Scientific Research Publishing

Journal of Quantum Information Science, 2024, 14, 234-258
https://www.scirp.org/journal/jqis
ISSN Online: 2162-576X
ISSN Print: 2162-5751

Quantum Software Engineering: Best Practices from Classical to Quantum Approaches

Abdullah Ibrahim S. Alsalman[1,2]

[1]Executive Office, Riyadh Region Municipality, Riyadh, Saudi Arabia
[2]Saudi Quantum Computing Association, Riyadh, Saudi Arabia
Email: abdullh.70@gmail.com

How to cite this paper: Alsalman, A.I.S. (2024) Quantum Software Engineering: Best Practices from Classical to Quantum Approaches. *Journal of Quantum Information Science*, **14**, 234-258.
https://doi.org/10.4236/jqis.2024.144010

Received: September 9, 2024
Accepted: December 3, 2024
Published: December 6, 2024

Abstract

As quantum computing transitions from a theoretical domain to a practical technology, many aspects of established practice in software engineering are being faced with new challenges. Quantum Software Engineering has been developed to address the peculiar needs that arise with quantum systems' dependable, scalable, and fault-tolerant software development. The present paper critically reviews how traditional software engineering methodologies can be reshaped to fit into the quantum field. This also entails providing some critical contributions: frameworks to integrate classical and quantum systems, new error mitigation techniques, and the development of quantum-specific testing and debugging tools. In this respect, best practices have been recommended to ensure that future quantum software can harness the evolving capabilities of quantum hardware with continued performance, reliability, and scalability. The work is supposed to act as a foundational guide for the researcher and developer as quantum computing approaches widespread scientific and industrial adoption.

Keywords

Quantum Software Engineering, Quantum Computing, Quantum Algorithms, Quantum Software Development Lifecycle, Error Mitigation, Classical-Quantum Integration, Quantum Debugging, Scalable Quantum Systems, Fault Tolerant Computing

1. Introduction

Classical computing has always been at the core of modern technological development, using binary digits for information processing and storage. In recent decades, advances in classical computing have enormously accelerated technological

advances in various fields, including artificial intelligence, cryptography, and massive data processing. However, classical computing is restricted to a great extent with increasing computational difficulties. These tasks could be more problematic for activities such as simulating molecular interactions, optimizing extensive networks, and factoring large integers applied to cryptography. These problems often require resources that grow exponentially, which makes them impractical to solve computationally, even for the most advanced classical supercomputers [1].

The limitations inherent in classic computing systems have sparked a growing interest in quantum computing—a new computational paradigm that naturally solves some problems impossible for a classic machine. Quantum computers take the central ideas of quantum mechanics (superposition, entanglement, and interference) to information processing in some really new ways. Whereas a classical bit has to be one of two discrete states, 0 or 1, a qubit is the basic unit of computing a superposition that can be in both states simultaneously. By leveraging those quantum features, quantum computers, by harnessing unique quantum characteristics, can carry out certain computations at speeds far beyond those of classical computing, leading to breakthroughs in cryptography, materials science, and machine learning [2].

Even with its capacity for transformation, attaining the complete functionalities of quantum computing entails considerable obstacles while concurrently providing substantial prospects for innovation. The path to functional quantum computing involves advancements in hardware and requires developing robust, scalable, and fault-tolerant software. It is this need that called upon the establishment of Quantum Software Engineering as a branch of study to fulfill the unique requirements of quantum systems [3]. Classical software engineering is well understood, but it cannot be applied directly to quantum environments because quantum mechanics principles are intrinsically different. For example, quantum software has to cope with decoherence and noise, problems that barely exist in classical computing. In addition, the inherently probabilistic nature of quantum states brings further complications into debugging and testing quantum algorithms.

Quantum Software Engineering has consequently emerged as an essential field that is responsible for modifying established software engineering principles to align with the quantum realm [4]. This encompasses the design, development, testing, and maintenance of software that mainly addresses the needs of quantum computing infrastructures. Additionally, QSE offers a systematic process for quantum software lifecycle management, ensuring that quantum applications perfectly integrate with classical systems, optimize quantum resources, and remain flexible to evolve continuously with developments in quantum technologies. As quantum computing is transitioning from a theoretical approach into practical usage, developing dependable and well-structured quantum software will be essential.

This will, in particular, depend on efficient, scalable software development that can fully unlock quantum hardware capabilities for future success and broad

impact across many sectors and scientific fields. In this work, the state of the art in quantum software engineering is discussed, and also how the methodologies of classical software engineering could be adapted for the quantum domain. Such topics will be explored in the areas of error mitigation techniques, hybrid classical-quantum system integration, and quantum software lifecycle management. We will give a general overview to let researchers and developers take these tools on to tackle such a complex and transformational challenge as quantum computing.

2. Why Does Classical Software Engineering Need Adaptation?

With the advent of quantum computing as a powerful paradigm, it is increasingly evident that adapted software engineering methodologies will be required. Classical Software Engineering (CSE) has been the backbone of modern computation, through which progress in many fields has become possible via structured development processes, testing protocols, debugging techniques, and optimization methods [5]. However, quantum computing introduces fundamental changes that render many of these traditional practices at least questionable. This section highlights why CSE needs adaptation for the quantum domain, pinpointing where classical techniques fall short in the face of quantum phenomena.

2.1. Quantum Information Representation: Qubits vs. Bits

Classical computing operates with binary bits, representing information as 0 or 1. In contrast, quantum computing operates with qubits, simultaneously representing states in superpositions that exist as 0 and 1. This fundamental difference implies that all classical data structures, algorithms, and storage methods must be rethought entirely for use in quantum computing. Classical software engineering methods, based on definite binary states, will require more inherent support for representing and handling the continuous and probabilistic properties unique to quantum information [6].

2.2. Non-Deterministic Nature of Quantum Operations

Classical software engineers rely on deterministic behavior while they test and debug; for the same inputs, a classical program should always produce the same outputs. In dramatic contrast, quantum algorithms must be fundamentally probabilistic due to the indeterminacy inherent in quantum mechanics. The output of a quantum program can only be statistically interpreted, and it usually requires numerous executions to achieve consistent output. Thus, testing methods, assuming repeatability, based on classical approaches will need adaptation due to this complexity and the need for statistical validation.

2.3. Error and Noise Sensitivity

Classically, computing systems have always relied on conventional strategies for achieving fault tolerance, most of which revolve around simple error detection and correction techniques [7]. Quantum computing systems are susceptible to

decoherence and noise, which can cause the state of the qubits to deteriorate or change with external environmental interference. Software engineering principles conventionally developed for handling hardware-software reliability are inefficient in quantum regimes because error mitigation and correction difficulties are strongly interwoven with the algorithm under study. This requirement entails the incorporation of quantum error correction codes (QECC) alongside error-mitigation strategies throughout each phase of the software development lifecycle.

2.4. Testing and Debugging in Quantum Systems

Classical testing frameworks in software engineering rely on extensive simulations and thorough debugging. With quantum software, additional complications arise since a quantum state collapses as soon as it undergoes observation due to the measurement problem. Therefore, the so-called intermediate states, widely used in classical debugging techniques, cannot be observed directly anymore [8]. This challenge calls for new debugging methodologies incorporating quantum simulators and shadow copies of quantum states to trace errors without disturbing the course of computation.

2.5. Algorithmic Complexity: Exponential Gains and Pitfalls

Quantum algorithms have the potential to offer exponential speedups relative to their classical counterparts. This advantage is fundamentally tied to the quantum phenomenon of entanglement. Unlike classical systems where each subsystem remains in a definite state, quantum systems can exist in entangled states, vastly increasing the space of possible solutions and contributing to the speedup seen in algorithms like Shor's algorithm, which drastically outperforms classical algorithms in factoring large numbers [9]. However, extending classical algorithmic complexity theory to quantum computing presents significant challenges. The complexity of quantum algorithms is influenced not only by time complexity but also by quantum-specific factors like qubit entanglement, coherence times, and gate fidelities. As a result, software engineers must reconsider traditional optimization techniques, as some approaches effective in classical computing cannot be directly transferred to quantum contexts due to these unique constraints.

2.6. Hybrid Classical-Quantum Systems

In the near future, quantum computing is expected to operate in tandem with classical computing systems, acting as an accelerator for specific applications such as optimization and cryptography. This hybrid model leverages the strengths of both systems, requiring the effective integration of classical and quantum processors. As discussed in recent studies, quantum processing units (QPUs) can serve as accelerators in high-performance computing (HPC) systems, where specialized kernels are used to offload select workloads while maintaining traditional computing infrastructure [10]. However, this integration presents unique challenges in terms of communication protocols, data interchange formats, and task orchestration

methodologies. Classical software engineering practices for system integration, data flow management, and resource scheduling must be adapted to effectively handle interactions between classical CPUs and quantum processors. In particular, the logical differences between conventional processors and QPUs necessitate innovative frameworks and operating systems to manage these resources and ensure efficient performance.

3. Fundamentals of Quantum Software Engineering

3.1. Overview of Quantum Computing

Quantum computing is an area that is evolving at a rapid pace and is bound to revolutionize both computational and problem-solving capabilities beyond those possible with conventional classical computing. Leveraging quantum mechanics, quantum computers manipulate qubits, which, due to superposition and entanglement, can occupy multiple states at the same time. The unique property of qubits allows quantum computers to execute certain computations exponentially faster than classical computers, mainly for problems that involve complex analysis of data, optimization, and cryptographic functions. As such, quantum computing has much potential to impact major issues in chemistry, materials science, finance, and machine learning. Functional quantum computing has emerged at the forefront of most research and development agendas globally, with notable improvements seen in the last few years. Major technology companies, academic institutions, and governments have invested extensive resources into the optimization of quantum hardware, algorithms, and software, driving the field forward at an unprecedented pace and fueling optimism for future success.

3.2. Definition

Software engineering has been defined as "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software" IEEE Standard Glossary of Software Engineering Terminology, 1990 [11]. By extension, Quantum Software Engineering (QSE) would be the pragmatic application of the principles of engineering in developing, operating, and maintaining software that would be applied on quantum computing systems. This means the leveraging of superposition and entanglement powers from quantum mechanics for the execution of hard computation tasks effectively and efficiently. QSE covers everything from the design of quantum algorithms to the implementation and testing of quantum software applications, with the intention of making them trustworthy and optimizing them on quantum hardware.

3.3. Importance of Quantum Software Engineering

QSE is more than just a discipline that acts as a driving force for the revolutionary potential of quantum computing. While quantum computing is gradually moving from a research model into a phase of actual implementation, skilled quantum software engineers are in greater demand. Industry leaders, research institutions,

and governments are investing massively in this field, believing that quantum computing cannot succeed without sophisticated quantum software. This ramp-up in demand indicates that QSE lies at the heart of the advance in quantum technology. The importance of QSE underlines how quantum software, worthy of trusted solutions, must be created, scalable, efficient, and use the full potential of quantum hardware.

QSE bridges the gap from quantum theory to practical implementation. The field of QSE requires new, innovative methods of programming, debugging, and optimizing quantum algorithms so that they run correctly and efficiently on hardware. Lousy software engineering methodologies would imply the application of only a fraction of the extraordinary promise of quantum computers. Good quantum software allows researchers and developers to define and execute challenging quantum algorithms that realize cryptography, materials science, and artificial intelligence breakthroughs. QSE is a discipline and, at the same time, a critical factor in realizing the full potential of quantum computing; it will make quantum systems serve efficiently to allow unrivaled advances in many scientific and technological fields. Addressing these unique challenges of quantum mechanics, QSE fosters the creation of robust and scalable software to form the core of a bright future in computation with many exciting promises.

3.4. Key Challenges

It introduces new challenges when development in the quantum computing field is advanced, and these need to be understood so that complete benefits from quantum computing can be realized through development. Issues that will precipitate problems include defining stakeholder needs and transformation into technical requirements, among others, where the high risks and undefined benefits make it difficult for this technology to be adopted. Nevertheless, quantum software design is equally heating up because of a lack of standardization in architectural description languages and design patterns, requiring new ways of integrating quantum components. Establishing dependable quantum software brings challenges from fault models, testing, debugging, and verification aspects. Moreover, quantum software maintenance brings a challenge with modification strategies and integrity preservation, while component reuse reveals the need for sound management methodologies in quantum software development. It brings along two challenges that involve extensive research and development to establish the reliability and effectiveness of quantum software systems [12].

3.5. Knowledge Requirements

The complicated and fascinating area of QSE requires an in-depth understanding of quantum physics, computer science, and mathematics. Adequate knowledge of linear algebra is to be known, particularly about vector spaces, linear transformations, eigenvalues, and eigenvectors. Most importantly, matrix mathematics capability is much needed, including manipulations with tensors and Kronecker

products [13]. Beyond this, the operation with complex numbers must be understood because quantum states are often expressed through complex amplitudes. Quantum State Engineering depends heavily on the foundational principles of basic quantum mechanics. Superposition, entanglement, and quantum state measurements are the base concepts behind quantum computing and are the bedrock of QSE. These mathematical and physical postulates will unavoidably underpin the conception of quantum software and hardware if developed in this field [14].

Several elements and concepts currently under development are needed to make quantum software. Mastery is set to be inclusive, from knowing quantum gates and their functionality to how they are implemented [15]. Not only that, but QSE requires expertise in quantum programming languages and quantum frameworks—Qiskit by IBM, Cirq by Google, and Quipper, which is a functional programming quantum language.

The overall analysis of the industry connotes successful QSE careers and their relevance to interdisciplinary competencies. Besides quantum-specific ones, professionals are expected to provide evidence of classical programming competencies, statistical data analysis, and practical hands-on laboratory experience. The latter is significant for a classical software engineer migrating into QSE. Of course, knowledge of IBM's Qiskit, Google's Cirq, and Microsoft's Q# programming environments must be familiarized. Such tools let developers create and check quantum algorithms, allowing deep insight into the different quantum computing paradigms. Interaction with quantum hardware and simulators is essential [14]. Experience in internships and collaborative projects that provide hands-on interaction with quantum systems adds richness to the engineer's learning. This should be further complemented by continuous professional development through workshops, online courses, and, most importantly, being current with the latest happenings within quantum technologies. Such heterogeneous training will guarantee theoretical and practical proficiency for software engineers who will keep the pace of the quantum revolution.

It requires a comprehensive set of skills, and one has to keep learning and adapting to new tools and approaches. New evidence related to the growing demand for work roles, such as quantum algorithm developers and error correction scientists, who require deep theoretical knowledge along with applied problem-solving skills [16], supports this. The integration of all these diversified competencies will advance the development of resilient quantum software and foster effective interdisciplinary collaboration that is helpful for innovation within the quantum ecosystem.

4. Quantum Software Development Lifecycle

The Quantum Software Development Lifecycle (QSDL) maps an overall framework for the disciplined development of quantum software, integrating classical and quantum approaches. Organized life cycle models usually consist of requirements

analysis, design, implementation, testing, and maintenance—manifest for both identified processes on classical and quantum software. In this process of quantum computing advancement, there is an ever-growing need for a systematic approach toward software development to overcome such unique challenges with quantum technologies with high-quality quantum applications. Quantum software development represents new worries: particular design languages, specialized implementation patterns for quantum, testing tools at early stages, and more complex maintenance tasks due to the continuous evolution of quantum hardware support [17]. Therefore, this QSDL would give software engineers clear, detailed guidance regarding integrating classical and quantum computing error correction methods and iterative improvements. The following paragraphs detail more of the elements and phases of QSDL, as well as a section detailing how vital the subject is to QSE.

4.1. Integration with Classical Computing

One of the critical ingredients of QSDL is the integration of quantum computing seamlessly with classical computing resources. Since quantum computers will accelerate a fraction of computation inside a larger classical context, the development of quantum software can communicate well with the classical setting for devising quantum-classical splitting: deciding which tasks are to be divided by quantum or classical processors, depending on needs. Integrating the benefits deriving from both paradigms increases the system's overall efficiency. Recent literature underlines that the orchestration phase is effective in quantum software development, able to manage and guide different portions of code and to offload classical tasks into classic computing infrastructures, defining quantum tasks into environments optimized for quantum processing. This orchestration ensures that classical code is managed by traditional systems and quantum code is executed over quantum-optimized platforms—effectively harnessing the advantages of both systems [18]. This prudent allocation and execution of resources are vital in the faithful execution of hybrid quantum-classical computing.

4.2. Enhanced Error Correction and Mitigation Strategies

Quantum computing has lots of key challenges, among them handling errors and inherent noises in quantum systems. In the development and realization of the QSDL framework, a lot of attention is paid to effective error correction and mitigation strategies in all the phases of the software life cycle. Strategies for this target include resilience to errors in the development of quantum algorithms, error-correcting codes, and runtime error mitigation. Ensuring reliability and precision in quantum computations forms the core and backbone for deploying any quantum application efficiently. Recently, quantum error mitigation-related developments emphasize the need to incorporate various approaches that will be required to develop fault-tolerant quantum computation. For instance, some noise-reduction techniques, such as Zero-Noise Extrapolation (ZNE), have been suggested whereby

the results of executions at different levels of noise are extrapolated to approximate a noisy-free result. Coupled with learning-based error mitigation strategies that adapt to different noise levels, using the classical training datasets, this methodology is a significant step toward the preservation of quantum computation accuracy, especially within noisy intermediate-scale quantum systems [19]. In this manner, these methods ensure greater fidelity when running quantum algorithms, opening up the path to practical quantum-computing applications.

4.3. Advanced Testing and Validation Techniques

The validation of the developed quantum software has unique challenges due to the non-deterministic nature of quantum computations. Advanced testing methodologies related to those have been integrated into the QSDL to further best practices in traditional software testing. This encompasses simulation-based testing for the correctness of quantum algorithms on classical simulators and hardware-in-the-loop testing to validate performance under realistic conditions. This is also where quantum-specific debugging tools for quantum code become essential.

Since these tests deal with the probabilistic outcomes of quantum programs, running multiple experiments is necessary to ensure that results follow expected distributions. While useful, classic simulators suffer from a challenge or impossibility in the performance of extensive simulations concerning exponentially increased complexity about the number of qubits. It has been proposed that dynamic assertions be used for quantum state assertion without collapse to solve these issues. However, applications are restricted only to a few features, such as superposition and entanglement. Other techniques, such as mutation testing, property-based testing, or bug benchmarking, have been adapted or developed mainly in quantum programming with coverage criteria or benchmark test suites to make the quantum software more reliable [20].

4.4. Lifecycle Management and Iterative Development

It involves iterative development of QSDL and incremental improvement in the coverage of the relevance of the entire software lifecycle. Quantum technologies are developing at unprecedented velocity, so this agile mindset is relevant for iterative-incremental development. It allows the inclusion of new research findings, technological advances, and users' feedback in software development. This effective change management strategy shall be adopted to handle changing requirements and ensure that software remains aligned with the project's vision. QSDL extends conventional software development methodologies into the quantum computing sphere; in a hybrid classical and quantum lifecycle framework, such a call is integrated into enabling the development of quantum applications consisting of quantum and classical subsystems. The QSDL adopting iterative practices helps keep quantum software adaptable to new discoveries, aligning continuous work with the project objectives [21]. It is essential because quantum technologies are so complex and still in an infant stage that they require a disciplined but

flexible development process.

4.5. Collaboration and Knowledge Sharing

Developing successful quantum software in itself is a process entailing collaboration from a diversity of stakeholders, including quantum physicists, software engineers, domain experts, and end-users. An environment for such interaction will be provided by the QSDL framework when interdisciplinary teams come together to work on these quantum computing challenges. It thus makes knowledge sharing and continuous learning an integral part of the same process, which helps the team stay in the know with the latest news around the domain and apply the best practices to their projects. This requires combining various kinds of software development life cycles in building robust and efficient quantum hybrid applications. Smooth cooperation of quantum algorithms with classical programs will mean increased data exchange and general improvements in the development process. According to Dwivedi *et al.* (2024), it harmonizes the life cycles of different software artifacts: it assures the interdisciplinary teams work effectively and benefit from both the strength of classical computing and quantum computing. This is a setup that not only enhances problem-solving but also fosters innovation because it integrates diverse expertise and technologies [22].

5. Best Practices for Quantum Software Development

Quantum software development is crucial to harnessing the full power of quantum computing. Classical and agile software development life cycle (SDLC) lessons can provide a solid foundation for developing quantum software. The scope and complexity of the entire development process should be understood to set up an efficient system [23]. This principle is also suitable for quantum software development, where meticulous planning and systematic deployment are the keys to success. The iterative improvement, forming the highlight of both the iterative and spiral models of the frameworks, will no doubt be beneficial in this respect. These models allow for continuous improvement and early detection of problems—an essential factor when dealing with the fast-moving quantum computing area. To meet whatever unique challenges one faces in Quantum Software Development, those must be worked out using flexible, customer-oriented Agile approaches such as Scrum and Kanban. Now, looking back into the details of those points from the principles mentioned above of SDLC, here are some best practices to consider for quantum software development.

5.1. Comprehensive Requirements Analysis

Any good quantum software project requires a complete requirements analysis on which to base its work. It needs to consider the needs of researchers, domain experts, and end-users by eliciting, analyzing, and prioritizing functional and non-functional requirements. In this way, peculiar complications and opportunities offered by quantum computing have been considered timely from the beginning.

Furthermore, great care should be taken to distinguish between quantum and classical solutions since current classical technologies might already satisfy the user's needs. Also, users will opt out when the technical risks of quantum software systems are high and the benefits are unclear.

To illustrate the best practices for comprehensive requirements analysis in quantum software development, consider the case of developing a quantum cryptography system designed to enhance secure communication. The example below demonstrates how to effectively gather, analyze, and prioritize requirements for the project.

1) Stakeholder Engagement:

- **Stakeholders:** All these would involve collaborations with experts in cryptography, those in quantum physics, software engineers, network security analysts, and the end-users themselves.
- **Purpose:** Understand different needs and expectations from various perspectives to ensure every viewpoint is noticed.
- **Activity:** Conduct workshops and interviews for preliminary requirements capture and use case determination.
- **Special for Quantum Software:** This cannot be avoided because the very principles of quantum, the basics in themselves—superposition and entanglement—are already different from classical cryptography.

2) Gathering Functional Requirements:

- **Definition:** Clearly define what functionalities the quantum cryptography system should be able to provide. An example can be generating quantum keys through entanglement with a 256-bit key length at least. Another point may be the secure distribution of keys over a quantum channel with real-time detection of any attempts at eavesdropping. In all this, the unique features of quantum technology are brought to bear in producing much safer protocols than all the classical systems can ever offer.
- **Documentation:** Use clear, measurable criteria to document these requirements.
- **Special for Quantum Software:** Functional requirements must account for quantum-specific processes like quantum key distribution (QKD) protocols, which do not have classical equivalents.

3) Gathering Non-Functional Requirements:

- **Definition:** Define clearly quality features like performance, scalability, security, and maintainability. An example could be that the system grants a BER of less than 1% to guarantee communication security; furthermore, it should enable scaling up to 1000 nodes capable of secure communications with each other. They are essential parameters for general dependability and effectiveness in facing actual application needs when stringent security rules are applied.
- **Documentation:** Create a requirements specification document detailing these attributes.
- **Special for Quantum Software:** Non-functional requirements must include

considerations unique to quantum technology, such as quantum error correction and coherence time.

4) Analyzing and Prioritizing Requirements:

- **Technique:** Use methods such as MoSCoW, AHP, or Evolve to prioritize requirements.
- **Activity:** Facilitate a requirements prioritization session with stakeholders to categorize and prioritize the requirements based on their importance and feasibility.
- **Special for Quantum Software:** Prioritization must consider the current limitations of quantum technology, such as qubit coherence times and the technical challenges of maintaining entanglement over long distances.

5) Defining the Scope and Objectives:

- **Scope:** This is to spell out clearly what the quantum cryptography system will and will not do to ensure stakeholders' expectations about the results are met. Also, the project's quantifiable objectives are on the required level of efficiency in generating and distributing keys securely.
- **Documentation:** Create a project scope statement that includes these details.
- **Special for Quantum Software:** The scope must explicitly address quantum-specific capabilities and constraints, such as the maximum distance for quantum key distribution without significant loss of fidelity.

6) Developing a Project Schedule:

- **Plan:** Develop an appropriate schedule for the project with milestones for requirement verification, design, development, and testing. Utilize any other available project management software to track progress and cover all requirements in time.
- **Documentation:** Develop a Gantt chart or similar project timeline to visualize the schedule.
- **Special for Quantum Software:** Since quantum technology is still largely experimental, the project schedule must accommodate lengthier development sprints and iterative testing phases that provide exhaustive validation and verification.

7) Continuous Validation:

- **Activity:** Regularly review and validation of requirements with stakeholders during the project's life cycle. In addition, it should ensure that the requirements are current and met as the project transits.
- **Documentation:** Maintain a requirements traceability matrix to track the implementation of each requirement.
- **Special for Quantum Software:** Continuous validation is critical in quantum software development to adapt to rapid advancements in quantum technology and to incorporate emerging best practices and standards in quantum cryptography.

The methodologies align projects involving quantum software development with the project-specific needs, the various stakeholders involved, and the

inherent technological limitations. The job: to ensure that all perspectives on the project have been considered critically, well documented, and prioritized for delivery. This in-depth analysis shall mean something in this ever-shifting phase of quantum computing, for which risk mitigation is a must. Through this, high-quality functional software systems will be delivered to exploit the potential of quantum.

5.2. Adopt a Hybrid Approach

The building of quantum computing really raises much of its challenges, which contrast classical computing, especially for challenging applications such as drug discovery [24]. In this respect, growing complexity underlines the pressing need for Quantum Software Engineering as a discipline taking care of the effective and cost-efficient development, operation, and maintenance of quantum software [25]. Due to its unique nature, QSE must be an adaptation of traditional principles of software engineering, including iterative and agile processes, new abstractions suited for quantum paradigms, and quantum-structured programming practices. The discipline urgently requires an adaptation of traditional software engineering techniques for the requirements of quantum programming, even though empirical evidence that directly links agile practices to quantum software development is still lacking [26]. The AGILE methodologies that emphasize active user participation, short iterations, frequent releases, and refactoring may be a promising strategy in facing the multidimensional challenges placed at the core of quantum software development [27].

Agile integration with QSE provides a comprehensive framework that supports iterative, collaborative, and technically demanding characteristics of quantum software projects. As such, Crystal methodologies are suited to teams to iteratively work out complex quantum problems that naturally involve much collaboration and adaptive planning [28]. The Dynamic Systems Development Method (DSDM) emphasizes iterative development and continuous user involvement, which can be adapted to quantum projects by engaging quantum domain experts throughout the development process to refine requirements and solutions [29]. Feature-Driven Development (FDD) focuses on building and designing features, enabling quantum software development to break down complex algorithms into manageable components, facilitating incremental progress and early testing [30]. Lean Software Development aims to enhance efficiency by eliminating waste and concentrating on value delivery, ensuring streamlined processes and resource optimization in quantum software projects [31]. With its flexible planning, iterative progress, and collaborative sprints, Scrum is particularly well-suited for managing quantum software development's dynamic and experimental nature [32]. Finally, Extreme Programming (XP) emphasizes technical excellence and frequent releases in short development cycles. It allows quantum software to maintain high-quality code and rapidly iterate on quantum algorithms and their integration with

classical systems [23]. Collectively, these agile methodologies empower the effective and efficient delivery of quantum solutions by facilitating solutions to the particular challenges and requirements of QSE.

In developing, for example, a Quantum Cryptography System (QCS), agile methodologies integrated into QSE may provide an effective way to deal with the particular challenges of this quantum software project. Every agile methodology has unique and more efficient features in specific contexts. Therefore, integrating different agile approaches within one project is beneficial. The integration increases flexibility, fosters a spirit of collaboration, and enhances technical efficiency, thus improving project management efficiency. This collaborative nature of agile methodologies makes the team feel connected and part of a unified effort. The following will illustrate how integration may be realized in the QCS project using agile methods:

1) Crystal Methodologies:

- **Communication and Team Dynamics:** Communication is the key to success within the QCS project, which involves quantum cryptographers, software engineers, and stakeholders alike. Crystal methodologies encourage frequent meetings for updates on progress and challenges within the teams. In this case, daily stand-ups by the team may be held to keep everybody aligned and obstacles resolved promptly.
- **Adaptive Planning:** The dynamic nature of quantum cryptography requires the team to adapt plans as new quantum algorithms or vulnerabilities are discovered. Crystal methodologies support this by allowing the team to revise project plans regularly based on new insights.

2) Dynamic Systems Development Method (DSDM):

- **Iterative Development:** The development of the QCS will involve iterative cycles where, for example, protocols like QKD are continually improved. DSDM encourages the division of the project into smaller, more manageable iterations. For example, in this case, the team may start with a basic QKD protocol and incrementally enhance its security features.
- **Constant User Involvement:** DSDM emphasizes involving end-users and stakeholders throughout the development process. For QCS, this means regularly consulting with cybersecurity experts and potential users to refine requirements and ensure the system meets practical security needs.

3) Feature-Driven Development (FDD):

- **Feature Building and Design:** The QCS can be divided into features such as key generation, key exchange, and encryption/decryption modules. FDD reinforces this approach to building independent features. For example, the team may first build the key generation module, test it thoroughly, and then proceed to build the key exchange feature.
- **Incremental Progress:** By focusing on individual features, the team can make incremental progress and ensure each component works correctly before integrating them into the overall system. This approach minimizes risks and

facilitates early detection of issues.

4) Lean Software Development:

- **Efficiency and Value Delivery:** Lean principles help streamline the development process by eliminating unnecessary steps and focusing on value delivery. In the QCS project, this means optimizing quantum algorithms to ensure they perform efficiently on available quantum hardware, avoiding redundant computations.
- **Resource Optimization:** Given the limited availability and high cost of quantum computing resources, lean methodologies ensure these resources are used judiciously. For instance, simulations and small-scale tests might be used to validate algorithms before deploying them on actual quantum hardware.

5) Scrum:

- **Flexible Planning and Sprints:** Scrum has an iterative approach that involves breaking down the project into small sprints, each dealing with specific tasks. QCS could perhaps plan a two-week sprint to develop and test the QKD protocol. At the end of every sprint, the progress has to be reviewed by the team to include improvements for the next sprint.
- **Collaborative Environment:** Scrum fosters collaboration through regular sprint planning, daily stand-ups, and sprint reviews. For example, during sprint reviews, the team demonstrates the working features of the QCS to stakeholders and incorporates their feedback into the next sprint cycle.

6) Extreme Programming (XP):

- **Technical Excellence and Frequent Releases:** XP's emphasis on high-quality code and frequent releases ensures that the QCS maintains robustness and security. The team might adopt practices like pair programming and continuous integration to ensure code quality. Frequent releases allow for rapid iteration and integration of user feedback.
- **Rapid Iteration:** The short development cycles of XP enable the team to quickly iterate on quantum algorithms and classical-quantum integrations. For instance, the team might release a new version of the QCS every few weeks, each with improved security features and optimizations based on the latest research.

Such Agile techniques can enable a Quantum Cryptography System to gracefully manage peculiar complexities and the experimental nature of such quantum software projects. A framework considering all aspects of iterative development, continuous enhancement, and collaborative problem-solving in developing a robust, powerful, and secure quantum solution.

5.3. The Right Quantum Programming Language

The choice of an adequate quantum programming language is extremely important in quantum software development. In fact, it decides the efficiency, scalability, and success of a project. While in classical software development, the choice among different programming languages often depends on the experience of the

developer or specific restrictions imposed by the project, quantum software requires much finer analysis because of emergent and fast-growing aspects of quantum computing. Chapter reviews the technical problems presenting practical considerations and examples to enable such a choice.

5.3.1. Understand the Project Requirements

Before choosing a quantum programming language, it is essential to thoroughly understand the specific requirements of the project. These include:

- **Project Objectives:** Define whether the goal is to develop quantum algorithms (e.g., Shor's algorithm for factoring), simulate quantum phenomena, or optimize specific problems (e.g., the Traveling Salesman Problem using quantum annealing).
- **Quantum Hardware:** Identify the target quantum hardware platform, such as IBM's superconducting qubits (IBM Q), Google's Sycamore, or D-Wave's quantum annealing processors. Each platform has associated programming languages optimized for its architecture.
- **Classical Integration:** Consider the classical computing environments and programming languages that will interface with the quantum code. Quantum languages like Qiskit and Cirq are tightly integrated with Python, while Q# integrates with .NET languages such as C#.

5.3.2. Evaluate Language Features and Libraries

Quantum programming in each language consists of its own set of tools and libraries, which can fairly influence the development process's efficiency and result in general. Qiskit, for example, has an extensive list of quantum information science-related libraries: some libraries perform quantum circuit simulation, others are for circuit optimization to run on quantum hardware, even quantum chemistry applications. The Ignis submodule focuses on error correction and noise mitigation within Qiskit—a highly relevant activity in the Noisy Intermediate-Scale Quantum era, where quantum noise management and reduction are at the top of the list of interests. In contrast, Cirq has low-level features relative to deep gate operations in the sense of creating quantum circuits. Thus, it can actually be a particularly appropriate library for a project to carry out complicated quantum state manipulation or-say-the development of new quantum algorithms right on the hardware level.

The comparison in Table 1 is not exhaustive but rather an illustrative case aimed at giving a representative example of critical parameters that need to be considered when deciding which tool is best suited for any quantum software development project. This comparison helps developers make informed decisions based on project requirements, ensuring efficient and scalable quantum software development.

The above comparison in the table shows that quantum programming languages differ in their characteristics concerning certain features of quantum computing. Besides technical issues, a developer should consider community support,

Table 1. Comparative analysis of quantum programming languages, highlighting key features and target devices for informed selection.

Feature	Qiskit [33]	Cirq [28]	Quipper [34]	Silq [35]	Q# [36]
Developer	IBM	Google	Quipper	ETH Zurich	Microsoft
Primary Language	Python	Python	Haskell	Stand alone	Stand-alone
Abstraction Level	Low to Medium	Low to Medium	High	High	high
Target Quantum Devices	IBM Q devices, simulators	Google's NISQ devices, simulators	Hardware-agnostic	Hardware-agnostic, Quantum Random Access Machine (QRAM)	Quantum simulators, hardware.
Syntax	Pythonic, with quantum-specific APIs	Pythonic, with quantum-specific APIs	Functional (Haskell-based)	High-level, unique and self-contained	Domain-specific quantum language
Integration with Classical	Full integration with Python	Full integration with Python	Full integration with Haskell	Limited, focus on quantum code	Full integration with .NET ecosystem
Quantum Circuit Representation	QASM (Quantum Assembly)	Cirq's internal circuit representation	Functional composition, circuit libraries	Abstract quantum operations	No explicit circuit representation
Community & Support	Extensive documentation, large community	Growing community, active development	Limited, academic community	Emerging, small community	.NET languages & Azure Quantum katas
Learning Curve	Moderate (requires understanding of quantum mechanics)	Moderate (focus on NISQ applications)	Steep (requires understanding of functional programming)	Moderate (simplified syntax for ease of use)	Steep (requires understanding of quantum mechanics and integration within the .NET ecosystem).
Unique Features	Versatile, supports hybrid quantum-classical computing	Optimized for NISQ, built-in circuit optimization tools	High-level functional programming abstractions	Automatic handling of uncomputation, simplified quantum programming	Strong type safety, integration with classical computation

the ability to work with a classical system, and, what is important, the learning curve of a certain language. Although this table summarizes several major programming languages, one must conduct an in-depth analysis based on the project requirements. Hardware compatibility, scalability needs, and availability of advanced libraries must be weighed before making any decisions. As long as one relates all these to the project's goals, then he or she is better placed to decide on the choices that enhance his or her quantum software development productivity and effectiveness.

5.3.3. Assess Performance and Scalability

Performance is a critical consideration in quantum software development, particularly because current quantum hardware is limited by the number of qubits and gate fidelities. Therefore, the selected quantum programming language must support efficient simulation environments, as extensive testing on simulators is often necessary before deploying code on actual quantum processors.

The Aer simulator for Qiskit is performance-optimized and can run gigantic circuits, which may contain thousands of qubits and gates. It further adds to its support for modeling noise, which is necessary for building error correction code and testing realistic behavior in quantum algorithms [37]. Similar is Cirq, which has not lagged in providing hard-core simulation capabilities but allows one to simulate quantum circuits under noise models representative of Google's quantum processors [38].

Scalability is the most important aspect to consider here. As the size of quantum circuits increases, handling quantum resources effectively becomes highly challenging [39]. Cirq does an excellent job in the former case by offering functionalities for circuit depth and gate-count optimization, both crucial for running big-size quantum algorithms on hardware with a short coherence time.

5.3.4. Leverage Community and Support

The strength of a quantum programming language in terms of community robustness and accessibility to an infrastructural support system forms the threshold of effectiveness. It is not just a help in continuous development but is also beneficial in finding a way around the obstacles that come in one's way during implementation, and the key factors mentioned below must be noted to reason with the community and support ecosystem of a quantum programming language [40]:

- **Community Size and Activity:** A large and active community often means better support, more shared resources, and faster resolution of issues. Qiskit, for instance, has an extensive community with numerous tutorials, forums, and GitHub repositories available for developers.
- **Documentation and Tutorials:** Ensure the language has comprehensive documentation and up-to-date tutorials. This is especially crucial for complex languages like Quipper, which may offer advanced features but require a steeper learning curve.

5.3.5. Prototype and Benchmark

It is, therefore, important to prototype and benchmark with a quantum programming language that fits the project at hand. Such an approach provides 'top-down' feedback that may be lost in a more formal theoretical review. Second, benchmarks are necessary to estimate your code's runtime and resource consumption for different languages and simulators [41]. For instance, compare the runtime for some implementation of Grover's algorithm between Qiskit and Cirq. This, therefore, indicates that the most efficient and effective language is selected for your given application based on the data.

5.4. Building an Experience

Competence in QSE is to be flexible and predictive since the software industry grows very fast and new development technologies emerge almost daily. The developer shall find ways to participate in the emergent landscape of quantum computing by keeping up with developments and adopting cutting-edge tools and methods. This, therefore, becomes crucial in the face of an acknowledged deficiency in software engineering methodologies among quantum computer scientists [42]. The addressing of such a shortage supports better quality and efficiency of quantum software, whereby resilient and scalable quantum software solutions will be effective in harnessing the distinctive potentials of quantum systems.

Moreover, it feels right to expedite methods development in QSE in a focused fashion, given the development of quantum programming languages, rather than waiting for them to 'stabilize' [43]. This will also help mature this field quicker and solve significant challenges of comprehending sophisticated quantum theories, creating sophisticated hardware and software infrastructures, and low practical applications. This set of ideal methodologies through which software development should be done—testing, modularity, ample documentation—allows the developer to provide the sound framework of the quantum software projects. Therefore, it allows more effective and innovative quantum solutions that will make it very easy for future quantum software engineers to navigate and upgrade in this fast-developing domain of quantum computing.

6. Future Directions in Quantum Software Development

Quantum Software Engineering is poised for significant advancements that will enhance its efficiency, reliability, and applicability across various domains. Several key areas will shape the future of this field.

6.1. High-Level Abstractions and Frameworks

A promising research direction is the design of advanced abstractions and conceptual frameworks. These tools are intended to make quantum programming more accessible for developers who are not highly aware of quantum mechanics and physics [44]. This approach will enable the development of quantum applications and will, therefore, bring about even more participation in the development of quantum software.

6.2. Hybrid Architectures

Integrating quantum and classical systems will become increasingly critical as quantum computing evolves. The future of quantum software will likely employ more sophisticated hybrid architectures based on the current efforts to integrate quantum and classical computing resources [45]. These include a combination of classical and quantum processors in such a way that the strengths of each are optimized within a strategy to access more efficient problem-solving processes.

6.3. Advanced Error Correction and Mitigation

Error correction and mitigation are crucial for the practical deployment of quantum applications. Current work is more based on reduced errors in quantum systems that are available today, and in the future, more robust and scalable approaches will be developed [46]. Dependability, thus strongly enhanced besides the accuracy of quantum computations, will enable making quantum applications more stable and scalable.

6.4. Agile and Iterative Development Methodologies

Adoption of the agile and iterative development methodologies, already proving very beneficial toward addressing the unique challenges of quantum software development, is going to increase [27]. Furthermore, future QSE practices will experience such methodologies to be further refined and adapted in order to meet the specific needs of quantum computing in a way that allows more flexible and responsive development cycles to be capable enough to keep pace with rapid technological advancements.

6.5. Integration with Artificial Intelligence and Machine Learning

The synergy between quantum computing and artificial intelligence (AI) is expected to be a major focus area. It has been highly probably that considerable improvement in the terms of computational power from quantum-amplified machine learning algorithms will revolutionize parts of health, finance, and logistics domains [47]. Thus, new ways to innovate, stemming from the building-on advantage accruing from AI and quantum computing, will be realized.

6.6. Standardization and Industry Collaboration

Quantum technologies are developing at an ever-increasing pace. In addition, with it comes a growing need for standardization and collaboration both in the field and across industry sectors. Setting common standards and practices will help interoperability and is a precondition to facilitate wide adoption of quantum applications within existing technological systems [48]. To that effect, standards will be set through collaboration between academia, industry, and governments to drive quantum technology into more widespread use.

6.7. Education and Talent Development

The development of the quantum computing industry would include growing a skilled workforce with specific expertise in quantum technologies. These efforts are directed toward enhancing present education curricula, from quantum mechanics and information theory to practical laboratory work. It is very desirable to come together with academia and industry to create appropriate curricula, courses, workshops, and other internship opportunities to help knit academic knowledge together with real-world applications in the quantum technology arena. It also recognizes the need to develop programs that will provide the

students and the existing workforce with the specific technical skills required, especially in quantum areas such as quantum sensing, networking, and quantum computing. This belief will ensure the survival of the labor supply, particularly in emerging fields such as quantum engineering, for the growing needs of the industry [49].

7. Conclusions

The potential of quantum computing has been realized by addressing unique challenges presented both at the quantum hardware level and at the integration level of software. This paper outlines several essential best practices based on the rich background of classical software engineering, including considerations in extending this to the quantum domain. The practices discussed in this paper are intended to extend near-term solutions for scaleable, reliable, efficient quantum software to lifecycle management, error mitigation, and integration of classical and quantum systems. One of the principal contributions is the introduction of the structured framework QSDL, which guides quantum software development. Iterative development, advanced testing techniques, and solid error correction strategies ensure that quantum applications work in today's noisy quantum environments. Implementing a classical resource with the quantum computer, mitigating errors, and testing all in one address current quantum hardware limitations and lead to immediate, practical, implementable solutions.

Additionally, the insights on knowledge requirements and interdisciplinary collaboration highlight the importance of developing a skilled workforce capable of handling the complexities of QSE. As the field continues to evolve, these best practices guide current efforts and lay a strong foundation for future advancements. By adopting these structured approaches, the field can accelerate the transition from experimental research to practical quantum computing applications, driving innovation and ensuring that the next generation of quantum software is robust, scalable, and ready for industrial deployment.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Harrow, A.W. and Montanaro, A. (2017) Quantum Computational Supremacy. *Nature*, **549**, 203-209. https://doi.org/10.1038/nature23458

[2] Uehara, G.S., Spanias, A. and Clark, W. (2021) Quantum Information Processing Algorithms with Emphasis on Machine Learning. 2021 12*th International Conference on Information*, *Intelligence*, *Systems & Applications* (*IISA*), Chania Crete, 12-14 July 2021, 1-11. https://doi.org/10.1109/iisa52424.2021.9555570

[3] Pérez-Castillo, R., Serrano, M.A. and Piattini, M. (2021) Software Modernization to Embrace Quantum Technology. *Advances in Engineering Software*, **151**, Article ID: 102933. https://doi.org/10.1016/j.advengsoft.2020.102933

[4] Choe, S. and Perkowski, M. (2022) Continuous Variable Quantum MNIST Classifiers—

Classical-Quantum Hybrid Quantum Neural Networks. *Journal of Quantum Information Science*, **12**, 37-51. https://doi.org/10.4236/jqis.2022.122005

[5] Rüde, U., Willcox, K., McInnes, L.C. and Sterck, H.D. (2018) Research and Education in Computational Science and Engineering. *SIAM Review*, **60**, 707-754. https://doi.org/10.1137/16m1096840

[6] Duan, H. (2022) The Principles, Algorithms and State-of-Art Applications of Quantum Computing. *Journal of Physics*: *Conference Series*, **2386**, Article ID: 012025. https://doi.org/10.1088/1742-6596/2386/1/012025

[7] Avizienis, A., Laprie, J., Randell, B. and Landwehr, C. (2004) Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, **1**, 11-33. https://doi.org/10.1109/tdsc.2004.2

[8] García de la Barrera, A., García-Rodríguez de Guzmán, I., Polo, M. and Piattini, M. (2021) Quantum Software Testing: State of the Art. *Journal of Software*: *Evolution and Process*, **35**, e2419. https://doi.org/10.1002/smr.2419

[9] Jozsa, R. (1997) Complexity and Algorithms in Quantum Computation. *IEE Colloquium on Quantum Computing*: *Theory*, *Applications and Implications*, London. https://doi.org/10.1049/ic:19970792

[10] Britt, K.A., Mohiyaddin, F.A. and Humble, T.S. (2017) Quantum Accelerators for High-Performance Computing Systems. 2017 *IEEE International Conference on Rebooting Computing* (*ICRC*), Washington DC, 8-9 November 2017, 1-7. https://doi.org/10.1109/icrc.2017.8123664

[11] Van Vliet, H., Van Vliet, H. and Van Vliet, J.C. (2008) Software Engineering: Principles and Practice (Vol. 13). John Wiley & Sons. https://gnindia.dronacharya.info/CSE/6thSem/Downloads/Software-Engineering/Books/Software-Engineering-text-book-5.pdf

[12] Carbonelli, C., Felderer, M., Jung, M., Lobe, E., Lochau, M., Luber, S., *et al.* (2024) Challenges for Quantum Software Engineering: An Industrial Application Scenario Perspective. In: *Quantum Software*, Springer Nature Switzerland, 311-335. https://doi.org/10.1007/978-3-031-64136-7_12

[13] Camps, D., Van Beeumen, R. and Yang, C. (2020) Quantum Fourier Transform Revisited. *Numerical Linear Algebra with Applications*, **28**, e2331. https://doi.org/10.1002/nla.2331

[14] Asfaw, A., Blais, A., Brown, K.R., Candelaria, J., Cantwell, C., Carr, L.D., *et al.* (2022) Building a Quantum Engineering Undergraduate Program. *IEEE Transactions on Education*, **65**, 220-242. https://doi.org/10.1109/te.2022.3144943

[15] Williams, C.P. (2011) Quantum Gates. In: Williams, C.P., Ed., *Explorations in Quantum Computing*, Springer, 51-122. https://doi.org/10.1007/978-1-84628-887-6_2

[16] Hughes, C., Finke, D., German, D., Merzbacher, C., Vora, P.M. and Lewandowski, H.J. (2022) Assessing the Needs of the Quantum Industry. *IEEE Transactions on Education*, **65**, 592-601. https://doi.org/10.1109/te.2022.3153841

[17] Khan, A.A., Ahmad, A., Waseem, M., Liang, P., Fahmideh, M., Mikkonen, T., *et al.* (2023) Software Architecture for Quantum Computing Systems—A Systematic Review. *Journal of Systems and Software*, **201**, Article 111682. https://doi.org/10.1016/j.jss.2023.111682

[18] Khan, A.A., Taibi, D. and Perrault, C.M. (2024) Advancing Quantum Software Engineering: A Vision of Hybrid Full-Stack Iterative Model. https://www.researchgate.net/profile/Arif-Khan-23/publication/378077106_Advancing_Quantum_Software_Engineering_A_Vision_of_Hybrid_Full-

Stack_Iterative_Model/links/65c5f6081bed776ae3395b01/Advancing-Quantum-Software-Engineering-A-Vision-of-Hybrid-Full-Stack-Iterative-Model.pdf

[19] Gulania, S., Gray, S., Peng, B., Govind, N. and Alexeev, Y. (2024) Quantum Error Mitigation and Correction Mediated by Yang-Baxter Equation and Artificial Neural Network. Bulletin of the American Physical Society. https://meetings.aps.org/Meeting/MAR24/Session/OD01.123

[20] Ramalho, N.C.L., de Souza, H.A. and Chaim, M.L. (2024) Testing and Debugging Quantum Programs: The Road to 2030.

[21] Weder, B., Barzen, J., Leymann, F. and Vietz, D. (2022) Quantum Software Development Lifecycle. In: Serrano, M.A., Pérez-Castillo, R. and Piattini, M., Eds., *Quantum Software Engineering*, Springer International Publishing, 61-83. https://doi.org/10.1007/978-3-031-05324-5_4

[22] Dwivedi, K., Haghparast, M. and Mikkonen, T. (2024) Quantum Software Engineering and Quantum Software Development Lifecycle: A Survey. *Cluster Computing*, **27**, 7127-7145. https://doi.org/10.1007/s10586-024-04362-1

[23] Leau, Y.B., Loo, W.K., Tham, W.Y. and Tan, S.F. (2012) Software Development Life Cycle AGILE vs Traditional Approaches. *International Conference on Information and Network Technology*, **37**, 162-167.

[24] Ali, S., Yue, T. and Abreu, R. (2022) When Software Engineering Meets Quantum Computing. *Communications of the ACM*, **65**, 84-88. https://doi.org/10.1145/3512340

[25] Piattini, M., Peterssen, G. and Pérez-Castillo, R. (2020) Quantum Computing: A New Software Engineering Golden Age. *ACM SIGSOFT Software Engineering Notes*, **45**, 12-14. https://doi.org/10.1145/3402127.3402131

[26] Piattini, M., Serrano, M., Perez-Castillo, R., Petersen, G. and Hevia, J.L. (2021) Toward a Quantum Software Engineering. *IT Professional*, **23**, 62-66. https://doi.org/10.1109/mitp.2020.3019522

[27] Khan, A.A., Akbar, M.A., Ahmad, A., Fahmideh, M., Shameem, M., Lahtinen, V., *et al.* (2023) Agile Practices for Quantum Software Development: Practitioners' Perspectives. 2023 *IEEE International Conference on Quantum Software* (*QSW*), Chicago, 2-8 July 2023, 9-20. https://doi.org/10.1109/qsw59989.2023.00012

[28] Mangudo, P., Arroqui, M., Marcos, C. and Machado, C.F. (2012) Rescue of a Whole-Farm System: Crystal Clear in Action. *International Journal of Agile and Extreme Software Development*, **1**, 6-22. https://doi.org/10.1504/ijaesd.2012.048306

[29] Chapram, S.B. and Solutions, Z.I. (2018) An Appraisal of Agile DSDM Approach. *International Journal of Advanced Studies in Computers, Science and Engineering*, **7**, 1-3.

[30] Mahdavi-Hezave, R. and Ramsin, R. (2015) FDMD: Feature-Driven Methodology Development. *Proceedings of the* 10*th International Conference on Evaluation of Novel Approaches to Software Engineering*, Barcelona, 29-30 April 2015, 229-237. https://doi.org/10.5220/0005384202290237

[31] Mehta, M., Anderson, D. and Raffo, D. (2008) Providing Value to Customers in Software Development through Lean Principles. *Software Process: Improvement and Practice*, **13**, 101-109. https://doi.org/10.1002/spip.367

[32] Hema, V., Thota, S., Naresh Kumar, S., Padmaja, C., Rama Krishna, C.B. and Mahender, K. (2020) Scrum: An Effective Software Development Agile Tool. *IOP Conference Series: Materials Science and Engineering*, **981**, Article ID: 022060. https://doi.org/10.1088/1757-899x/981/2/022060

[33] Kanazawa, N., Egger, D.J., Ben-Haim, Y., Zhang, H., Shanks, W.E., Aleksandrowicz, G., *et al.* (2023) Qiskit Experiments: A Python Package to Characterize and Calibrate Quantum Computers. *Journal of Open Source Software*, **8**, Article No. 5329. https://doi.org/10.21105/joss.05329

[34] Green, A.S., Lumsdaine, P.L., Ross, N.J., Selinger, P. and Valiron, B. (2013) An Introduction to Quantum Programming in Quipper. In: Dueck, G.W. and Miller, D.M., Eds., *Reversible Computation*, Springer, 110-124. https://doi.org/10.1007/978-3-642-38986-3_10

[35] Bichsel, B., Baader, M., Gehr, T. and Vechev, M. (2020) Silq: A High-Level Quantum Language with Safe Uncomputation and Intuitive Semantics. *Proceedings of the* 41*st ACM SIGPLAN Conference on Programming Language Design and Implementation*, London, 15-20 June 2020, 286-300. https://doi.org/10.1145/3385412.3386007

[36] Svore, K., Geller, A., Troyer, M., Azariah, J., Granade, C., Heim, B., *et al.* (2018) Q# Enabling Scalable Quantum Computing and Development with a High-Level DSL. *Proceedings of the Real World Domain Specific Languages Workshop* 2018, 24 February 2018, 1-10. https://doi.org/10.1145/3183895.3183901

[37] Li, Y., Hsieh, C., Li, Y. and Li, J.C. (2023) Diagnosis of Quantum Circuits in the NISQ Era. 2023 *IEEE* 41*st VLSI Test Symposium* (*VTS*), San Diego, 24-26 April 2023, 1-7. https://doi.org/10.1109/vts56346.2023.10140030

[38] Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J.C., Barends, R., *et al.* (2020) Hartree-Fock on a Superconducting Qubit Quantum Computer. *Science*, **369**, 1084-1089. https://doi.org/10.1126/science.abb9811

[39] Mahmud, N. and El-Araby, E. (2018) Towards Higher Scalability of Quantum Hardware Emulation Using Efficient Resource Scheduling. 2018 *IEEE International Conference on Rebooting Computing* (*ICRC*), McLean, 7-9 November 2018, 1-10. https://doi.org/10.1109/icrc.2018.8638610

[40] De Stefano, M., Pecorelli, F., Di Nucci, D., Palomba, F. and De Lucia, A. (2022) Software Engineering for Quantum Programming: How Far Are We? *Journal of Systems and Software*, **190**, Article ID: 111326. https://doi.org/10.1016/j.jss.2022.111326

[41] Kordzanganeh, M., Buchberger, M., Kyriacou, B., Povolotskii, M., Fischer, W., Kurkin, A., *et al.* (2023) Benchmarking Simulated and Physical Quantum Processing Units Using Quantum and Hybrid Algorithms. *Advanced Quantum Technologies*, **6**, Article ID: 2300043. https://doi.org/10.1002/qute.202300043

[42] Haghparast, M., Mikkonen, T., Nurminen, J.K. and Stirbu, V. (2023) Quantum Software Engineering Challenges from Developers' Perspective: Mapping Research Challenges to the Proposed Workflow Model. 2023 *IEEE International Conference on Quantum Computing and Engineering* (*QCE*), Bellevue, 17-22 September 2023, 173-176. https://doi.org/10.1109/qce57702.2023.10204

[43] Spoletini, P. (2023) Towards Quantum Requirements Engineering. 2023 *IEEE* 31*st International Requirements Engineering Conference Workshops* (*REW*), Hannover, 4-5 September 2023, 371-374. https://doi.org/10.1109/rew57809.2023.00072

[44] Cobb, A., Schneider, J. and Lee, K. (2022) Towards Higher-Level Abstractions for Quantum Computing. *Australasian Computer Science Week* 2022, Brisbane, 14-18 February 2022, 115-124. https://doi.org/10.1145/3511616.3513106

[45] Stirbu, V. and Mikkonen, T. (2023) Software Architecture Challenges in Integrating Hybrid Classical-Quantum Systems. 2023 *IEEE International Conference on Quantum Computing and Engineering* (*QCE*), Vol. 2, 203-204. https://doi.org/10.1109/qce57702.2023.10212

[46] Gill, S.S., Kumar, A., Singh, H., Singh, M., Kaur, K., Usman, M., *et al.* (2021)

Quantum Computing: A Taxonomy, Systematic Review and Future Directions. *Software: Practice and Experience*, **52**, 66-114. https://doi.org/10.1002/spe.3039

[47] Gill, S.S., Xu, M., Ottaviani, C., Patros, P., Bahsoon, R., Shaghaghi, A., *et al.* (2022) AI for Next Generation Computing: Emerging Trends and Future Directions. *Internet of Things*, **19**, Article ID: 100514. https://doi.org/10.1016/j.iot.2022.100514

[48] Jenet, A. (2020) Standards4Quantum: Making Quantum Technology Ready for Industry. Putting Science into Standards. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.3888296

[49] Fox, M.F.J., Zwickl, B.M. and Lewandowski, H.J. (2020) Preparing for the Quantum Revolution: What Is the Role of Higher Education? *Physical Review Physics Education Research*, **16**, Article ID: 020131. https://doi.org/10.1103/physrevphyseducres.16.020131