qubit and gate count reduction strategies with higher order formulations. *IEEE Transactions on Quantum Engineering* 5 (2024), 1–12. DOI: https://doi.org/10.1109/TQE.2024.3393437

[156] Lorenzo Saraiva, Edward Hermann Haeusler, Vaston G. Costa, and Marcos Kalinowski. 2021. Non-functional requirements for quantum programs. In *Q-SET@QCE*, 89–73. Retrieved from https://ceur-ws.org/Vol-3008/paper4.pdf

[157] Naoto Sato and Ryota Katsube. 2024. Locating buggy segments in quantum program debugging. In *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER '24)*. ACM, New York, NY, 26–31. DOI: https://doi.org/10.1145/3639476.3639761

[158] D. C. Schmidt. 2006. Guest editor's introduction: Model-driven engineering. *Computer* 39, 2 (2006), 25–31. DOI: https://doi.org/10.1109/MC.2006.58

[159] Raphael Seidel, Nikolay Tcholtchev, Sebastian Bock, Colin Kai-Uwe Becker, and Manfred Hauswirth. 2021. Efficient floating point arithmetic for quantum computers. arXiv:2112.10537 (2021). Retrieved from https://arxiv.org/abs/2112.10537

[160] Manuel A. Serrano, José A. Cruz-Lemus, Ricardo Perez-Castillo, and Mario Piattini. 2022. Quantum software components and platforms: Overview and quality assessment. *ACM Computing Surveys* 55, 8, Article 164 (Dec. 2022), 31 pages. DOI: https://doi.org/10.1145/3548679

[161] Minqi Shao and Jianjun Zhao. 2024. A coverage-guided testing framework for quantum neural networks. arXiv:2411.02450. Retrieved from https://arxiv.org/abs/2411.02450

[162] Ruslan Shaydulin, Caleb Thomas, and Paige Rodeghero. 2020. Making quantum computing open: Lessons from open source projects. In *Proceedings of the 42nd International Conference on Software Engineering, Workshops (ICSE '20)*. ACM, New York, NY, 451–455. DOI: https://doi.org/10.1145/3387940.3391471

[163] Peter W. Shor. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review* 41, 2 (1999), 303–332. DOI: https://doi.org/10.1137/S0036144598347011

[164] Daniel R. Simon. 1997. On the power of quantum computation. *SIAM Journal on Computing* 26, 5 (1997), 1474–1483. DOI: https://doi.org/10.1137/S0097539796298637

[165] Janakan Sivaloganathan, Ainaz Jamshidi, Andriy Miranskyy, and Lei Zhang. 2024. Automating quantum software maintenance: Flakiness detection and root cause analysis. arXiv:2410.23578. Retrieved from https://arxiv.org/abs/2410.23578

[166] Robert S. Smith, Michael J. Curtis, and William J. Zeng. 2017. A practical quantum instruction set architecture. arXiv:1608.03355. Retrieved from https://arxiv.org/abs/1608.03355

[167] John A. Smolin, Jay M. Gambetta, and Graeme Smith. 2012. Efficient method for computing the maximum-likelihood quantum state from measurements with additive gaussian noise. *Physical Review Letters* 108, 7 (2012), 070502. DOI: https://doi.org/10.1103/PhysRevLett.108.070502

[168] Susan Stepney, John Clark, Andy Tyrell, Colin Johnson, Jonathan Timmis, Derek Partridge, Andy Adamatsky, and Robert Smith. 2004. Journeys in non-classical computation: Grand challenges in computing research, 29–32. Retrieved from https://www.cs.york.ac.uk/nature/gc7/journeys.pdf

[169] V. Stirbu, M. Haghparast, M. Waseem, N. Dayama, and T. Mikkonen. 2023. Full-stack quantum software in practice: Ecosystem, stakeholders and challenges. In *Proceedings of the 2023 IEEE International Conference on Quantum Computing and Engineering (QCE '23)*, 177–180. DOI: https://doi.org/10.1109/QCE57702.2023.10205

[170] Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. 2018. Q#: Enabling scalable quantum computing and development with a high-level DSL. In *Proceedings of the Real World Domain Specific Languages Workshop 2018 (RWDSL '18)*. ACM, New York, NY, Article 7, 10 pages. DOI: https://doi.org/10.1145/3183895.3183901

[171] Valerio Terragni, Partha Roop, and Kelly Blincoe. 2024. The future of software engineering in an AI-driven world. arXiv:2406.07737. Retrieved from https://arxiv.org/abs/2406.07737

[172] Javier Troya, Nathalie Moreno, Manuel F. Bertoa, and Antonio Vallecillo. 2021. Uncertainty representation in software models: A survey. *Software and Systems Modeling* 20, 4 (2021), 1183–1213. DOI: https://doi.org/10.1007/s10270-020-00842-1

[173] Felix Truger, Martin Beisel, Johanna Barzen, Frank Leymann, and Vladimir Yussupov. 2022. Selection and optimization of hyperparameters in warm-started quantum optimization for the MaxCut problem. *Electronics* 11, 7 (2022), 1033. DOI: https://doi.org/10.3390/electronics11071033

[174] Eñaut Mendiluze Usandizaga, Tao Yue, Paolo Arcaini, and Shaukat Ali. 2023. Which quantum circuit mutants shall be used? An empirical evaluation of quantum circuit mutations. arXiv:2311.16913. Retrieved from https://arxiv.org/abs/2311.16913

[175] David Valencia, Jose Garcia-Alonso, Javier Rojo, Enrique Moguel, Javier Berrocal, and Juan Manuel Murillo. 2021. Hybrid classical-quantum software services systems: Exploration of the rough edges. In *Quality of Information and Communications Technology*. Ana C. R. Paiva, Ana Rosa Cavalli, Paula Ventura Martins, and Ricardo Pérez-Castillo (Eds.), Springer International Publishing, Cham, 225–238. DOI: https://doi.org/10.1007/978-3-030-85347-1_17

[176] Tamás Varga, Yaiza Aragonés-Soria, and Manuel Oriol. 2024. Quantum types: Going beyond qubits and quantum gates. In *Proceedings of the 5th ACM/IEEE International Workshop on Quantum Software Engineering (Q-SE '24)*. DOI: https://doi.org/10.1145/3643667.3648225

[177] Jiyuan Wang, Ming Gao, Yu Jiang, Jian-Guang Lou, Yue Gao, Dongmei Zhang, and Jiaguang Sun. 2018. QuanFuzz: Fuzz testing of quantum program. arXiv:1810.10310. Retrieved from https://arxiv.org/abs/1810.10310

[178] Jiyuan Wang, Qian Zhang, Guoqing Harry Xu, and Miryung Kim. 2021. QDiff: Differential testing of quantum software stacks. In *Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE '21)*, 692–704. DOI: https://doi.org/10.1109/ASE51524.2021.9678792

[179] Qing Wang, Junjie Wang, Mingyang Li, Yawen Wang, and Zhe Liu. 2024. A roadmap for software testing in open collaborative development environments. arXiv:2406.05438. Retrieved from https://arxiv.org/abs/2406.05438

[180] Xinyi Wang, Shaukat Ali, Aitor Arrieta, Paolo Arcaini, and Maite Arratibel. 2024. Application of quantum extreme learning machines for qos prediction of elevators' software in an industrial context. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE '24)*. ACM, New York, NY, 399–410. DOI: https://doi.org/10.1145/3663529.3663859

[181] Xinyi Wang, Shaukat Ali, Tao Yue, and Paolo Arcaini. 2024. Quantum approximate optimization algorithm for test case optimization. *IEEE Transactions on Software Engineering* (2024), 1–16. DOI: https://doi.org/10.1109/TSE.2024.3479421

[182] Xinyi Wang, Paolo Arcaini, Tao Yue, and Shaukat Ali. 2021. Application of combinatorial testing to quantum programs. In *Proceedings of the 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS '21)*, 179–188. DOI: https://doi.org/10.1109/QRS54544.2021.00029

[183] Xinyi Wang, Paolo Arcaini, Tao Yue, and Shaukat Ali. 2021. Generating failing test suites for quantum programs with search. In *Search-Based Software Engineering*. Una-May O'Reilly and Xavier Devroey (Eds.), Springer International Publishing, Cham, 9–25. DOI: https://doi.org/10.1007/978-3-030-88106-1_2

[184] Xinyi Wang, Paolo Arcaini, Tao Yue, and Shaukat Ali. 2022. Quito: A coverage-guided test generator for quantum programs. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (Melbourne, Australia) (ASE '21)*. IEEE Press, 1237–1241. DOI: https://doi.org/10.1109/ASE51524.2021.9678798

[185] Xinyi Wang, Paolo Arcaini, Tao Yue, and Shaukat Ali. 2022. QuSBT: Search-based testing of quantum programs. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings (ICSE '22)*. ACM, New York, NY, 173–177. DOI: https://doi.org/10.1145/3510454.3516839

[186] Xinyi Wang, Paolo Arcaini, Tao Yue, and Shaukat Ali. 2023. QuCAT: A combinatorial testing tool for quantum software. In *Proceedings of the 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE '23)*. IEEE Computer Society, Los Alamitos, CA, 2066–2069. DOI: https://doi.org/10.1109/ASE56229.2023.00062

[187] Xinyi Wang, Asmar Muqeet, Tao Yue, Shaukat Ali, and Paolo Arcaini. 2024. Test case minimization with quantum annealers. *ACM Transactions on Software Engineering and Methodology* 34, 1 (Jul. 2024), 1–24. DOI: https://doi.org/10.1145/3680467

[188] Xinyi Wang, Tongxuan Yu, Paolo Arcaini, Tao Yue, and Shaukat Ali. 2022. Mutation-based test generation for quantum programs with multi-objective search. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '22)*. ACM, New York, NY, 1345–1353. DOI: https://doi.org/10.1145/3512290.3528869

[189] Benjamin Weder, Johanna Barzen, Martin Beisel, and Frank Leymann. 2023. Provenance-preserving analysis and rewrite of quantum workflows for hybrid quantum algorithms. *SN Computer Science* 4, 3 (2023), 233. DOI: https://doi.org/10.1007/s42979-022-01625-9

[190] Benjamin Weder, Johanna Barzen, Frank Leymann, and Daniel Vietz. 2022. *Quantum software development lifecycle*. Springer International Publishing, Cham, 61–83. DOI: https://doi.org/10.1007/978-3-031-05324-5_4

[191] Benjamin Weder, Johanna Barzen, Frank Leymann, and Michael Zimmermann. 2021. Hybrid quantum applications need two orchestrations in superposition: A software architecture perspective. In *Proceedings of the 2021 IEEE International Conference on Web Services (ICWS '21)*. IEEE, 1–13. DOI: https://doi.org/10.1109/ICWS53863.2021.00015

[192] Benjamin Weder, Uwe Breitenbücher, Frank Leymann, and Karoline Wild. 2020. Integrating quantum computing into workflow modeling and execution. In *Proceedings of the 2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC '20)*. IEEE, 279–291. DOI: https://doi.org/10.1109/UCC48980.2020.00046

[193] Manuela Weigold, Johanna Barzen, Frank Leymann, and Daniel Vietz. 2021. Patterns for hybrid quantum algorithms. In *Symposium and Summer School on Service-Oriented Computing*, Vol. 1429. Springer, 34–51. DOI: https://doi.org/10.1007/978-3-030-87568-8_2

[194] Y. S. Weinstein, M. A. Pravia, E. M. Fortunato, S. Lloyd, and D. G. Cory. 2001. Implementation of the quantum Fourier transform. *Physical Review Letters* 86, 9 (Feb. 2001), 1889–1891. DOI: https://doi.org/10.1103/PhysRevLett.86.1889

[195] Nathan Wiebe and Vadym Kliuchnikov. 2013. Floating point representations in quantum circuit synthesis. *New Journal of Physics* 15, 9 (Sep. 2013), 093041. DOI: https://doi.org/10.1088/1367-2630/15/9/093041

[196] Karoline Wild, Uwe Breitenbücher, Lukas Harzenetter, Frank Leymann, Daniel Vietz, and Michael Zimmermann. 2020. TOSCA4QC: Two modeling styles for TOSCA to automate the deployment and orchestration of quantum applications. In *Proceedings of the 2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC '20)*. IEEE, 125–134. DOI: https://doi.org/10.1109/EDOC49727.2020.00024

[197] Shangzhou Xia and Jianjun Zhao. 2023. Static entanglement analysis of quantum programs. In *Proceedings of the 2023 IEEE/ACM 4th International Workshop on Quantum Software Engineering (Q-SE '23)*. IEEE, 42–49. DOI: https://doi.org/10.1109/Q-SE59154.2023.00013

[198] Shangzhou Xia, Jianjun Zhao, Fuyuan Zhang, and Xiaoyu Guo. 2024. Concolic testing of quantum programs. arXiv:2405.04860. Retrieved from https://arxiv.org/abs/2405.04860

[199] Jiaming Ye, Shangzhou Xia, Fuyuan Zhang, Paolo Arcaini, Lei Ma, Jianjun Zhao, and Fuyuki Ishikawa. 2023. QuraTest: Integrating quantum specific features in quantum program testing. In *Proceedings of the 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE '23)*, 1149–1161. DOI: https://doi.org/10.1109/ASE56229.2023.00196

[200] Haibo Yu and Jianjun Zhao. 2025. The quantum program dependence graph and its uses in quantum software development. In *Proceedings of the 2025 IEEE/ACM 6th International Workshop on Quantum Software Engineering (Q-SE '25)*.

[201] Tao Yue, Shaukat Ali, and Paolo Arcaini. 2023. Towards quantum software requirements engineering. In *Proceedings of the 2023 IEEE International Conference on Quantum Computing and Engineering (QCE '23)*, Vol. 2, 161–164. DOI: https://doi.org/10.1109/QCE57702.2023.10201

[202] Tao Yue, Wolfgang Mauerer, Shaukat Ali, and Davide Taibi. 2023. *Challenges and Opportunities in Quantum Software Architecture*. Springer Nature Switzerland, Cham, 1–23. DOI: https://doi.org/10.1007/978-3-031-36847-9_1

[203] Lei Zhang and Andriy Miranskyy. 2024. Automated flakiness detection in quantum software bug reports. In *Proceedings of the IEEE International Conference on Quantum Computing and Engineering (QCE '24)*, Vol. 2. DOI: https://doi.org/10.48550/arXiv.2408.05331 To appear.

[204] Lei Zhang, Andriy Miranskyy, and Walid Rjaibi. 2021. Quantum advantage and the Y2K bug: A comparison. *IEEE Software* 38, 2 (2021), 80–87. DOI: https://doi.org/10.1109/MS.2020.2985321

[205] Lei Zhang, Andriy Miranskyy, Walid Rjaibi, Greg Stager, Michael Gray, and John Peck. 2023. Making existing software quantum safe: A case study on IBM Db2. *Information and Software Technology* 161 (2023), 107249. DOI: https://doi.org/10.1016/j.infsof.2023.107249

[206] Lei Zhang, Mahsa Radnejad, and Andriy Miranskyy. 2023. Identifying flakiness in quantum programs. In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '23)*. IEEE, 1–7. DOI: https://doi.org/10.1109/ESEM56168.2023.10304850

[207] Jianjun Zhao. 2020. Quantum software engineering: Landscapes and horizons. arXiv:2007.07047. Retrieved from https://arxiv.org/abs/2007.07047

[208] Jianjun Zhao. 2021. Some size and structure metrics for quantum software. In *Proceedings of the 2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE '21)*. IEEE, 22–27. DOI: https://doi.org/10.1109/Q-SE52541.2021.00012

[209] Jianjun Zhao. 2023. On refactoring quantum programs in Q#. In *Proceedings of the 2023 IEEE International Conference on Quantum Computing and Engineering (QCE '23)*, Vol. 2. IEEE, 169–172. DOI: https://doi.org/10.1109/QCE57702.2023.10203

[210] Jianjun Zhao. 2024. Towards an architecture description language for hybrid quantum-classical systems. In *Proceedings of the 2024 IEEE International Conference on Quantum Software (QSW '24)*. IEEE, 19–23. DOI: https://doi.org/10.1109/QSW62656.2024.00016

[211] Pengzhan Zhao, Zhongtao Miao, Shuhan Lan, and Jianjun Zhao. 2023. Bugs4Q: A benchmark of existing bugs to enable controlled testing and debugging studies for quantum programs. *Journal of Systems and Software* 205, C (Nov. 2023), 13 pages. DOI: https://doi.org/10.1016/j.jss.2023.111805

[212] Pengzhan Zhao, Xiongfei Wu, Zhuo Li, and Jianjun Zhao. 2023. QCchecker: Detecting bugs in quantum programs via static analysis. In *Proceedings of the 2023 IEEE/ACM 4th International Workshop on Quantum Software Engineering (Q-SE '23)*. IEEE, 50–57. DOI: https://doi.org/10.1109/Q-SE59154.2023.00014

[213] Pengzhan Zhao, Xiongfei Wu, Junjie Luo, Zhuo Li, and Jianjun Zhao. 2023. An empirical study of bugs in quantum machine learning frameworks. In *Proceedings of the 2023 IEEE International Conference on Quantum Software (QSW '23)*, 68–75. DOI: https://doi.org/10.1109/QSW59989.2023.00018

[214] Pengzhan Zhao, Jianjun Zhao, and Lei Ma. 2021. Identifying bug patterns in quantum programs. In *Proceedings of the 2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE '21)*. IEEE, 16–21. DOI: https://doi.org/10.1109/Q-SE52541.2021.00011

[215] Xudong Zhao, Xiaolong Xu, Lianyong Qi, Xiaoyu Xia, Muhammad Bilal, Wenwen Gong, and Huaizhen Kou. 2024. Unraveling quantum computing system architectures: An extensive survey of cutting-edge paradigms. *Information and Software Technology* 167 (2024), 107380. DOI: https://doi.org/10.1016/j.infsof.2023.107380

[216] Huiyang Zhou and Gregory T. Byrd. 2019. Quantum circuits for dynamic runtime assertions in quantum computation. *IEEE Computer Architecture Letters* 18, 2 (2019), 111–114. DOI: https://doi.org/10.1109/LCA.2019.2935049