Cluster Computing (2024) 27:7127–7145
https://doi.org/10.1007/s10586-024-04362-1

Quantum software engineering and quantum software development lifecycle: a survey

Kanishk Dwivedi[1] · Majid Haghparast[2] · Tommi Mikkonen[2]

Received: 29 November 2023 / Revised: 16 January 2024 / Accepted: 10 February 2024 / Published online: 25 March 2024

Abstract
Quantum software engineering is advancing in the domain of quantum computing research and application, yet the documentation is scattered. The slow transition from Von-Neumann based computation systems to quantum systems, and conserving the fundamental computing principles in software development and software engineering helps in enrichment of quantum software development. The evolution of quantum computing over the past years shows a shift in the domain of classical computation to quantum computation in the years to come. Future applications such as, quantum AI and quantum machine learning will benefit from quantum software engineering. This survey collects and explores the various documentations in the domain of quantum systems and quantum software engineering. The survey provides an in-depth exploration of quantum programming languages, which is combined with explanations of quantum computing's fundamentals. The review also goes in-depth about quantum software engineering and quantum software life cycle development, outlining the quantum software reuse methodology that is introduced in the quantum software lifecycle development domain.

Keywords Quantum software engineering · Quantum programming languages · Quantum life-cycle development · Quantum hybrid systems · Quantum software analysis · Quantum software development

1 Introduction

A classical computer that exists today follows the Von-Neumann computation model, also defined as Von-Neumann Computers (VMC). Although VMC models have advanced recently, computing has undergone another revolution. In recent years, quantum computing has quickly risen to the forefront of computing, achieving breakthroughs in both theory and applications. Quantum computing systems are still under development; hence, full access is not yet available [1, 2]. Since most quantum computing systems combines VMC and quantum computing, they are regarded as hybrid models. The recent research subject is quantum software engineering, which focuses on the ideas, guidelines, and procedures for setting up, advancing, and preserving applications based on quantum computing.

By methodically implementing quantum software engineering in all the developmental phases in accordance with the initial needs of the quantum software life cycle, the emphasis on reusing quantum software is succinctly described. The VMC principles, which are the documentation of specific development phases that software shall abide by, are the only ones that were taken into consideration when designing the *software development cycle* [2, 3]. However, the life cycles at this time only apply to VMC models and quantum software and do not take into account the difficulties in integrating a hybrid model. For the best possible data transfer between the VMC and quantum software, it is also necessary to address the orchestration of VMC and quantum computers. [4–6]

The following are examples of current taxonomy-related challenges in quantum computing [7] as demonstrated in Fig. 1:

✉ Majid Haghparast
majid.m.haghparast@jyu.fi

[1] Fachbereich Informatik, MIN-Fakultät, Universität Hamburg, 22527 Hamburg, Germany

[2] Faculty of Information Technology, University of Jyväskylä, 40014 Jyvaskyla, Finland

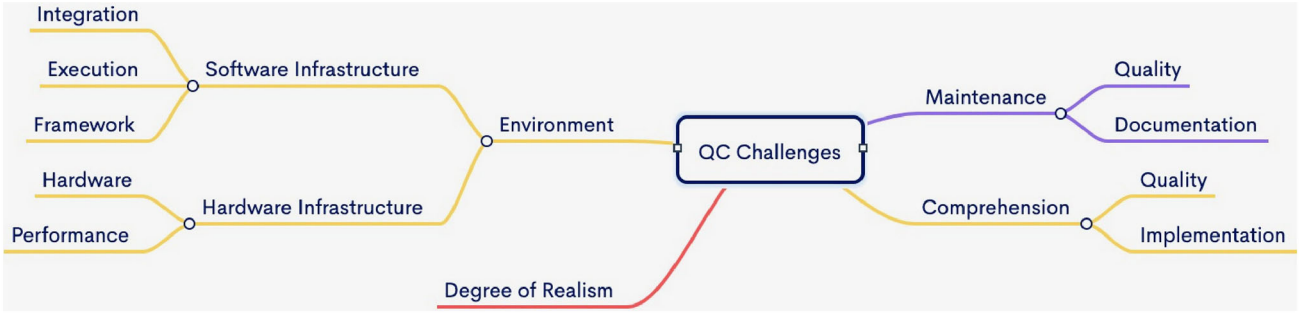

Fig. 1 Graphical representation of quantum computing challenges [7]

- Environment: The environment pivots a vital role with respect to quantum behavior, both with respect to software and hardware taxonomy. This can further be divided into Software infrastructure, Framework, Integration, and Execution. Software infrastructure challenges the framework that can be reused in the execution environment. Framework are mostly created for the *API protocols*. The rigorous changes in the API protocols can make it difficult for the developers to correlate the environments. It is also difficult to understand the theoretical and practical implications of quantum study due to the lack of standardization for the same. **Integration** addresses the challenges to integrate quantum systems with other technologies, such as blockchain or AI, and bringing down the computational costs with regards to QC. **Execution** struggles with the set-up for environments and simulators for interaction between VMC and quantum systems conflicting with the same.
- Hardware infrastructure: The challenges related to it require more specialized tools for the hardware, which at this time is going through frequent changes. The noise in these quantum systems also defines a degree of expertise to tune them for accurate results. **Performance** is another constraint due to the limitation of resources in quantum systems in comparison to VMC models. The greediness of computation with fewer resources conflicts with the same.
- Comprehension: The macro-category consists of the challenges faced by the developers. It emphasizes documentation to the inadequacy of theoretical grounds.
  - Documentation: The most important thing to maintain the systems is documentation. Therefore, the documentation comprehensibility and the quality of the document shall be maintained and must be understandable to future developers to maintain the documentation.
  - Coding: Inadequate **Integrated development environment (IDE)** and compilation challenges fall into this category. Different languages require different IDEs, making it challenging for developers to understand different quantum computation languages for the same task.
  - Code Quality: Code quality poses a challenge as well, as we aim to follow the re-usability and understanding of the code. The main components of code quality, are addressed as debugging, testability, and readability. For debugging, it was found to be hard to understand the error messages for quantum Software development kits (SDKs). Also includes that it was stressful to test the quantum circuits once they are sent to the quantum computer in real-time. Since readable code remains a persisting challenge.

This survey introduces new techniques for evolving quantum software engineering and quantum computing languages. Section 2 digs into the fundamentals of quantum computing i.e., qubits and quantum computing architecture. A brief overview of the quantum software engineering and technologies that currently exist in the market is provided in section 3, which also focuses on the various quantum programming languages. We delve into the understanding of the quantum software life-cycle in great detail in section 4, and we include all of the work that has been completed in the area of the quantum software lifecycle. The future of quantum software engineering and the quantum software lifecycle were also discussed, along with a new paradigm of quantum software reuse, quantum circuit reuse, and quantum state reuse that offers more sustainable solutions. The case studies in quantum software engineering are discussed in section 5. The future of quantum software engineering and research gaps are discussed in section 6. Conclusions are presented in section 7.

2 Background

Quantum computing is built on the principles of quantum mechanics, where the three main components are **Superposition, Entanglement and Interference** [8]:

- Superposition: "It states that, much like waves in classical physics, any two (or more) quantum states can be added together ("superposed"), and the result will be another valid quantum state; and conversely, that every quantum state can be represented as a sum of two or more other distinct states."
- Entanglement: "A physical phenomenon that occurs when a group of particles is generated, interact, or share spatial proximity in a way such that the quantum state of each particle of the group cannot be described independently of the state of the others, including when the particles are separated by a large distance."
- Interference: "This phenomenon was already widely known in the context of waves, but it came as a shocking relation that matter sometimes also behaves like a wave in addition to its particle behavior and therefore also exhibits interference effects. Because the matter-wave can spread in space, it can destructively and constructively interfere at certain positions."

2.1 Qubit and quantum gate

A qubit is a quantum mechanical analog that is similar to a bit. With classical computing, the information is coded in a bit, where every bit has a value of zero or one. In quantum computing, the information is coded in qubits. A qubit is a two-level quantum system where the two basis qubit states are usually written as $|0\rangle 0$ and $|1\rangle 1$. A qubit is in a state $|0\rangle, |1\rangle$ or (unlike a classical bit) in a linear combination of both states. The states of $|0\rangle 0$ and $|1\rangle 1$ can be represented in the matrix state as Eq. 1:

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

When the qubit harnesses the probability of being in $|0\rangle$ or $|1\rangle$, this makes quantum computers feasible and optimized and helps in developing less complexity.

The qubit probability is calculated by the quantum gates. The quantum gates are the matrices that have different operations for particular qubits. Most common gates are defined as the *Pauli X* (Eq. 2), *Pauli Y* (Eq. 3), *Phase Shift gate (Pauli Z)* (Eq. 4), *and Hadamard gate* (Eq. 5) [3].

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

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

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

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

These matrices formulate the mathematics of algorithms into quantum circuits, which help in measuring the probability of the states and predicting the behavior of the algorithm. This makes the foundation of quantum mechanics also known as the "quantum state basis". For a qubit $|x\rangle$ to exist in the state basis, follows the linear combination to be represented by $|x\rangle = \alpha|0\rangle + \beta|1\rangle$, as α and β represents the complex numbers with which the normalization state shall be equal to 1 that is, $|\alpha|^2 + |\beta|^2 = 1$. It can also be referred to as the superposition of two basis states.

2.2 Quantum architecture

The architecture of the quantum computer is provided in Fig. 2; there are various quantum circuits coded in languages such as Q# and Python-based frameworks such as *Qiskit* and *tensorflow 2.0*. These are compiled by the *Quantum Error Correction* compiler and sends the data to the quantum compiler and lastly to the quantum processor that decodes and computes the likelihood of maximum correct answers [9].

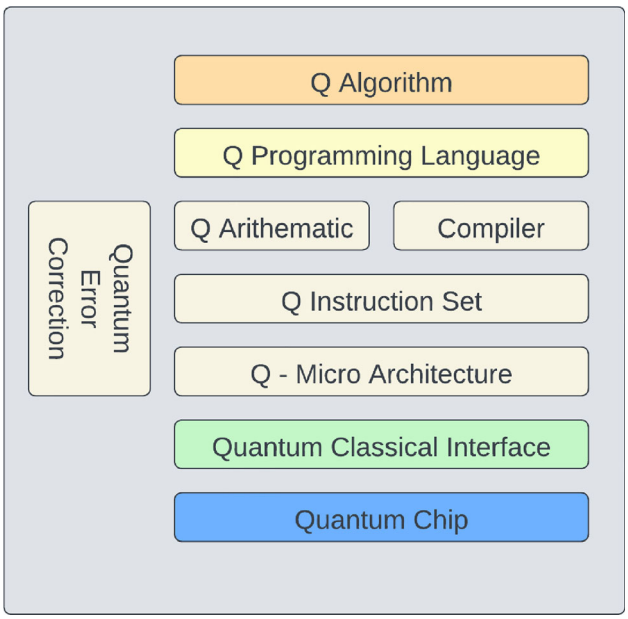

Fig. 2 Quantum computer architecture [10]

The Fig. 2, gives us a general understanding of the architecture and the architectural layers that exist for quantum software. The whole general architecture is designed with respect to the classical VMC software architecture, but extending the architecture gives us more details for an existing system for the quantum software architecture i.e., **Classical-Quantum Hybrid Software**. With a top-down approach, the high-level quantum computing language is to be connected with the network host to access the **quantum computing services (QaaS)** or **quantum processing unit (QPU)** shown in Fig. 3, which converts the language to the desired quantum circuits making it possible to pass through the quantum assembly language. The next step is where the Quantum-classical interface decodes the quantum logic gates and sends them to the hardware. This introduces to the **Classical-Quantum Hybrid software (CQHs)** which is discussed as Software as a Service (SaaS) [11].

3 Related works

Quantum researchers are developing novel approaches for interpreting and integrating software engineering with current VMC models, which are progressing in the field. In the past, models based on server requests were used in this industry. As a result, user-friendly quantum computing programming languages on the host languages of Python and C# were created and incorporated into existing programming languages [2]. Web APIs developed were facilitating the development of new web-based quantum applications for machine learning and AI [6, 12].

3.1 Quantum software engineering

In today's development of quantum computing, it is majorly owned by the big tech industries promoting their developed **quantum software development kits (QSDKs)** and programming languages can be shown in Fig. 4 [13].

The Fig. 4 can be further elaborated concerning every technological partner that can be associated as follows:

- DWave: After designing *adiabatic quantum computers*, it explores the performance in optimization and machine learning-based applications, with their programming language called as **QMASM** that uses decoupling of **QUBO** optimization problems [15, 16].
- Google: They designed what is called *Quantum AI*, while designing algorithms for near-term quantum computers. Most of the Google-based quantum applications can be developed using **Cirq** or **Tensorflow Quantum** [15, 17].
- Honeywell: uses the Trapped-ion technology, providing dynamic qubits which rearranges for better quantum algorithms and applies them in various industries as use cases. [7, 15].
- IBM: provides a cloud based **Quantum Computing as a Service (QCaaS)**. It runs the **QISKIT**, an open-QMASM-based quantum computing programming language, and Qiskit Terra as a platform providing service [15, 16].
- Xanadu: develops hardware based on photon technology with pennylane full-stack libraries, making it scalable and robust in room temperatures. It also provides the quantum programming language that can train quantum computers very similar to neural networks [15, 18].

Because of the accessible quantum resources and the rigorous innovation in the field of quantum software engineering, we now can foresee second-generation Quantum-VMC architecture based models. At the same time, the diverse development of these machines via various technical collaborations has resulted in extremely distinct solutions, architectures, and use cases as applications produced using their compilation abilities and quantum programming languages.

In addition to QaaS and API-based quantum applications, various organizations and institutes are focusing on the development of different quantum-based software development environments. For each setting, the quantum software for the various simulations and tools are unique. Some are as follows :

- Entanglement as a Service is optimizing and securing the quantum-secure networks and communications

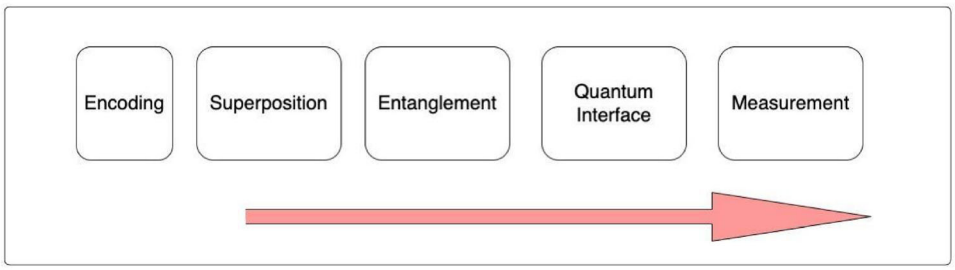

Fig. 3 Working of quantum algorithms, [14]

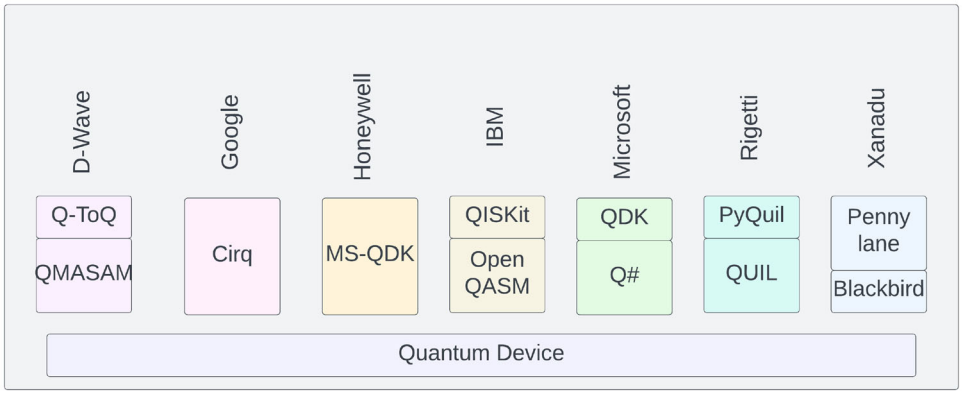

Fig. 4 Main quantum full-stack platforms [19]

while simultaneously creating the quantum networks, designed by *Aliro* [19].

- Platform as a Service is also in design phase by *Amazon* creating a QSDK cloud-based platform that can access any hardware in the existing markets, also integrating classical servers and quantum platforms.
- Hybrid systems that exists are based on the hardware-agnostic platform that captures optimization for the intraceable industry problems provided by *1QBit* [19].
- Machine Independent algorithms can compose and compute the translation to executable circuits, optimizing minimum required operations in a quantum environment developed by *Cambridge Quantum Computing* [19].

A critical problem in the field of quantum software engineering is the development of a mature discipline for quantum software engineering, with the goal of fully harnessing the potential of commercial quantum computer technology. They argue that the entire classical software engineering needs to be reinvented and extended to the quantum domain, and they define this challenge from different angles of quantum software engineering, spanning quantum computing models, languages, quantum compilers, methodologies, and tools [20].

3.2 Quantum programming language (QPGL)

Since the development of quantum computing, the **Quantum Programming Language** has been under constant development. A summary of different Quantum Programming Languages is provided in Table 1. QPGL also emphasized the integration of VMC models with quantum physics. QPGL and quantum mechanics have been combined to incorporate quantum mechanics principles such as superposition, entanglement, and interference. Shor and Grover has also demonstrated the applicability of quantum computing benefits with a quantum speed boost [21, 22].

With the understanding of computing, the **Noisy Intermediate-Scale Quantum** Computers **(NISQ)** have been designed as low-qubit level quantum computers with batch processing-based cloud network architecture constraining the communication between existing VMC models and Quantum computers. The restriction between VMC and quantum computers prohibits the development of high-level QPGL, which exists in the market for VMC-based models [22–24].

The first Quantum language was based on the **Quantum Turing Machine** but fails to provide insights for the practical QPGL. The heterogeneity of different programming paradigms has also made it difficult for a universal programming paradigm, unlike VMC-based programming paradigms.

Quantum computing industries such as IBM, Google, and Microsoft provide various quantum computational-based solutions via **PaaS** (Platform as a Service), which also enables high qubits environments. This may seem like a different environment, but one of the fascinating is that there is a relation between classical-quantum architecture yet remains stagnant across any of the platforms to be there.

Following the development of NISQ, batch run-time architecture is developed, which addresses the hindrance that persists in the present, such as decoherence, gate fidelity, and restricting the quantum run-time execution complexity. A different point of view can be discussed as various quantum computing libraries such as Criq, OpenGL, ProjectQ, Pyquill, and Qiskit [19] precisely appending to the quantum circuits and executing to their respective quantum architecture [22, 25]. This paves the way for developing high-level quantum computing, such as Ket programming language, which abstracts the QPGLs and compilations from the developers. QFC, Q, Lambda q and Quipper are the most cited QPGLs for being published as main paper et al. [26]. The CQHs puts a constraint on executing parallel codes for the classical and quantum environments. Due to heavy noise and fault-tolerant quantum computers at present, more emphasis is being

Table 1 Historical Summary of Quantum Programming Languages [7, 19]

Language	Semantics	Host	Paradigm
Lambda Calculi	Denotational	Lambda Calculus	Functional
QCL	Denotational	C	Imperative
QGCL	Operational	Pascal	Imperative
Lambda q	Operational	Lambda Calculus	Functional
Q	Operational	C++	Imperative
QFC	Denotational	Flowchart Syntax	Functional
QPALg	Operational	Process Pascal	Other
QML	Denotational	Syntax (Haskell)	Functional
CQP	Operational	Process Calculus	Other
cQPL	Denotational	Syntax Similar	Functional
LanQ	Operational	C	Imperative
NDQJava	Operational	Java	Imperative
Cove	Operational	C#	Imperative
QuECT	Denotational/Operational	Java	Circuit
Scaffold	Operational	C/C++	Imperative
QuaFL	Operational	Haskell	Functional
Quipper	Operational	Haskell	Functional
Chisel-Q	Operational	Scala	Imperative, Functional
LIQUi$\|>$	Denotational	F#	Functional
Proto-Quipper	Operational	Haskell	Functional
QASM	Denotational	Assembly Language	Imperative
FJQuantum	Operational	Feather-weight Java	Imperative
ProjectQ	Operational	Python	Imperative, Functional
pyQuil	Operational	Python	Imperative
Forrest	Operational	Python	Declarative
Open QASM	Operational	Assembly Language	Imperative
qPCF	Operational	Lambda Calculus	Functional
QWIRE	Denotational	Coq proof Assistant	Circuit
cQASM	Operational	Assembly Language	Imperative
Qiskit	Operational	Python	Imperative, Functional
IQu	Denotational	Idealized Algol	Imperative
Strawberry Fields	Operational	Python	Imperative, Functional
Blackbird	Operational	Python	Imperative, Functional
Cirq	Operational	Python	Imperative, Functional
Q#	Operational	C#	Imperative
Q$\|$SI$>$	Operational	.Net	Imperative
Silq	Operational	QRAM + C#	Imperative, Functional

Language: The name of the language.

- Semantics: The type(s) of semantics for the language.
- Host language: The classical language on (or to) which the language is based (or extended).
- item Paradigm: Each language belongs to one of the paradigms: imperative language, functional language, and circuit design language

provided on the quantum simulation on respective quantum algorithms.

4 Quantum software development lifecycle (QSDL)

Quantum software engineering cannot be successful without complete software architecture which describes the quantum software lifecycle. For high-quality software, a

reference model that supports the development and design of future software is required. There also exist various models in the software development lifecycle such as the waterfall model, evolutionary model, and spiral model [27, 28]. A quantum software lifecycle can be a lifecycle where corresponding sequential and systematic software lifecycle models at present can be initiated at a system level and develops through requirement analysis, design, testing, implementation, and maintenance.

Helping in the organization of quantum software engineering, better insights into the quantum software development phases are explored in the following hierarchy:

- Quantum software requirement analysis
- Quantum software design
- Quantum software implementation
- Quantum software testing
- Quantum software maintenance
- Quantum software reuse

The following relationship can be explored in Fig. 5. The quantum software requirements activity is intended to determine the requirements and limits of quantum software. The quantum software design activity intends to use appropriate modelling approaches and tools to define the architecture of quantum software. The quantum software implementation activity seeks to convert architectural designs into executable code using QPGLs and relevant architectures. The goal of quantum software testing is to ensure that quantum software is correct, reliable, and performs well. The architectural deployment operation involves deploying quantum software into the intended environment. The process includes supporting tasks such as architectural assessment, evolution, and management. The

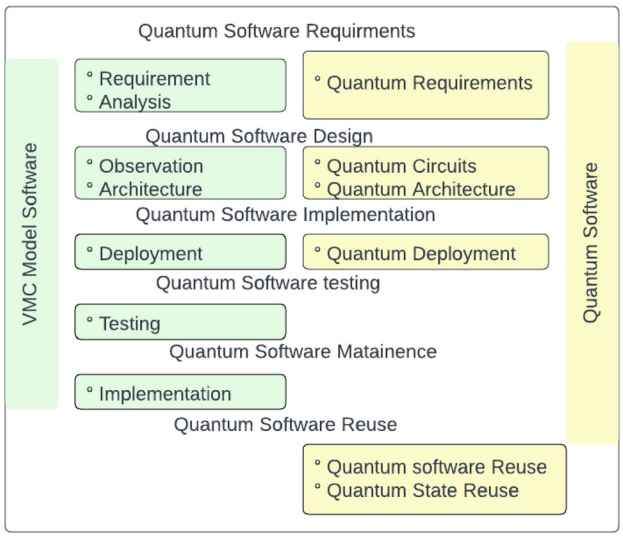

Fig. 5 Relationship between quantum software development and software engineering

quantum software maintenance aims to maintain the Quantum system scalability, maintainability, reusability, performance, and security. The quantum software reuse helps in defining crucial quantum qubits and circuits that can be implemented for different quantum softwares [29].

As quantum software engineering paces with recent trends and requirements concerning the high-level quantum-based software architecture desired, the classical approach of software engineering methods can be induced and reused for the same. The Quantum software development phase follows the problem of the defined requirement analysis and design constraints, which shall be the focus. The hierarchy described above is divided into the following sections.

4.1 Quantum software requirement analysis

For quantum software requirement analysis (QSRA), the classical methods are proposed to be reused, but the new architecture emphasizes more on the management aspect of this phase. QSRA demands the structuring of the new modeling methods and specifications that can be done for a quantum software architecture.

The QSRA has to capture the requirements from the stakeholders and establish that every scope of work is understood and the nature of completing every requirement is defined. The requirements for the QSDC, such as characteristics, attributes, functional and performance requirements quantum software architecture, are required. [29]. This proposes the research community to address the extension of the classical use cases, more defined user stories, as use cases [13]. The functional and non-functional quantum software are requirements are presented in Table 2. The functional QSRA depicts feature definitions, user requirements, use cases, and security. The non-functional QSRA focuses on the properties, expectations, and quality attributes.

4.2 Quantum software design

Quantum software design has two important aspects to look into: **quantum software architecture** and **quantum software design details** are defined for the new models of quantum architecture-based data structures, quantum algorithms, and module interaction. The quantum software architecture defines the abstract level of the assembled components and designs their interactions [7, 16].

The principles of quantum computing create difficulty in designing the quantum architecture compared to classical software architectural design [31]. This is due to the intrinsic features of quantum computing, such as superposition and entanglement. The high-level and low-level quantum software designs are referred to in Table 3. The

Table 2 Quantum software requirement analysis

Functional	Non-Functional
Defining quantum features	Defining quantum properties
Emphasis on user requirements with quantum	Emphasis on user expectations with quantum
Figuring the quantum mechanism in the use case	Figuring quantum as a quality attribute
Authentication and security levels	Usability, reliability, scalability, performance

Table 3 Quantum software design

High-level design	Low-level design
Quantum abstract particulars	Quantum data structure design
Develop Quantum Modules	Quantum Circuit and Algorithm design
Describing the functionality of each module	Quantum interface design
Understanding of quantum gates	Modules communication

patterns that are required for the quantum software and algorithm designs are referred to in Table 4. Some of the following proposed methodologies are:

UML-based Software Designing : ***Unified Model Language(UML)*** is defined as the language for general, understandable language-based modeling in the domain of classical software designing [32]. The UML can be extended with quantum specifications, which are *Class diagrams* and *Sequential diagrams* [16]. The nature of quantum computing, internal implementation, and its virtue of information handling are some of the observation that serves as the foundation for **Quantum Universal Modelling Language (Q-UML)** [16, 32, 33].

Generic Model Languages : A model-based engineering language is proposed to cater to important concepts of quantum software design like quantum - variables, states, and operations, which are designed to be independent of any existing modeling language, as well as quantum platforms and programming languages. Thus, this paves the way to design a new quantum-based modeling language, also described as **Domain Specific Language (DSL)** [34, 35].

Instead of UML and GDLs, there exists various quantum software modeling tools, i.e., Petri Nets, temporal logics, quantum4BPMN, and SysML, providing distinctive insights and advantages [29].

Table 4 Patterns of quantum algorithm designs

Pattern	Description
State preparation	Initializing the input of a quantum register
Systematic superposition	Creating an equally weighted superposition of all possible states of the qubits of a quantum register
Designing entanglement	Creating an entanglement state
Function table	Designing a function table for finite Boolean function
Black box	Reusing the computation from previous quantum algorithm
Un-compute	Removing entanglement
Phase shift	Defining important aspects of a state
Amplification	Increasing the probability of the solution
Speedup (verifying)	Achieving a speedup after verifying a solution
Quantum-classic split	Splitting the solution between a quantum and classical parts

4.3 Quantum software implementation

The initial understanding of quantum software engineering and quantum software life-cycle has emphasized the focus on quantum computing languages. It is important to understand the importance of the programming language and its suitability while considering it for a quantum-based application since the language must support at least one base quantum programming language [36, 37]. Languages available can be referred to in Table 1.

Abstraction as a quantum software requirement was necessary for quantum software. Also defined as **Abstract Data Types (ADTs)**, they are designed for data structures that can be modeled along with the quantum circuit instructions, which can be manipulated. ADTs provide a generic approach for developers designing instruction-based data structures, providing semantic variations that make it difficult to reuse. The entanglement can be seen in the functionality of the code, which confuses with the codes designed for data structures.

All of the aforementioned abstractions are bits of software functionality consisting of a series of instructions that eventually alter data and communicate with one another through a control-transfer protocol. Again, this is because VMC, upon which traditional computers are founded, requires instructions to be executed in a specific order, and on which our models must reflect reality [38]. Even while the basic computer paradigm remained the same, it was exceedingly difficult for programmers back then to change the way they conceptualized software systems [6, 39]. Since the underlying VMC model paradigm is fundamentally different, our current transformation is anticipated to be significantly more difficult. Considering that the problems that will be addressed with quantum computing belong to the Bounded-error Quantum Polynomial time class while thinking about these abstractions is intriguing. Domain-specific modeling languages may be created for these issues if their typology can be characterized [39].

4.4 Quantum software testing

Even when powerful quantum computers exist, programmers may make mistakes more frequently while creating programs for quantum computers as opposed to their classical counterparts since human intuition is far more suited to the classical world than the quantum one. Quantum software has challenging behavior. Therefore, novel quantum software testing and debugging methods must be developed. Recent years have seen an increase in studies on finding defects in quantum software as well as testing and debugging it. Designing test plans is a crucial part of test planning. Some of the test plans can be seen in Table 5.

Table 5 Quantum software testing

Designing Test Plans
Test Design Descriptions
Test Case Description
Test Report and logs
Qubits Measurements

Bug patterns are erroneous code idioms or improper coding techniques that have been repeatedly proven to fail, frequently due to an inaccurate understanding of a programming language's capabilities, the usage of erroneous design patterns, or simple blunders sharing similar behavior. Understanding the behavior of faults in quantum programs is crucial for debugging and testing quantum software. Two things that needed to be seen were bug types and patterns as well as bug benchmarks. These following bug types and their defenses are provided in Table 6.

Bug Types and Patterns : The issue kinds for specialized quantum programs to provide quantum software debugging. Based on their experiences putting various quantum algorithms into practice, they identified numerous bug categories that are unique to quantum software and suggested protection tactics for each type of defect. These bug categories include incorrect quantum initial values, inaccurate operations and transformations, incorrect compositions of operations using iteration, recursion, and mirroring, incorrect classical input parameters, and incorrect deallocation of qubits [21, 32]. To provide researchers and programmers with a clear understanding of what kinds of flaws may occur in quantum programs and how to detect them, various bug patterns in the quantum programming language, such as Qiskit, have been found and categorized, and an illustration is found of the pattern's symptoms for each bug pattern with QuSBT [40]. By identifying these similarities in a quantum programming language, programmers may uncover flaws more quickly and spend less money on software maintenance.

Table 6 Bug types and defenses

Bug Types	Solving Strategy
False Quantum initial values	Assertion checks for classical and superposition preconditions
Flawed operations and transformations	Assertion checks for unit testing
Flawed composition of operations using iteration	Assertion checks for classical intermediate states
Flawed composition of operations using recursion	Assertion checks for entangled intermediate states
Flawed composition of operations using mirroring	Assertion checks for product state postconditions
Flawed classical input parameters	Assertion checks for classical postconditions
Flawed deallocation of qubits	Assertions on algorithm postconditions

Bug Benchmarks : QBugs is a database of reproducible flaws in quantum algorithms that supports controlled experiments for the debugging and testing of quantum software [41, 42]. To help the evaluation and comparison of new research as well as the repeatability of existing research findings on quantum software engineering, QBugs offers some preliminary suggestions for developing a benchmark as the test scenarios for simulating glitchy behavior [43]. Bugs4Q facilitates downloading and running test cases for quantum software testing and gathers repeatable problems in Qiskit applications. Every genuine problem and the corresponding patches are available for investigation in the public domain. Almost all of the Qiskit bugs that are currently active are gathered and updated in real-time by Bugs4Q [44].

Code review is one example of an established software engineering method that is easily adaptable to the quantum computing field. Some are challenging to translate, like interactive debugging [2, 42]. The remainder must be included (such as the location of a validation program based on its complexity). They attempted to outline the software engineering techniques for quantum software rather than suggesting a specific testing method [38, 45]. With this description at their disposal, experts in software engineering from academia and business may begin investigating this intriguing area of quantum computing and then broaden their research to include more testing areas as well as the remaining stages of the life cycle of quantum software.

Defensive tactics in software programming help to stop various bugs and issues from re-occurring during programming to create bug-free quantum programs. However, statistical claims for quantum programs were observed based on statistical testing on VMC models [35]. With the aid of these, programmers may determine if a quantum program state corresponds to its predicted value in a classical, superposition, or entangled kind of state. Using this information, potential claims in quantum software were divided into three categories: classical assertion, superposition assertion, and entanglement assertion. They expanded Scaffold, a currently used quantum programming language. The fundamental drawback of Huang and Martonosi's assertion-based methodology is that each measurement made while debugging requires stopping the program, and the assertions demand averages of runs when measuring the real computation results *et al.* [46]. Motivated by nondestructive discrimination (NDD) and quantum error correction, this constraint must be addressed [35, 41, 47].

Robustness analysis: Robustness is defined as the quantum program that limits the difference between the output of a noisy program and that of its equivalent ideal program on the same input [2]. A semantics is defined for quantum computing with faults computed error limits for noisy versions, which also provides instances of utilizing error correction is advantageous and when there are tradeoffs between the efficiency of error corrections and associated costs, demonstrating how the technique may be used to determine the error boundaries for tiny circuits with and without error correction [44, 48].

Entanglement analysis: Entanglement analysis, which can conservatively pinpoint every potential pair of entangled qubits in the system. A programmer can use this tangled knowledge to build and troubleshoot algorithms [15, 17, 49].

A coverage criteria can be assessed with a set of rules that can be included to be satisfied by any test-case executed, which is the coverage criterion = $Gates_{Total} - Gates_{Infeasible}$, assuming the gates are reachable, feasible and software is optimal [50]. Kumar et al. [51], discusses a Modified approach for mutation testing is developed on the principle of the Coupling effect and component programmer hypothesis, identifying the

equivalent mutant to ensure the quality of the future quantum software.

4.5 Quantum software maintenance

In the creation of VMC model software, maintenance support is of utmost importance. This is motivated by the well-known statistic that the maintenance phase of software development accounts for 60 to 80 percent of the cost of the software [45]. The major goal of quantum software maintenance is to maintain and alter software programs once they are delivered to fix bugs and boost performance [3, 49]. Recent studies in this field are primarily concerned with re-engineering current conventional software systems to interface with new quantum algorithms. The quantum software maintenance can be more explained by re-engineering, which will be implemented by re-processing the classical information system to a quantum information system [2, 48]. Some of the quantum software maintenance requirements can be seen in table 7.

Re-engineering from classical to quantum information systems

The examination and alteration of the intended software system through several processes like design recovery, re-documentation, reverse engineering, and forward engineering. To create software that is of higher quality and easier to maintain, the old system will be rebuilt into a new form. By placing particular calls from VMC models to distant quantum computers in the cloud, it will be normal practice to employ quantum computers to tackle some significant issues [37, 52]. In this situation, the majority of businesses need to move and integrate their initial quantum algorithm or subsequent quantum algorithm with their current classical information systems. To address the issues posed by the migration of quantum computing and the ensuing cohabitation of classical and quantum software, re-engineering must be explored.

Model-driven reengineering is a method of software modernization that restructures classical systems along with existing or brand-new quantum algorithms to create systems that incorporate both classical and quantum information systems [12, 44]. It has been established that the software modernization approach used in traditional software engineering is an efficient mechanism for achieving software migration and evolution while preserving business knowledge [16, 33, 49]. Reducing the creation of new quantum information systems could help the answer. It would also be independent of any quantum programming languages as it is built on international standards to describe the knowledge in an agnostic manner.

To incorporate quantum computation primitives (for example, quantum software components) into an existing conventional software system is yet another challenge in re-engineering (maintenance). Since a quantum computer is fundamentally different from earlier technologies and approaches, it must not only tackle this issue at the same level as algorithm implementation but at many other significant issues that have been extensively researched in software engineering to be integrated into current software systems. A case study demonstrates how to add a quantum software component for the Boolean satisfiability problem to an already installed software system. In this case study, they examined the corresponding quality metrics for quantum components and demonstrated how to change the Boolean satisfiability problem into a quantum annealer, which is structurally distinct but mathematically identical [2, 52].

Table 7 Quantum software maintenance

Maintenance Requirements
Maintenance Scope
Quantum Architectural Design Maintenance
Quantum Modules Maintenance
Unit Testing
Classical to Quantum Re-engineering

4.6 Quantum software reuse

Throughout the whole development life-cycle, including requirement elaboration, design, and implementation, VMC model software may be systematically reused. Even in the stages of development after delivery, such as ongoing quality assurance or software maintenance, it has a place [16, 34, 53]. This section provides a current state-of-the-art overview of quantum software reuse in many areas, including quantum state, quantum circuit, and quantum pattern reuse.

4.6.1 Quantum circuit reuse

From the standpoint of software reuse, a few quantum patterns can aid in the development of quantum algorithms. Additionally, a representation of these quantum patterns in a pattern repository, a specialized database used to store pattern documents and maintain linkages among them [54, 55]. Such a quantum pattern repository supports browsing the content of each pattern document and enables pattern navigation based on links between patterns, allowing a quantum software developer to query the database to find appropriate quantum patterns (for example, to identify the entry pattern corresponding to a problem). In this fashion, a developer of quantum software may select

appropriate patterns from the repository, which might span multiple distinct domains, and combine them to address a challenging issue.

There are a few quantum circuit design approaches that are now known; most of them are based on heuristic search methods like genetic algorithms and simulated annealing [45, 56]. These techniques, however, are restricted to a tiny range of circuit sizes, and the results they produce are frequently illogical. A fresh approach to designing quantum circuits that are built precisely on the notion of building new quantum circuits utilizing parts of old ones [7, 2]. The design principle's distinguishing feature is that if we already have a useful collection of quantum circuits, we may systematically build effective quantum circuits by reusing and merging a collection of highly optimized quantum circuits. This design method's solid mathematical underpinnings provide understandable and accurate descriptions of the circuits that are produced. Authors in [2] proposed that it could be worthwhile to create a database of medium-sized matrix groups with functional quantum circuits from a practical standpoint. Linear algebra may then search this database for a particular transformation, and automatically generating quantum circuit implementations in this way could be appealing [35]. Qubit-resue compilation, a method achieved on ion-trapped models where quantum circuits, using very few qubits were executed in compilation of mid-circuit measurement and resets. Algorithms such as local brickwork circuits, quantum convolutional neural networks and Berstein-Vazirani were implemented to preserve the total gates in the circuit resulting in increase of circuit depths,as discussed by DeCross et al. [57]

4.6.2 Quantum state reuse

The main idea is that in the future, manufacturers may create a valuable quantum state and make several copies of that quantum state using a particular device. These copies can then be checked for quality and kept until they are required [6, 9, 58, 59]. Customers may pay to download that state, which they can then use to speed up their quantum computer.

> "Some quantum states are hard to create and maintain but are a valuable resource for computing. Twenty-first-century entrepreneurs could make a fortune selling disposable quantum states" [13]

A fascinating concept known as plug-in quantum software was proposed for perhaps reusing the quantum states that make up a quantum software program. There are fault-tolerant universal set quantum gates that are simple to construct and others that are challenging for every known quantum error correction system. Quantum software, which may be created beforehand and then consumed while the quantum gate is being operated, can effectively execute the latter. Using software to implement the gate has the benefit of allowing one to check that it is prepared per requirements before usage. Quantum software metrics are primarily concerned with assessing the size and structure of quantum software.

5 Case study

The case study provides better insights into the domain of quantum software engineering. We will be looking at two of the cases, **Hybrid Quantum Applications** and **Quantum Computing Service (QCS)** [33, 43]. The integration of new technology and conventional methodologies provides a breakthrough in developing these quantum-based technologies that showcase necessary advancement.

5.1 Hybrid quantum application

A hybrid system is made up of a VMC model and a quantum co-processor coupled by a feedback loop. Data processing, error correction, and optimisation are all examples of tasks that classical systems can accomplish reliably, quickly, and scalably. A quantum co-processor can do operations that are intractable for VMC model, such as factoring big numbers and solving NP-hard problems. The feedback loop allows the VMC model to regulate the quantum co-processor based on the measurement findings. The hybrid system can thus make use of both VMC model and quantum systems while avoiding their drawbacks. [60]. The life cycles of various software artifacts must be integrated to create hybrid quantum applications. Then, we go over the many stages of the life cycle of developing quantum software. The hybrid quantum application may have many quantum algorithm implementations; for example, clustering may be performed first, and then a classifier may be trained using the clustering findings. Thus, the pre-processing chores are carried out by conventional programs and on conventional machines. When running the quantum programs, pre-processing, for instance, generates state preparation circuits depending on input data to initialize the register of the quantum computer, shown in Fig. 6. The major challenges exists in hybrid quantum applications are developing and maintaining large-scale quantum computers; designing optimized, efficient and scalable quantum algorithms; and identifying advantages and sustainablity over VMC models over given set of use cases.

Coordination between the quantum algorithm instruction and classical programs of hybrid quantum applications for the transfer of the necessary data shall be established

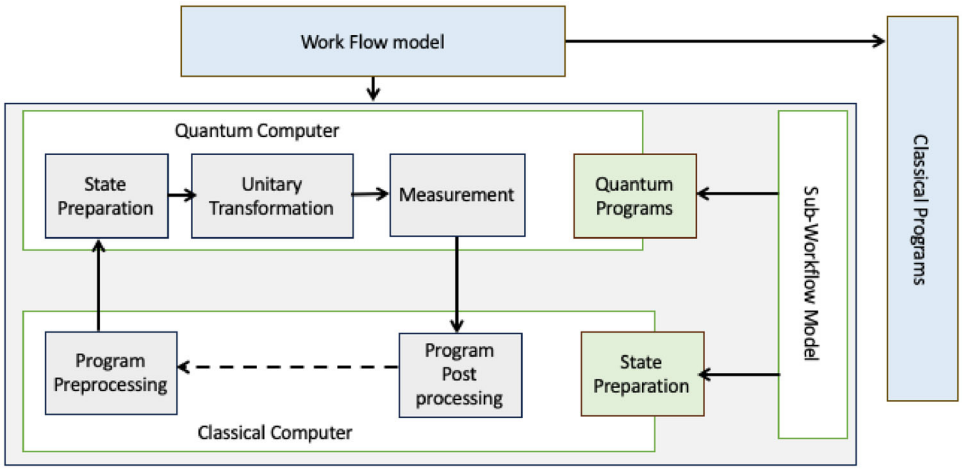

Fig. 6 Hybrid quantum structure

between them. Workflow technology is an orchestration strategy that has been shown to work in a variety of diverse application domains throughout the years. Workflows should thus be utilized to coordinate the programs that make up a quantum application [49]. For this, so-called workflow models are used to specify the necessary operations involving the execution of the quantum and classical programs, their sequencing, and the data flow between them.

As shown in Fig. 7 (i) the quantum workflow life-cycle, (ii) the classical software life-cycle, and (iii) the quantum circuit lifecycle, the quantum software development life cycle contains many life cycles that are intertwined. Additionally, it is necessary to manage the numerous software artifacts that are prepared for and carried out during the operation's life cycle. To support quick and frequent releases, developers and operations staff should be closely connected using the widely accepted DevOps concept [7, 27].

Distinct studies have proposed distinct business process management life cycles and processes. Leymann and Roller and Dumas et al. [31] life cycles were discussed as the foundation for our life cycle for quantum processes. The operation of each generated software artifact making up a quantum application is required. This covers, for instance, how the quantum application is packaged to be sent to the target environment or how it is deployed. For the quantum computing area, different concepts and techniques are needed from traditional DevOps.

In comparison to the splitting of quantum applications and quantum processes, the splitting in the quantum circuit lifetime is splitting with the lowest level of granularity. It is

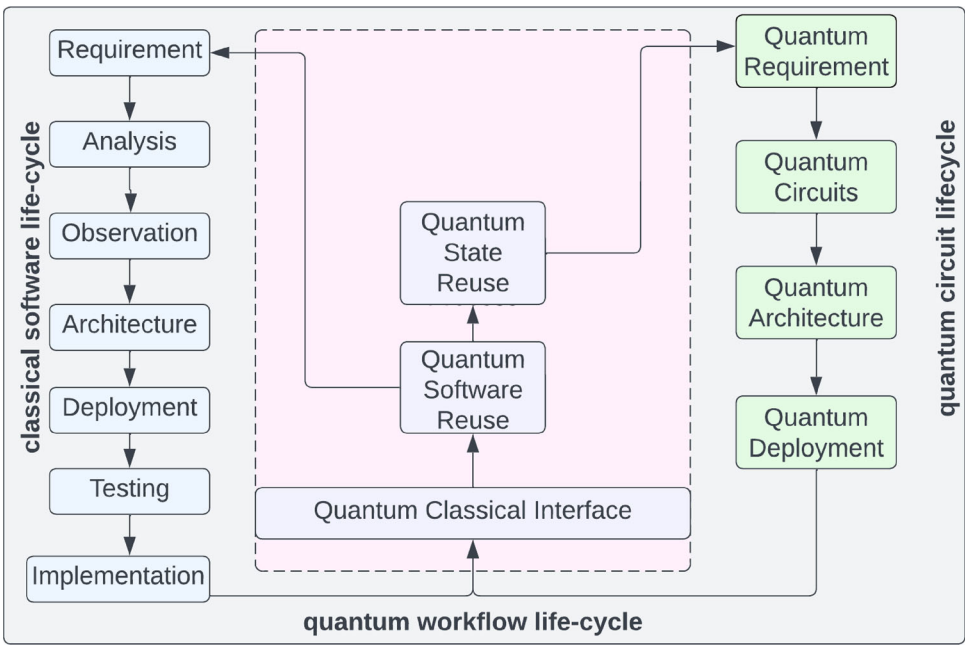

Fig. 7 Quantum software development life-cycle

meant to determine if a pure quantum algorithm or a hybrid quantum algorithm should be employed. It is entered with a description of the issue to solve. Implementing the relevant quantum circuit is necessary. Because of this, the implementation should be hardware-independent to allow for a subsequent hardware choice depending on the present properties of the many quantum computers that are now available. The quantum circuit can thus be utilized to solve many instances of the issue. Many different technologies can be used to implement the quantum circuit.

To guarantee that the quantum circuit behaves as intended, which is tested and confirmed [32, 38]. One strategy is to include statistical claims in the quantum circuit [28, 32]. When the quantum circuit is run up until the assertion is defined, it is then confirmed that the stated state is measured. As a result, programmers are guided in detecting flaws by the assertions' outcomes [61]. However, this calls for running the circuit for every claim, which is only practical for brief quantum circuits with a limited number of assertions. First methods also make an effort to runtime dynamically verify claims. But then more ancilla qubits and gates are needed, which restricts the application with the constrained quantum computers of today [62].

It is not necessary to solve a specific issue instance to build the quantum circuit from the hardware-independent implementation phase. As a result, it is enhanced with the information needed to resolve a specific instance of the problem during this stage. There are two phases to this enrichment: state preparation and oracle expansion, respectively. Based on the input data, a circuit initializing the quantum computer's register with the necessary state is constructed for the state preparation stage. After then, the original circuit is prepended with the state preparation circuit that was created. As a result, other encoding, such as the angle, amplitude, or basis encoding, exist.

For the existing base of business applications, classical computing is currently demonstrating its ability to perform across a wide range of solutions. However, some of the computations they must perform can be accelerated using quantum computing, analogous to how GPUs are utilised today. As a result, quantum systems should not operate in isolation, but rather coexist and interact with classical systems. The aim of the proposed Quantum Algorithm card is twofold, enacting firstly as it acts as a repository for insights into the algorithm's implementation, and secondly, it transmits vital information to users who rely on the implementation to meet the application's specific requirements. [60]

5.2 Quantum computing services

Given the current state of quantum computing, the greatest comparison for this access would be to the more conventional Infrastructure as a Service model, which allows customers to interact directly with the cloud's physical resources. Using a quantum programming language supported by the hardware they use, users may create and run quantum programs. The program is sent to a scheduler for execution, which eventually sends the code to the quantum computer whenever a time slot becomes available. The Quantum Application as a Service idea is of special importance in the unique realm of quantum services [62, 66]. The primary obstacles arise from two areas: technical combining VMC models and quantum components, and process matching the technological solution with user needs and requirements with respect to a full stack quantum software development [63]. The authors in [18, 64–66] advise using a conventional application programming interface (API) to include all quantum application and deployment functionality. Then, in quantum computing as a service offering, this API may take care of input data encoding and launching the quantum software, easing the integration of the quantum program with conventional applications. Quantum software development life-cycle as an API [18, 31] is demonstrated in Fig. 8.

For certain quantum computing systems, a quantum algorithm is executed using pulses generated by quantum circuits. Today, we have numerous methods for deploying programs to execution. Containers, merely delivering instructions, and quantum circuits are all viable options. The design of this deployment and execution middleware requires no unique considerations related to quantum computing, and regular software architecture will meet the requirements. However, because the middleware interacts directly with the quantum computer, it cannot be easily shared with other quantum computers. The following provides the existing open-source quantum API's at the moment: $\bullet \boldsymbol{GoogleCirc - https://quantumai.google/cirq}$ $\bullet \boldsymbol{IBMQiskit - https://qiskit.org/}$

Some major constraints added to for quantum computing services are variability in design, deployment and in interpretation [67].

A service composition paradigm called an API Gateway was created to enable the development of end-user applications based on the combination of many micro-services. The system's entry point, the API Gateway, directs queries to the proper micro-services. Additionally, it can modify protocols, calls aggregate results, and apply common logic.

A Python and Flask implementation of the Quantum API Gateway idea for the Amazon Bracket quantum computing platform is presented. This platform was chosen because it provides a comprehensive approach to running quantum programs in various quantum processors. The API Gateway enables programmers to select between gate-based or annealing machines at run-time depending on the type of code to be executed (presuming both quantum

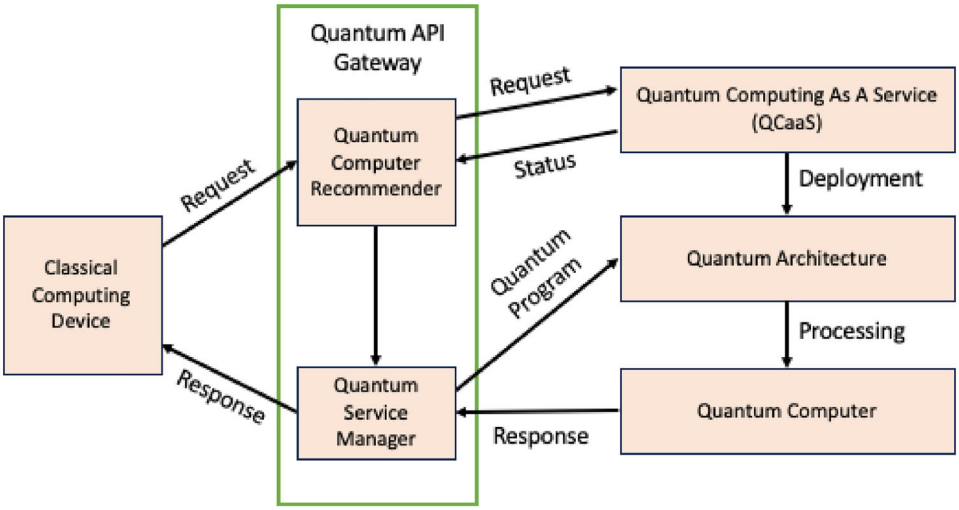

Fig. 8 Quantum software development life-cycle as an API [18, 31]

service implementations are accessible). For gate-based computers, the necessary number of qubits is utilized to identify those with sufficient processing capacity. Calculate the cost per computer chosen. To that purpose, the number of shots specified by the developer, the cost per shot, and the intended maximum cost threshold are employed, with "shots" defined as the number of repetitions necessary to identify the solution and, therefore, the service execution outcomes in Eq. 6.

$$Cost_{Service} = \frac{Cost_{Execution}}{Execution} + \left(\frac{Cost_{shots}}{Shot} * N_{shots}\right) \quad (6)$$

The estimated execution time for each machine is calculated based on the parameters of the execution, its context (particularly the day of the week and the hour of execution start), and the actual time taken by previous executions done on that VMC model. In addition, a time variable connected to the analysis of prior executions as a time series is included [49]. As a consequence, the resultant time includes both the time spent waiting in the machine's execution queue and the time necessary to perform the quantum service. The following Quantum API Gateway architecture is modular. Both of the static and dynamic features, as well as those used to estimate execution time, can be supplemented with variables from other vendor suppliers [4, 62].

The Quantum API Gateway has two major endpoints: one for obtaining suggestions regarding which quantum machine to carry out the service on, and the second for providing observations about the time it took for the service to finish execution, which records and enhances the execution time prediction model.

Execute/ "GET" method : It performs the Quantum API Gateway optimization with the variables and settings specified preceding and delivers the most ideally suited machine for execution of the code.

Feedback/ "POST" method : To enhance the execution time estimation model, it allows reporting to the Quantum API Gateway the amount of time necessary to carry out the service in a given VMC model, together with the executed service's description; qubits and shots and context factors day of the week and time of it.

A query made on the Quantum API Gateway consumed 26.671 seconds on average. 26.668 seconds (99.98 percent of the Quantum API Gateway time) consisted of inquiring about the real-time status of the various machines accessible via the Amazon Bracket API. The suggestion process was executed in less than 0.01 seconds. Using a caching technique to monitor the state of the quantum computer might greatly cut querying time but at the expense of precision. Another option is that the platform will provide a real-time API to examine this data [62]. One such example for QCS is QPath, which is an ecosystem that bridges the gap between classical and quantum domains within a Quantum Development and Application Lifecycle platform, providing access to a broad range of potential applications with cutting-edge quantum software and quantum containerization as well as quantum continuous development [68]. Some of the successful use cases are presented as QaaS web services. The application of quantum web services using amazon brackets, openAPI and github extensions [69]. Another successful usecase is, **QFaaS** (Quantum function as a Service) is first serverless

based quantum framework, which tries to unify different QC providers under a single framework. [70]

6 Discussion and future works

The introduction of quantum programming languages has fueled the development of quantum software in its brief existence. At the same time, quantum software development has traditionally been associated with quantum programming. While such languages have assisted in popularising quantum programming, designing new languages is not a long-term solution. It is critical, especially in this case, that a comprehensive software engineering discipline be established for quantum software development [27, 71]. A swift advancement is being achieved in the domain of quantum software engineering as well as quantum software lifecycle. However, claiming maturity in the themes treated by the present piece would be incorrect; certainly, while numerous strategies have arisen, further expertise is needed to evaluate their relative strengths as well as, to concentrate suggestions into a limited number of key approaches.

The major advice for researchers is that they need to be quick to update the review's findings in the coming years [36]. In this regard, the items below emphasize some significant research recommendations to make quantum programming more practical:

- QPLs: The original languages' difficulty was the lack of higher abstraction techniques. Furthermore, several control structures and operations that are typical in classical programming languages will almost certainly be included. This is because most contemporary quantum programming languages still have shortcomings when compared to their conventional counterparts.
- Simulator and Environments: Several simulators are already available to help users understand quantum computing foundations and design quantum algorithms. However, the next degree of abstraction is required for design environments that concentrate more on software design rather than quantum circuits. We are referring to the high-level design of completely hybrid software systems, which involve current modeling languages such as Unified Modeling Language (UML) or other domain-specific languages.
- Optimizers: Optimization is critical for quantum computing success, especially as quantum computers progressively increase in the number of qubits. It is not, however, a key topic of quantum software development because it is more of a challenge of co-designing quantum hardware and the firmware required for optimizing programs for the specific architecture and other shortcomings.
- Quantum Software Development kits: Integrated development environments (IDEs) for creating, writing, and executing quantum programs, like high-level tools for designing, must be enhanced. Future Quantum IDEs must add standard features that current IDEs for classical software have, such as support for many quantum programming languages and technology agnosticism.
- Error Correction Tools: Error correction is an area undergoing intense investigation since it is critical to maximizing the full capacity of quantum computers and, by extension, the software programs that are executed on those machines.

The need for exceptional quantum solutions will skyrocket in the next years. However, the creation of such applications is hard and requires the participation of professionals from numerous sectors. Common knowledge of the development procedure for quantum software is required to enable their effective collaboration and to facilitate the teaching of future quantum software developers [32, 72].The meta-level development step is concerned with software that transforms hybrid design specifications into quantum source code. A rudimentary grasp of the quantum mechanism and software engineering is required for quantum programming. Because we are dealing with circuits and quantum programming languages, we rarely use approaches that are typical in programming. We are still in the beginning phases of quantum software development. More activities are required in terms of building tools and technologies, quantum simulators, running programs on servers, constructing circuit diagrams, and creating an outline for quantum programming. Implementing code quality standards decreases a wide range of issues and the probability of failure of the project [16, 41, 47]

7 Conclusion

Quantum computation is, without a doubt, a technological revolution that we are witnessing. This revolution is akin to nineteenth-century industrialization, which gave the beginning of the machine era or to the developments that occurred during the twentieth-century information age. In this paper, we focused on the recent trends that are found in quantum software engineering and diving more deeply into the quantum software life-cycle.

Quantum computing, in general, and quantum software engineering, in particular, are highly engaged research topics with novel theories and techniques being published regularly. As a result, the quantum software development

life-cycle is a live document that can be updated and expanded in response to new advances. This includes, for example, incorporating new concepts and tools or the addition of an additional growth phase.

Author contributions All authors contributed equally to the paper. All authors read and approved the final manuscript.

Funding Open Access funding provided by University of Jyväskylä (JYU). This work has been supported by the Academy of Finland (project DEQSE 349945) and Business Finland (project TORQS 8582/31/2022). Open Access funding provided by University of Jyväskylä (JYU).

Data availibility All relevant data are included in the article.

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Endo, S., Cai, Z., Benjamin, S.C., Yuan, X.: Hybrid quantum-classical algorithms and quantum error mitigation. J. Phys. Soc. Jpn. **90**(3), 032001 (2021)
2. Huang, Y., Martonosi, M.: Qdb: from quantum algorithms towards correct quantum programs. arXiv preprint arXiv:1811.05447 (2018)
3. Nielsen, M.A., Chuang, I.L.: Quantum computation and quantum information (2010)
4. Ying, M., Duan, R., Feng, Y., Ji, Z.: Predicate transformer semantics of quantum programs. Semant. Tech. Quantum Comput. **8**, 311–360 (2010)
5. Yan, P., Jiang, H., Yu, N.: On incorrectness logic for quantum programs. Proc. ACM Program. Lang. **6**(OOPSLA1), 1–28 (2022)
6. Mielke, A., Ricken, T.: Evaluating artificial neural networks and quantum computing for mechanics. Pamm **19**(1), 201900470 (2019)
7. Zhao, J.: Quantum Software Engineering: Landscapes and Horizons.(2020), (2007)
8. Einstein, A., Podolsky, B., Rosen, N.: Can quantum-mechanical description of physical reality be considered complete? Phys. Rev. **47**(10), 777 (1935)
9. Endo, S., Benjamin, S.C., Li, Y.: Practical quantum error mitigation for near-future applications. Phys. Rev. X **8**(3), 031027 (2018)
10. Smith, R.S., Curtis, M.J., Zeng, W.J.: A practical quantum instruction set architecture. arXiv preprint arXiv:1608.03355 (2016)
11. Bernstein, E., Vazirani, U.: Quantum Complexity Theory, (1993)
12. Dwivedi, K., Nigam, A.: Quantum deep learning for phoniatrics biomarker based disease detection deep learning model for disease detection by phoniatrics biomarkers using quantum computation. In: 2nd International Conference on Data, Engineering and Applications (IDEA), pp. 1–8 (2020). IEEE
13. Hassija, V., Chamola, V., Saxena, V., Chanana, V., Parashari, P., Mumtaz, S., Guizani, M.: Present landscape of quantum computing. IET Quantum Commun. **1**(2), 42–48 (2020)
14. Awan, U., Hannola, L., Tandon, A., Goyal, R.K., Dhir, A.: Quantum computing challenges in the software industry a fuzzy ahp-based approach. Inf. Softw. Technol. **147**, 106896 (2022)
15. Ajagekar, A., Humble, T., You, F.: Quantum computing based hybrid solution strategies for large-scale discrete-continuous optimization problems. Comput. Chem. Eng. **132**, 106630 (2020)
16. Zhao, J.: Some size and structure metrics for quantum software. In: 2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE), pp. 22–27 (2021)
17. Mlnarık, H.: Quantum Programming Language LanQ, (2007)
18. Garcia-Alonso, J., Rojo, J., Valencia, D., Moguel, E., Berrocal, J., Murillo, J.M.: Quantum software as a service through a quantum api gateway. IEEE Internet Comput. **26**(1), 34–41 (2021)
19. Serrano, M.A., Cruz-Lemus, J.A., Perez-Castillo, R., Piattini, M.: Quantum software components and platforms: overview and quality assessment. ACM Comput. Surv. **55**(8), 1–31 (2022)
20. Stepney, S., Braunstein, S.L., Clark, J.A., Tyrrell, A., Adamatzky, A., Smith, R.E., Addis, T., Johnson, C., Timmis, J., Welch, P.: Journeys in non-classical computation i: a grand challenge for computing research. Int. J. Parallel Emerg. Distribut. Syst. **20**(1), 5–19 (2005)
21. Ömer, B.: Classical concepts in quantum programming, 2002. ArXiv. org: quant-ph/0211100
22. Preskill, J.: Quantum computing in the nisq era and beyond (2018). arXiv preprint arXiv:1801.0086261 (1801)
23. Almudever, C.G., Lao, L., Fu, X., Khammassi, N., Ashraf, I., Iorga, D., Varsamopoulos, S., Eichler, C., Wallraff, A., Geck, L., : The engineering challenges in quantum computing. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, pp. 836–845 (2017). IEEE
24. Killoran, N., Izaac, J., Quesada, N., Bergholm, V., Amy, M., Weedbrook, C.: Strawberry fields: A software platform for photonic quantum computing. Quantum **3**, 129 (2019)
25. Khammassi, N., Guerreschi, G.G., Ashraf, I., Hogaboam, J.W., Almudever, C.G., Bertels, K.: CQASM v1.0: Towards a common quantum assembly language (2018) arXiv:1805.09607 [quant-ph]
26. Garhwal, S., Ghorani, M., Ahmad, A.: Quantum programming language: a systematic review of research topic and top cited languages. Arch. Comput. Methods Eng. **28**, 289–310 (2021)
27. Moguel, E., Berrocal, J., García-Alonso, J., Murillo, J.M.: A roadmap for quantum software engineering: Applying the lessons learned from the classics. In: Q-SET@ QCE, pp. 5–13 (2020)
28. Boehm, B.: A view of 20th and 21st century software engineering. In: Proceedings of the 28th International Conference on Software Engineering, pp. 12–29 (2006)
29. Zhao, X., Xu, X., Qi, L., Xia, X., Bilal, M., Gong, W., Kou, H.: Unraveling quantum computing system architectures: an extensive survey of cutting-edge paradigms. Inf. Softw. Technol. **167**, 107380 (2024)
30. Huang, Y., Martonosi, M.: Statistical assertions for validating patterns and finding bugs in quantum programs.(may 2019).

Google Scholar Google Scholar Digital Library Digital Library (2019)
31. Leymann, F., Barzen, J., Falkenthal, M., Vietz, D., Weder, B., Wild, K.: Quantum in the cloud: application potentials and research opportunities. arXiv preprint arXiv:2003.06256 (2020)
32. Li, H., Khomh, F., Openja, M.: Understanding quantum software engineering challenges an empirical study on stack exchange forums and GitHub issues. In: 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 343–354 (2021)
33. Ali, S., Yue, T., Abreu, R.: When software engineering meets quantum computing. Commun. ACM **65**(4), 84–88 (2022)
34. Barbosa, L.S.: Software engineering for'quantum advantage'. In: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, pp. 427–429 (2020)
35. Bass, L., Klein, M., Bachmann, F.: Quality Attribute Design Primitives, (2000)
36. Piattini, M., Peterssen, G., Perez-Castillo, R.: "Quantum computing: a new software engineering golden age. ACM SIGSOFT Softw. Eng. **45**(3), 12–14 (2020)
37. Zuliani, P.: Quantum programming. PhD thesis, University of Oxford (2001)
38. Miszczak, J.A.: Models of quantum computation and quantum programming languages. arXiv preprint arXiv:1012.6035 (2010)
39. Akbar, M.A., Khan, A.A., Mahmood, S., Rafi, S.: Quantum software engineering: A new genre of computing. arXiv preprint arXiv:2211.13990 (2022)
40. Ali, S., Yue, T.: Quantum software testing: A brief introduction. In: 2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), pp. 332–333 (2023). IEEE
41. Ali, S., Arcaini, P., Wang, X., Yue, T.: Assessing the effectiveness of input and output coverage criteria for testing quantum programs. In: 2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST), pp. 13–23 (2021). IEEE
42. Hietala, K.: Quantum programming languages (2016)
43. Gyongyosi, L., Imre, S.: A survey on quantum computing technology. Comput. Sci. Rew. **31**, 51–71 (2019)
44. Cartiere, C.R.: Quantum software engineering: bringing the classical software engineering into the quantum domain. PhD thesis, Master's Thesis, University of Oxford, Department of Computer Science (2013)
45. Grattage, J.: Qml: A functional quantum programming language. PhD thesis, University of Nottingham Nottingham, UK (2006)
46. Li, G., Zhou, L., Yu, N., Ding, Y., Ying, M., Xie, Y.: Projection-based runtime assertions for testing and debugging quantum programs. Proc. ACM Program. Languages **4**, 1–29 (2020)
47. Ramezani, S.B., Sommers, A., Manchukonda, H.K., Rahimi, S., Amirlatifi, A.: Machine learning algorithms in quantum computing: A survey. In: 2020 International Joint Conference on Neural Networks (IJCNN), pp. 1–8 (2020)
48. Kakutani, Y.: A logic for formal verification of quantum programs, (2009)
49. Orús, R., Mugel, S., Lizaso, E.: Quantum computing for finance: overview and prospects. Rev. Phys. **4**, 100028 (2019)
50. Kumar, A.: Formalization of structural test cases coverage criteria for quantum software testing. Int. J. Theoretical Phys. **62**(3), 49 (2023)
51. Kumar, A., Kwatra, P., Garhwal, S.: Development of a tool for finding equivalent mutants in quantum program: a perspective to measure the quality of quantum software (2022)
52. Grubb, P., Takang, A.A.: Software Maintenance: Concepts and Practice, (2003)
53. Allouche, C., Baboulin, M., Brugière, T., Valiron, B.: Reuse method for quantum circuit synthesis. In: Recent Advances in Mathematical and Statistical Methods: IV AMMCS International Conference, Waterloo, Canada, August 20–25, 2017 IV, pp. 3–12 (2018). Springer
54. De Stefano, M., Pecorelli, F., Di Nucci, D., Palomba, F., De Lucia, A.: Software engineering for quantum programming: how far are we? J. Syst. Softw. **190**, 111326 (2022)
55. Zhao, P., Zhao, J., Ma, L.: Identifying bug patterns in quantum programs. In: 2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE), pp. 16–21 (2021). IEEE
56. Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., Lloyd, S.: Quantum machine learning. Nature **549**(7671), 195–202 (2017)
57. DeCross, M., Chertkov, E., Kohagen, M., Foss-Feig, M.: Qubit-reuse compilation with mid-circuit measurement and reset. Phys. Rev. X **13**(4), 041057 (2023)
58. Vartiainen, J.J., Möttönen, M., Salomaa, M.M.: Efficient decomposition of quantum gates. Phys. Rev. Lett. **92**(17), 177902 (2004)
59. Awschalom, D., Berggren, K.K., Bernien, H., Bhave, S., Carr, L.D., Davids, P., Economou, S.E., Englund, D., Faraon, A., Fejer, M.: Development of quantum interconnects (quics) for next-generation information technologies. PRX Quantum **2**(1), 017002 (2021)
60. Stirbu, V., Haghparast, M.: Quantum algorithm cards: Streamlining the development of hybrid classical-quantum applications. In: International Conference on Product-Focused Software Process Improvement, pp. 125–130 (2023). Springer
61. Taibi, D., Lenarduzzi, V., Pahl, C.: Architectural Patterns for Microservices: A Systematic Mapping Study, pp. 221–232 (2018)
62. Rahaman, M., Islam, M.M.: A review on progress and problems of quantum computing as a service (QCAAS) in the perspective of cloud computing. Glob. J. Comput. Sci. Technol. **15**(4), 15–18 (2015)
63. Stirbu, V., Haghparast, M., Waseem, M., Dayama, N., Mikkonen, T.: Full-stack quantum software in practice: ecosystem, stakeholders and challenges. In: 2023 IEEE International Conference on Quantum Computing and Engineering (QCE), vol. 2, pp. 177–180 (2023). IEEE
64. Aleksandrowicz, G., Alexander, T., Barkoutsos, P., Bello, L., Ben-Haim, Y., Bucher, D., Cabrera-Hernández, F.J., Carballo-Franquis, J., Chen, A., Chen, C.-F., et al.: Qiskit: An open-source framework for quantum computing. Accessed on: Mar **16** (2019)
65. Steiger, D.S., Häner, T., Troyer, M.: Projectq: an open source software framework for quantum computing. Quantum **2**, 49 (2018)
66. Svore, K., Geller, A., Troyer, M., Azariah, J., Granade, C., Heim, B., Kliuchnikov, V., Mykhailova, M., Paz, A., Roetteler, M.: Q# enabling scalable quantum computing and development with a high-level dsl. In: Proceedings of the Real World Domain Specific Languages Workshop 2018, pp. 1–10 (2018)
67. Haghparast, M., Mikkonen, T., Nurminen, J.K., Stirbu, V.: Quantum software engineering challenges from developers' perspective: Mapping research challenges to the proposed workflow model. In: 2023 IEEE International Conference on Quantum Computing and Engineering (QCE), vol. 2, pp. 173–176 (2023). IEEE
68. Moguel, E., Garcia-Alonso, J., Haghparast, M., Murillo, J.M.: Quantum Microservices Development and Deployment (2023)

69. Romero-Álvarez, J., Alvarado-Valiente, J., Moguel, E., Garcia-Alonso, J.: Quantum web services: Development and deployment. In: International Conference on Web Engineering, pp. 421–423 (2023). Springer
70. Nguyen, H.T., Usman, M., Buyya, R.: Qfaas: A serverless function-as-a-service framework for quantum computing. arXiv preprint arXiv:2205.14845 (2022)
71. Gayathri, S., Kumar, R., Dhanalakshmi, S., Kaushik, B.K., Haghparast, M.: T-count optimized wallace tree integer multiplier for quantum computing. Int. J. Theoretical Phys. **60**(8), 2823–2835 (2021)
72. Hilton, J.: Building the quantum workforce of the future. Forbes Technology Council (2019)

Kanishk Dwivedi is pursuing Intelligent Adaptive Systems as master's degree with a Bachelor of Engineering in Computer Science and Engineering as a major. He has worked and gained knowledge as an intern from various industries and startups with a focus on software development and deployment as well as the implementation of new technologies. I have research and professional experience in AI, blockchain, DevOps (CI / CD), and quantum computation. His research interests are Intelligent Systems, Quantum computing, and Quantum computational neuroscience.

Majid Haghparast is an IEEE senior member and a researcher scientist at the University of Jyväskylä, Finland. From April 2017 to January 2018, he conducted his sabbatical with Johannes Kepler University, Linz, Austria, where he was also a Research Fellow. He is an Associate Editor of the Cluster Computing (Springer) and Journal of Computational Electronics (Springer). Majid is also an Editorial Board Member of Optical and Quantum Electronics (Springer). He has been a supervisor/advisor of more than 10 Ph.D. theses. Majid is the invited referee for more than 30 prestigious journals, including IEEE Internet of Things, IEEE Transactions on Computer, IEEE Transactions on Circuits and Systems I, IEEE Transactions on Industrial Electronics, IEEE Transactions on Quantum Engineering. His focus is on quantum computing, where he has been involved in different projects.

Tommi Mikkonen is a full professor of software engineering at the University of Jyväskylä, Finland. His research interests lie in empirical software engineering, with a particular focus on topics that are relevant to software engineering at large from the company's perspective. For such research, he has been awarded twice by Tampere University of Technology, Finland, where he worked 2001-2017, for successful company collaboration. In the field of quantum software engineering, Mikkonen's research interests are two-fold. On the one hand, he is doing research to bridge the gap in developer experience in quantum and classical software; the former relies on tools and techniques that support individuals' creativity in creating new, innovative applications, whereas the latter emphasizes the role of collaboration and frequent release cycles where software is gradually developed. On the other hand, he is interested in creating a roadmap that helps companies discover opportunities for using quantum software in their daily processes and activities. Over the years, Mikkonen has authored more than 300 research articles of different formats, as well as supervised more than 500 theses of different levels, including, in particular, 27 doctoral theses, where he has been either the main or co-supervisor. He is a member of several scientific communities, including in particular, the International Conference on Web Engineering, where he has been a member of the steering group since 2022.