Challenges and Opportunities in Quantum Software Architecture*

Tao Yue[1][0000−0003−3262−5577], Wolfgang Mauerer[2,3][0000−0002−9765−8313], Shaukat Ali[1][0000−0002−9979−3519], and Davide Taibi[4,5][0000−0002−3210−3990]

[1] Simula Research Laboratory, Kristian August Gate 23, 0164 Oslo, Norway
tao,shaukat@simula.no
[2] Technical University of Applied Sciences Regensburg, Galgenbergstraße 32, 93053 Regensburg, Germany
[3] Siemens AG, Corporate Technology, Otto-Hahn-Ring 6, 81739 München, Germany
wolfgang.mauerer@othr.de
[4] University of Oulu, Oulu, Finland
[5] Tampere University, Tampere, Finland
davide.taibi@oulu.fi

Abstract. Quantum computing is a relatively new paradigm that has raised considerable interest in physics and computer science in general but has so far received little attention in software engineering and architecture. Hybrid applications that consist of both quantum and classical components require the development of appropriate quantum software architectures. However, given that quantum software engineering (QSE) in general is a new research area, quantum software architecture–a sub-research area in QSE is also understudied. The goal of this chapter is to provide a list of research challenges and opportunities for such architectures. In addition, to make the content understandable to a broader computer science audience, we provide a brief overview of quantum computing and explain the essential technical foundations.

Keywords: Quantum Computing · Software Architecture · Quantum Applications · Quantum Software Engineering

1 Introduction

There is, in general, the consensus among the quantum computing (QC) community that QC applications will primarily be hybrid; that is, some features will be executed efficiently on classical computers, whereas some specialized features will be executed efficiently on quantum computers [1,2]. This pattern is by pure choice; most known quantum algorithms, from Shor's seminal factoring

* Wolfgang Mauerer's work is funded by the German Federal Ministry of Education and Research within the funding program quantum technologies—from basic research to market—contract numbers 13N15645 and 13N16092. Tao Yue and Shaukat Ali are supported by Qu-Test (Project#299827) funded by the Research Council of Norway and Simula's internal strategic project on Quantum Software Engineering.

algorithm to recent variational approaches, rely on classical sub-routines at their very core that cannot be beneficially replaced by pure quantum approaches. As a result, there will be close interaction between classical and quantum software, demanding careful consideration of both worlds when designing quantum software architectures (QSAs). QSAs of quantum software define the high-level structure of the software system, including its components, their interactions, and data flow.

QSA is a relatively new research area within the broader area of quantum software engineering (QSE). Therefore, we believe it is crucial to identify high-level research questions (RQs) related to QSAs for hybrid applications, followed by devising potential research directions that deserve attention from the community. Thus, our goal is to present high-level research challenges for QSAs.

To achieve the above goal, we identify four high-level RQs. The first RQ identifies research directions on whether we need to consider architectural designs for quantum software or whether classical design considerations are sufficient. The second RQ focuses more specifically on architecture-level concerns for quantum software, such as issues related to optimising architectures, integration of various components, and interactions with hardware. The third RQ gets into further details of whether the knowledge gained from the classical world can be transferred to the quantum world, such as with respect to relevant standards, architecture specification languages, and architectural patterns. Finally, the fourth RQ discusses research directions related to processes of executing hybrid applications on a diverse set of computing resources from an architectural perspective.

We organize the rest of the chapter as follows: Section 2 provides some relevant background. Section 3 discusses the overall organization of the RQs, whereas Section 4 to Section 7 discusses each RQ in detail. Finally, we conclude the chapter in Section 8.

2 Background

Since QC is a new paradigm for many members of the software architecture community, let us briefly summarise the characteristics that differ from the classical computation. How hardware is implemented varies widely among commercial vendors (even more so between research prototypes). For instance, superconducting circuits, particle traps (cold ions, neutral atoms), ion implantation into crystal lattices or optical technologies find use. Computations can be performed by applying quantum logic gates on quantum states (which is, conceptually, similar to applying classical logical gates on classical bits) or by using optimisation-like approaches (quantum annealing) that are not equivalent in their computational capabilities to the quantum gate model, yet have some advantages in terms of physical implementation [3] and practical utility [4]. Details from both approach and implementation do not just have a substantial impact on the algorithmic level [5,6], but may also influence architecturally relevant qualities [7], and can impact architecture decisions.

In particular, it holds for any currently implemented quantum computer that: (a) results are usually approximate; (b) execution times vary stochastically (even for identical programs on identical machines); and (c) algorithmic scalability depends on low-level details like the circuit generation from source code.

Likewise, quantum computers will eventually be used as (a) an *embedded component* of a classical system that acts as a computational accelerator (like, *e.g.*, Graphical Processing Units (GPUs), Tensor Processing Unit (TPUs), or Floating-Point Units (FPUs)), or (b) a *cloud resource* that is accessed via APIs. While the same distinction holds classically, hardware constraints (for instance, the need to cool chips to near-zero temperature for some approaches) will amplify differences between these two variants in a much larger way than what architects are currently accustomed to. It is not yet well understood where the exact boundaries between software architecture, software engineering and algorithmic details reside for quantum computers.

A final distinction that is not known from classical computing is between the following types of systems:

- NISQ (noisy, intermediate scale quantum computer)—these systems are commercially available, usually in a prototypical form, at the time of writing;
- PISQ (perfect (error-corrected), intermediate scale quantum computer)—such machines are fully error-corrected, large-scale QC—these may be available within years, decades, or perhaps even never, as many technological and some foundational physical problems will need to be solved;
- Special-purpose accelerators (*e.g.*, annealers, coherent Ising machines, (Gaussian) boson sampling) whose computational capabilities may not necessarily be polynomially equivalent to gate-based quantum computers.

3 Overview

QSA is still in its early stage and is primarily a less-touched research area. Therefore, in this chapter, we aim to answer four basic RQs, which we believe are relevant to the QSA community at the current stage.

In the first step, we aim to check whether it is sufficient to rely on knowledge gained from the classical world to the quantum world regarding architectural designs (RQ1, Section 4). To answer this RQ, we particularly focus on checking the current practice of accessing quantum computers since access modes have direct impacts on architectural designs. We further check whether existing modelling notations for architecture descriptions can be reused because an architectural design needs to be specified and used as a medium for facilitating communications among stakeholders. Furthermore, to enable architectural-level analyses of trade-offs of design choices, a taxonomy of quantum-specific constraints and quality attributes is expected to be proposed, which we think is worth discussing in RQ1 as well. We further elaborate on the importance of striking a balance involving low-level, quantum-specific design choices at the architectural design level. One of the challenges for software engineers to develop quantum software is a lack of knowledge in quantum mechanics, physics, etc. There is, obviously,

a practical need to raise the level of abstraction while keeping necessary details in the architectural design.

Motivated by the positive answer to RQ1, when answering RQ2 (Section 5), we first define what we mean by an *optimal architecture*. Based on the definition, we then discuss relevant architectural-level concerns. One paramount concern is about how to integrate quantum components into existing architectures. We further highlight properties of quantum hardware that have an ineligible impact on result quality, speed of calculation, degree of variability in runtime, etc. We believe these properties are essential to consider during architectural designs. Furthermore, we list a set of other considerations (e.g., the impact of the evolution of QC hardware on architectural designs) and call for contributions.

By knowing the architecture level concerns identified in RQ2, in RQ3 (Section 6), we examine whether knowledge gained from the classical world is still applicable to the quantum world to address these concerns and present the literature of existing proposals particularly targeting QSAs. Specifically, we check whether existing architecture standards are still applicable and whether new standards are required for QSAs. We further check whether existing and classical-world architecture description languages (ADLs) can be used for designing QSAs, and discuss the state-of-art ADLs proposed specifically for quantum software. We also present existing proposals on quantum software architectural patterns. Considering that quantum computers are typically accessed via the cloud, we particularly discuss quantum DevOps and quantum services.

In RQ4, we present our thoughts on a possible process of running hybrid applications on diverse computing resources. Particularly we elaborate on key roles/players in the process and their tasks.

4 RQ1: Do we need to consider QSA designs in a specialised way?

This RQ looks into whether we need to consider QSA in a specialized way as compared to the classical way or whether the classical software architecture also applies to quantum software. We discuss this from multiple dimensions below.

4.1 Accessing Quantum Computers through Cloud

In the current state of practice, quantum computers are located in cloud computing infrastructures. A user gets access to them through cloud services. Commercial players such as Amazon with its Braket platform [6] provide access to various types of quantum computers such as gate-based, ion trap-based, and neutral-atom-based quantum computers. The same trend exists for other setups, too, e.g., NordIQuEst [7] in a Nordic and Estonian infrastructure that provides access to Nordic quantum computers from Chalmers University, Sweden

[6] https://aws.amazon.com/braket/
[7] https://nordiquest.net/

and Technical Research Centre of Finland (VTT), Finland [8] through European High-Performance Computing (HPC) platforms, *i.g.*, LUMI [9] and eX3 [10]. Irrespective of quantum computer types, services provided via the cloud do not differ from a standard way of accessing other services from a technical point of view. To this end, benefiting from existing architectural design notations to capture a user's connections to various quantum computers via cloud services might be the right direction to go. In Section 6.4, we will discuss this more.

In terms of research, we foresee the need for methods with software tools to help users automatically and optimally decide which quantum computers to use from the cloud infrastructure for their specific QC applications, which will provide the best computational capability with minimum noise. One work in the literature [8] describes various characteristics of QC platforms, followed by how different quality attributes, such as availability, performance, and portability, impact these attributes (also see some details in RQ2 and RQ3). Interested readers may consult the reference [8] for more details. To build such methods, one needs to develop a comprehensive quantum software architectural framework, which captures characteristics of various types of quantum computers, such as computational power measured as quantum volume, type of technologies, and physical arrangements of qubits. Such an architectural framework can be considered the foundation of helping in optimally deploying QC applications.

In general, we believe that accessing quantum computers through High-Performance Computing (HPC) platforms will remain one of the future dominant ways for QC. Therefore, QSAs shall consider various aspects of cloud computing.

4.2 Use of Existing Notations via Extensions

One aspect to investigate is whether a QSA could be captured with existing modelling notations, e.g., UML component diagrams, by providing extensions. Examples of QC concepts include gates, qubits, superposition, entanglement, etc. Keeping our example of a UML component diagram, one could define a Unified Modeling Language (UML) profile consisting of a set of stereotypes and their associated attributes, data libraries, and constraints, which could be used to model QSAs. We feel that existing architectural design methods/notations/tools/methodologies can largely be reused.

One point, however, to consider is that quantum software is basically about making quantum circuits transpiled for a given gate model-based quantum computer for its specific gate set and other characteristics. Unfortunately, all this is too low-level, and high-level abstractions are missing. Thus, once such high-level abstractions are built, the architecture of quantum software might look quite similar to a classical one, especially if we manage to develop successful abstractions away from low-level concepts of qubits and gates. We believe that building useful abstractions is the right way for QC to be more successful, as highlighted in existing work [9].

[8] https://www.vttresearch.com/
[9] https://www.lumi-supercomputer.eu/
[10] https://www.ex3.simula.no/

One immediate area of research that one could explore is empirically evaluating existing languages used for capturing the architecture of classical software for QSA of diverse QC applications. Examples of these languages include UML, System Modeling Language (SysML), and other architecture description languages (e.g., Architecture Analysis & Design Language (AADL) and Wright [10]). With such empirical evaluation, one could assess the strengths and weaknesses of each language in terms of capturing the architecture of quantum software. Once the weaknesses have been identified, we can assess whether we can extend these languages or need entirely new ones for QSA. In Section 6.2, we discuss more about applying existing modelling notations for architectural descriptions by examining the literature.

When designing QSAs, one may need to introduce pictorial representations (e.g., similar to Feynman diagrams) of quantum computation processes into architecture documents. This may be more important than in the classical world, as quantum software development is more complex, so high-level abstraction is important, which can also provide the abstraction for quantum computation in the architecture.

Toward this direction, we foresee the need to develop novel graphical notations that could extend the existing architectural design notations. Such notations could be at different levels of abstraction depending on a user's background. For instance, if a user has no background in quantum mechanics, then the details that are irrelevant to the user may be omitted.

4.3 Need for Quantum-specific Constraints

In a classical architectural design, it is common to identify and capture constraints that are used to make trade-off decisions about design choices and perform optimisation accordingly. When making the architecture of quantum software, do we also need to capture some constraints that are specific to the quantum domain?

There are indeed constraints that are specific to the quantum domain. Taking an example of superconducting quantum computers, physical qubits can be implemented in different ways, e.g., star structure. How such qubits are implemented has a significant impact on the architectural design of the quantum software, which shall be executed on a quantum computer with a particular implementation structure.

Given that the current quantum computers are immature with an abundance of hardware errors (e.g., in qubits, gates, crosstalk), specifications of such constraints are also essential for optimising the architectural design of quantum software for a specific quantum computer. In addition, other factors for optimisation, such as limited computation resources, continuously advancing quantum technologies, and optimal ways of splitting quantum and classical tasks, need to be considered.

In terms of future research, our recommendations are as follows. First, we must design a novel taxonomy of constraints specific to quantum software architectural designs. Such a taxonomy shall be generic and applicable to various

QC applications and quantum computers. Second, the taxonomy shall also provide the provision for specific domain-specific architectural constraints. Third, there is a need to develop novel methods to specify various constraints. Fourth, we need an optimisation method that can use these constraints and additional information (e.g., information about the underlying architecture of a quantum computer) and automatically generate an optimal QSA. Towards this end, one would like to explore the novel field of quantum meta-heuristics [11]. Such meta-heuristics aim to solve optimisation problems in the Hilbert space, which will be our context. However, currently, the field is immature and needs extensive development.

One might consider that, when the chapter is written, it might be premature to develop the taxonomy because a wide range of QC hardware exists without having any of them dominant, and more are emerging. In our opinion, developing the taxonomy might help converge to a set of generic understandings (in the form of constraints) that would help perform analyses and optimisations. Moreover, with the rapid advancements in QC, new constraints are emerging, and therefore the taxonomy needs to be constantly evolved.

4.4 Classifying Quantum-specific Quality Attributes

While optimising QSAs, one needs to consider quantum-specific quality attributes. Such quality attributes could be functional or extra-functional. An example of the functional quality attribute includes circuit depth[11]. For example, if an algorithm has more than one implementation as quantum circuits, then the implementation with a lower circuit depth is preferred. In terms of extra-functional quality attributes, the generic classification, such as performance, portability, and robustness, still hold. However, these categories need to be specialized for QC.

Regarding research recommendations, we believe there is a need to build a detailed taxonomy of functional and extra-functional quality attributes. Such attributes shall cover both quantum-specific attributes and attributes from a classical domain that apply to QC. Such a taxonomy can be continuously refined with more concrete metrics as those are developed. In addition, users can use such taxonomy for their own purposes.

4.5 Prevalence of Low-level Design Choices in architectural designs

For architectural design optimisation, we must consider available resources regarding acceptable noise level, number of qubits, etc. These 'low-level' details might be needed at the very early stage of the decision-making process in contrast to the classical counterpart. As we will detail in Section 5, it is vital to strike the proper balance regarding the level of abstraction. Thus, dedicated research methodologies and software tools are needed to elicit such requirements at the earlier stage of QSA design.

[11] In the context of quantum circuits, the circuit depth represents the longest path in a given quantum circuit

5 RQ2: What are relevant architecture-level concerns for QC?

Before addressing the question of what architecture-level concerns are relevant for software systems that employ Quantum Processing Units (QPUs) and their associated custom or generic software components, we need to set our notion of an *optimal architecture*, which can be the goal of optimally tending to all the concerns we will discuss in this section.

Optimal Architecture: a collection of classical and quantum software, and their interrelations, used to solve a given computational task within the required degree of correctness, using an acceptable amount of time, and respecting necessary and unavoidable trade-offs in the use of resources.

Note while this software-centric definition is independent of quantum hardware, an optimal quantum architecture *will* strongly depend on the specific technical details of the underlying QPUs. This is certainly true in the NISQ area [3], but may likely also apply beyond if variational quantum algorithms remain a technique of choice. The latter, however, will and can only be decided once sufficiently mature hardware has become available, or variational mechanisms (and related variants like Quantum Approximate optimisation Algorithm (QAOA) or Variational Quantum Eigensolver (VQE)) will be placed on more solid grounds with improved theoretical underpinning, or both.

While it might be tempting to, from a software architecture point of view, eliminate the signature of specific properties of quantum hardware by layers of abstraction [12], the degree of influence imposed by the varying physical foundations of different QPU approaches can endanger the potential quantum advantages when a too strong degree of abstraction is tried. While we will detail this consideration below; it is important to note that quantum software architectures cannot be decoupled from quantum system architectures, and vice versa. Consequently, many architectural considerations will differ from traditional software architectural designs, and require: (a) the awareness of software architects in the first place; and (b) a better understanding of low-level details that need to be subject to the relevant academic curricula, or post-graduate industrial education efforts.

We below discuss relevant criteria that allow us to achieve optimal quantum software architectures by focusing on two major aspects: influence and relevance of hardware details on architectural properties and quantum-classical integration. We believe that such considerations will differ between essentially four domains: general-purpose computing (where cost considerations prevail), HPC (focusing on latency and throughput), embedded industrial computing (focusing on latency and hardware integration), as well as solving singular problems of outstanding importance beyond pure demonstrations of quantum advantage.

5.1 Integrating Quantum Components into Existing Architectures

In contrast, how integrating quantum components into existing architectures is less of a concern, as some of the authors, for instance, argue in Ref. [13].

Essentially, we need to distinguish between two access modes: (a) *cloud-based* (or using some other queuing systems) with *asynchronous* means of interacting with the QPU—for instance, vendors IBMQ, D-Wave, or Rigetti currently operate using this access mode; and (b) *queue local based* with *synchronous* access that can be established by either direct attachment of the QPU to a classical system via a local network (with controlled maximal latency) or even via direct on-chip integration of CPU and QPU, assuming future extended physical manufacturing and engineering capabilities.[12] This mode of operation is targeted by many HPC use cases [14] and is currently pursued by vendors like IQM or IonQ.

On the other hand, we can broadly classify known quantum algorithms alongside two characteristics: (a) Stochastic/approximate algorithms that usually require interaction with a classical system, but deliver a result that, on perfect error-corrected hardware, is either sufficiently close to a desired optimal solution or a perfect solution with a high probability—for instance, Shor's seminal factoring algorithm or Grover search [15], as do quantum annealing based approaches [3]. Alternatively, there are (b) *variational* algorithms that run a parameterised quantum circuit multiple times and optimise employed quantum gates in each run towards producing an optimal desired solution based on classical numerical optimisation techniques—for instance, QAOA [16], VQE [17,18], or many machine learning techniques like QRL [19]. Here, the number of iterations and therefore the runtime is unknown upfront, as the convergence behaviour of the underlying parameter optimisation is not yet fully understood (yet, see, *e.g.*, Refs. [20,21] for the current state of hypotheses and understanding). The situation is summarised in Table 1.

Algo/Access	Cloud/Queue	Local
Stochastic	①	②
Variational	③	④

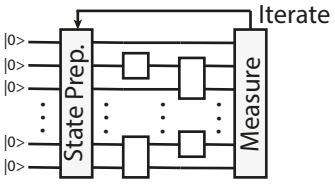

Table 1. Possible combinations of algorithmic types and QPU access modes (left), and a general interaction pattern between QPU and CPU (right).

Depending on the particular class an approach falls into, there will be different implications on architectural properties. However, once the functional and non-functional properties of a quantum component are given, the integration into any existing architecture can *in every case* be described by a single call to an API that follows the "data in, result out" pattern. This is nearly trivial from a programmer's point of view, but of course, the aspect of interest for architec-

[12] Note that for iteration-based algorithms, such details may make a difference [7], albeit we can safely ignore these in this architecture-centric overview.

ture is how the functional and non-functional properties reflect on the overall architecture.

5.2 Concerning Hardware-specific Properties in architectural designs

Quantum hardware design is very much in flux, with even individual vendors experimenting with different physical designs (*e.g.*, IBM with qubits based on superconducting Josephson junctions and silicon quantum dots). Any quantum problem, at least in the era of NISQ machines, requires a specific quantum encoding,[13] which then needs to be translated into specific physical inputs for a *particular* machines. While this seems identical to translating high-level languages into machine code at a superficial level, the properties of the translation may vary widely with the concrete physical design of the machine (see, *e.g.*, Ref. [7,22]).

Hardware and Software Coupling Properties of underlying hardware (as well as the translation process itself [22,7]) influence result quality, speed of calculation, the ability to find a result at all, the range of input data that can be processed, and degree of variability in runtime. As these properties directly arise as a consequence of:

- the degree and type of noise/loss on the capability of individual qubits to store data and the influence of decoherence that leads to increasing loss of quantum information with deeper circuits/longer programs;
- the connectivity between qubits (*i.g.*, which qubits can be subjected to joint operations, and which not) and imperfections arising from the coupling structure;
- the ability to prepare initial states for computation, and intermittently "clear" qubits for re-use;
- imperfections of the measurement process that turns quantum information into classical results such that they can be interpreted and processed by classical IT.

These aspects must be considered in designs that strive for optimal architecture for a given use case. Especially for variational algorithms, there is a direct trade-off between longer circuits/larger number of iterations (that can be shown to lead to monotonically increasing result quality for some approaches like QAOA) and the consequences like the increasing influence of noise, which in turn, deteriorates result quality.

[13] It is doubtful if it will be possible to produce perfect error-corrected machines for each imaginable use case; we expect noisy machines to remain relevant also in the long run if it will be possible to achieve quantum advantage with them, albeit this remains an open scientific question [3].

Hardware-specific Optimisation *Logical* circuits (the primary output of a translation/compilation from a high-level description in any quantum framework into an executable sequence of operations on quantum states) are, in principle, independent of the executing QPU. But they are already *implicitly* connected with the target hardware by the choice of gate set, which is used to manipulate individual or multiple qubits. When a logical circuit is transpiled to a concrete physical implementation, logical gates must be replaced by (combinations of) available native hardware gates, and this influences architectural properties like scalability, runtime, result quality, and so forth. There is optimisation potential by designing hardware-optimised algorithms [3] that center around the operations available in specific implementations, thus eliminiating some of the aforementioned overhead.

However, this does not solve (and is independent of) the connectivity problem: Since not every qubit can interact with each other qubit, any necessary operations between "remote" qubits must be enabled by first bringing the qubits into close enough proximity, and then execute the gate. (Logically) moving qubits around is possible with so-called swap gates that exchange the quantum state of two qubits (without requiring a measurement that would destroy the quantum state). As, in turn, native swap gates are not available on most architectures, it needs to be replaced by a sequence of three controlled-not gates, increasing circuit depths even further. Enhancing the connectivity of the underlying systems, possibly in a problem-specific way, can provide improvements, albeit at the cost of co-designing hardware and software. Likewise, technologies like cold neutral atoms provide universal connectivity between qubits—albeit under the negative influence by prolonged gate execution times [23] as compared top other implementation platforms, which especially deteriorates the temporal performance of iterative schemes ③, ④.

Some parts of applications could therefore be optimised for different types of quantum technologies. However, it remains a question whether doing so is commercially viable and whether the arising disadvantages (for instance, having to synchronise two queues in the case of cloud access ①, ③) defeat the advantages gained by a multi-QPU scheme.

Consequences As we have discussed, low-level properties of QPUs have a direct (and often unaccustomed to the classical world) impact on architecture-level properties. We, therefore, believe that: (a) the awareness of this relation is available to software architects, and (b) mapping between the architectural quality and low-level properties need to be defined and maintained in any architectural design document or model, of course satisfying established requirements on traceability [24].

Additionally, we expect that not the scalability of algorithms is the most relevant criterion, but the absolute performance for a given set of inputs; many industrial use cases (e.g., factory control, production planning, long-term resource distribution [4]) work on input data sets of fixed (or merely varying)

sizes, whereas absolute execution times are the most relevant concern. Similar considerations apply to HPC workloads.

5.3 Other Considerations

The properties of QC hardware are subject to considerable flux, and even many of the elementary theoretical characteristics (like runtime!) of algorithms themselves are not yet well understood. This induces uncertainties in the design of QSAs. We find that more research is desirable and necessary to understand such variability's short- and long-term implications.

- **Evolution of QC Hardware:** At this stage of QC development, we might also need to consider the fast evolution of QC technologies and the architectural design needs to particularly take care of changes of underlying QC technologies. To this end, architecture decay issues that quantum applications might need to face need to be addressed.
- **Empirical Studies:** To perform architectural design optimisation, empirical studies are needed to collect evidence. Currently, any possibility to compare structured approaches to architectural design with the classical world is limited, which is, however, also caused by the fact that no large-scale applications of QC are known.
- **Verification of Quantum Results:** It may be a challenge to verify the results of quantum computations depending on the class of algorithms [25,26] or to ascertain the correct functioning of QPUs [27]. This concern should also be considered at the architectural level, as it may influence correctness, performance, and quality properties.
- **Deployment of QC in Real-time/Safety-critical Contexts:** Especially in scenario ① (and to a lesser degree in ②), it is impossible to specify upper bounds on the execution time of a computation. To integrate such applications into industrial control scenarios, likewise, it is known that tail latencies, even if they are rare, can cause substantial issues in data processing schemes [28]. Latency is also known to be among the most important performance characteristic in HPC scheduling [29]. Appropriate countermeasures (*e.g.*, fallback to safe states, classical refinement of approximate results, latency tailing) must be considered directly at the architectural level.

6 RQ3: Is knowledge gained from the classical world still applicable to the quantum world, and to which extent?

This question is raised to examine whether we can reuse methodologies, tools, and standards, among others, created for developing classical software for developing quantum software and to which extent they can be reused. This question is crucial as we aim to save as much effort as possible.

6.1 Standards

The ISO/IEC/IEEE 42010:2022 (Software, systems and enterprise — Architecture description) standard[14] is a well-known and commonly practised standard for devising architecture descriptions. In this standard, terminologies such as architecture, architecture description (AD), architecture description framework (ADF), architecture description language (ADL), architecture view and viewpoints, stakeholders and concerns are defined. The standard also defines their relationships. For instance, *concern* is linked to a *stakeholder* describing her/his concern on the relevance or importance of a matter. Since these concepts are defined at a very high level, which is intended by most standards, we believe the conceptual model of the ISO/IEC/IEEE 42010:2022 is still applicable to the quantum world.

However, we like to mention that bringing concepts from the quantum world will be useful when instantiating the conceptual model. For instance, in addition to typical viewpoints such as operational, logical, technical, and deployment viewpoints, it might be important to bring integration viewpoints where interactions of the classical world and quantum world meet for developing hybrid software applications, as we discussed in Section 5.1. Also, a deployment viewpoint might be tailored to consider different computing resources, as some might be quantum computers of the same or different types. Another example is that we might need quantum-specific modelling solutions to describe architecture views. We will discuss this in detail in Section 6.2.

In summary, existing standards are still generally applicable to guide the development of QSAs. Methodologies and tools conforming to these standards might need to be customised for developing quantum and hybrid applications to enhance their applicability and usability.

6.2 Architecture Description Languages

Different modelling notations (e.g., UML, SysML) can be used to describe an architecture. For instance, shortly after UML was standardised by the Object Management Group (OMG), Rich Hilliard [30] investigated the application of UML in the context of IEEE P1471, which was superseded by ISO/IEC/IEEE 42010 later on and concluded that UML provides a set of notations that can be used to model various architecture aspects of systems.

In QSA, attention has been naturally drawn to UML. For instance, the authors of the short paper [31] suggested distinguishing UML model elements in terms of whether they carry quantum information. Such a suggestion is necessary; however, it is far from being sufficient for specifying architectures of quantum software. In [32], the authors presented novel ideas for developing quantum software modelling languages along with a conceptual model of quantum programs. They illustrated how to model the state-based behaviour of quantum programs. In [33], the authors presented a position paper arguing the potential

[14] ISO/IEC/IEEE 42010:2022: https://www.iso.org/standard/74393.html

benefits of applying Model-Driven Engineering (MDE) practices for developing software. Specifically, the authors highlighted that QC resources are mostly accessed via cloud services (*e.g.*, Amazon Braket) and therefore, current modelling languages for cloud computing might be worth being investigated.

Representing processes by graphical abstractions is not uncommon in quantum physics; for instance, Feynman diagrams [34] are widely used representations of certain mathematical aspects of quantum electrodynamics and related theories. However, they are in one-to-one correspondence with calculations, and do not provide an abstraction in the software engineering sense, but instead make complex formulas easier accessible to intuition. Quantum circuit diagrams, which are also commonly employed to visualise quantum algorithms, reside at a higher level of abstraction from a physical point of view, yet correspond to the layer of assembly language from a computer science point of view. This is also well below the usual levels of abstractions as they appear in software architecture and engineering. Approaches like quantum pictorialism, as proposed and detailed by Coecke and Kissinger [35], might pave a a road that bridges between physics and computer science via visual abstractions. Yet, none of the existing solutions seems sufficient to model the architecture of any non-trivial software application with quantum components.

Moreover, we observe that to develop a practical-useful ADF and ADL, we first need access to real-world quantum applications, which are hardly available to researchers at the moment. Existing modelling notations (e.g., UML) and paradigms (e.g., MDA) can be a natural foundation for developing quantum-specific ADFs and ADLs. We do not foresee the need for radically new modelling notations for now, which is partially due to our limited knowledge of real-world quantum applications.

6.3 Architectural Patterns

Architectural patterns (e.g., client-server patterns) represent ideas, practices, and solutions that are reusable for addressing reoccurring problems. By definition, this is still applicable to QSA.

Various patterns and pattern languages have been proposed in the literature for supporting quantum software development and testing. For instance, Leymann [36] proposed a template for structuring pattern documents, including fields such as name, intent, problem statement, solution and known use. With this template, Leymann proposed an initial set of basic patterns for developing quantum algorithms, such as *Initialization* (aka *State Preparation*) to answer the question of how the input of a quantum register can be initialized, and *Creating Entanglement*, and *Phase Shift*. However, as stated clearly by the author, these patterns are design (not architectural) patterns for developing quantum algorithms.

Weigold et al. [37] also proposed six encoding patterns to facilitate efficient data loading of quantum algorithms; three of the six patterns are for data encoding and state preparation (e.g., preparing an arbitrary state), and two of them

are for unitary transformation (e.g., matrix encoding), and one measurement pattern (i.e., post-selective measurement).

In [38], Zhao et al. presented a set of bug patterns ("erroneous code idioms or bad coding practices" in [38]) such as *Insufficient Initial Qubits*, along with a classification of them, in the context of developing quantum programs with Qiskit.

However, it is worth noting that these existing works focus on identifying patterns at the design and implementation levels, not on the architectural level. Consequently, the reported design and bug patterns are all quantum-specific. As discussed in Section 5.1 and Section 5.2, we see the need to define patterns for, for instance, various modes of integrating quantum components.

6.4 Quantum DevOps and Quantum Services

Inspired by the agile software development principle, DevOps was proposed to shorten the software development lifecycle and enable continuous delivery. In principle, being a set of recommended practices, DevOps does not put restrictions on using any specific architectural styles. However, the microservices architecture is often applied together with DevOps for developing continuously deployed software systems, which is composed of small services communicating via APIs and the cloud. From the architectural aspect, the microservices architecture is distributed and loosely coupled to facilitate fault isolation, continuous and fast integration of new features, etc. For instance, Azure DevOps[15] is a platform that implements DevOps with a set of cloud-hosted services for building, testing and deploying programs in various programming languages on multiple operating platforms.

To get access to QC resources, the cloud is mainly used for two purposes: 1) getting access to classical computing resources for the purpose of simulating quantum computers, such as the early stage of validation of quantum software can be achieved; 2) getting access to limited and shared quantum computers that are often placed in very specialised room conditions. Therefore, a number of cloud-based QC platforms (e.g., Amazon Braket[16], D-Wave Leap[17] and Xanadu's Quantum Cloud[18]) have been made available for use via the cloud.

Therefore, DevOps for QC emerged. For instance, Azure Quantum proposes two DevOps loops[19]: the outer loop (which is a complete DevOps cycle) and the inner loop, which is quantum-specific and involves activities (e.g., programming quantum software, executing quantum software on simulators or quantum computers, estimating required computing resources) that need to be performed by quantum architects and engineers. In the literature, Gheorghe-Pop et al. [39]

[15] Azure DevOps: https://azure.microsoft.com/en-us/products/devops/
[16] Amazon Braket, https://aws.amazon.com/braket/
[17] D-Wave Leap: https://cloud.dwavesys.com/leap/login/?next=/leap/
[18] Xanadu Cloud: https://www.xanadu.ai/
[19] DevOps for QC at Azure Quantum: https://learn.microsoft.com/en-us/azure/architecture/guide/quantum/devops-for-quantum-computing

promoted the concept of *Quantum DevOps* in the context of dealing with uncertainties of NISQ computers and envisioned building Quantum DevOps by extending the traditional DevOps concept. To enable the realisation of this concept, a set of quantum-specific software engineering methodologies (including tool support), across several key phases of a software development lifecycle, such as requirements engineering, design and testing are needed, are needed but are largely unavailable up to date, as also highlighted in [9].

As one of the key elements of DevOps, Gheorghe-Pop et al. [39] mentioned that, for testing quantum software to be run on NISQ computers, it is needed to go through three testing phases: testing in a simulator without noise, testing in a simulator with injected noise mimicking real Qubit environment, and testing on real quantum computers. In the literature, a set of approaches (e.g., [40,41,42,43]) on testing quantum software have been proposed. However, to realise Quantum DevOps in the NISQ era or beyond, scalable and efficient testing approaches across the three quantum software testing phases are expected.

To conclude, currently, the concept of Quantum DevOps seems promising; however, supporting its realisation, methodologies and tools across various software development life cycles need to be proposed or matured. Moreover, investigations on whether the microservices architecture is still applicable for Quantum DevOps are needed as well.

In addition to the idea of Quantum DevOps, back in 2004, the authors of [44] promoted to introduce a *layered software architecture* to support the development of quantum algorithms by following a four-phase design flow going from a high-level representation of a quantum algorithm all the way to a technology-specific implementation, through a series of transformations. Recently, Moguel et al. [45] investigated opportunities and challenges of introducing the Service-oriented Computing (SOA) architectural style to the quantum world in the context of developing classical-quantum hybrid software systems, where quantum software systems are invoked as *quantum services*. The proposal comes naturally because most quantum computers can currently be accessed through the cloud (Section 5.1). The authors experimented with Amazon Braket by wrapping quantum services as classical services and demonstrated the potential. But, the authors also concluded that, based on their investigation, there is an insufficient benefit for running quantum algorithms as classical services, and new methodologies and tools are needed, which we also agree.

7 RQ4: How to run hybrid applications from an architectural perspective?

As discussed in Section 7, there are various ways of getting access to quantum computers, and some properties of quantum hardware (e.g., scalability, runtime, result quality) are impacted by the degree and type of noise, connectivity of qubits, etc. of quantum hardware. All these aspects have an influence on achieving the optimal architecture we envision. However, before achieving such an optimal architecture for running quantum software on quantum hardware,

a common practice is using quantum simulators such that quantum programs can be run and tested before being executed on quantum hardware. We envision that in the NISQ era, architecture optimisation (Section 5 might need to rely on quantum computer simulators of various types, including simulators simulating pure qubits and various hardware noises.

There are two possible ways of performing a simulation: (1) using a quantum simulator (with the fundamental limits being about 45 qubits for noise gate-based simulation and 10,000 qubits for noisy quantum annealing) and carefully extrapolating results to larger computing resources; (2) using an effective model, based on the structure of a quantum circuit, imperfections of gates, speed of decoherence, etc., to predict most likely performance/result quality characteristics.

Moreover, it is important to identify QSA that cover both classical and quantum components of software systems, including, for instance, user interfaces (or APIs), databases, and computation parts. Submitting a computation request to quantum hardware needs a series of steps starting from data pre-processing. When results are ready, they will also need to be processed to be usable. This process is particularly commonly seen for systems running Artificial Intelligence (AI). Currently, AI is executed on traditional machines that send a specific part of the computation to GPUs. Developers only need to develop their AI algorithms, and a dedicated software or platform is responsible for orchestrating the computation between the CPU and GPU.

It is not yet possible to have a hybrid and seamless architecture that covers both the quantum and classical worlds. Developers must manually decide what part of the computation needs quantum hardware and which needs classical ones. Commonly, developers manually send input data to quantum hardware, and when results are ready, they fetch the results and continue the elaboration or processing with other classical machines. We foresee the need for an inte-

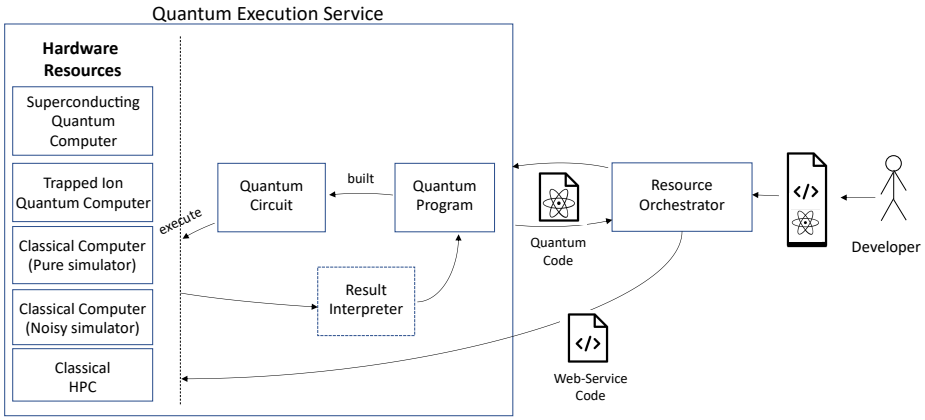

Fig. 1. Running Hybrid Applications—An Architectural Perspective

grated quantum and classical computing architecture. A process mainly has the following steps (as also illustrated in Figure 1):

- **Developers** develop quantum and classical code to implement their systems. We foresee the possibility of developing a framework, with which developers can specify which part of the computation must be sent to quantum computers and which part must be computed with classical hardware resources.
- **Resource Orchestrator** automatically orchestrates and therefore sends computation tasks to quantum and/or classical hardware resources.
- **Resource Orchestrator** sends the quantum program to the quantum program compiler, which compiles the program and executes it on quantum hardware, which could be identified by the orchestrator or manually.
- **Result Interpreter** receives results once the execution of the quantum program is concluded. Hardware noise is an important aspect to consider in this step. The Result Interpreter should be able to understand if there is the need to run the program once more, based on a pre-defined confidence level, for instance.

Let's consider a practical example to illustrate the process. A weather forecast company might decide to use QC to provide a web service reporting the weather forecast in certain areas. The developers will develop the code for the weather forecast station composed of four main steps: 1) Obtain sensor data; 2) Process the data and send the pre-processed data to the QC algorithm that has been developed; 3) Interpret data returned by the quantum algorithm; and 4) Save results in a database accessible via the web service.

With this process, we envision that software development effort will be reduced as the splitting of tasks and allocating them to different resources can be, in an ideal situation, automatically performed by *Resource Orchestrator*. We are aware that currently, there is no such an approach, and developers need to manually code each step, manually interpret results, and then again manually send them to an accessible database. In the far future, we foresee the option of adopting QC algorithm seamlessly with a process like the one we sketched.

8 Conclusion

Quantum software architecture (QSA) of quantum computing (QC) applications that require both classical and QC resources is an understudied area of research. Nonetheless, within quantum software engineering, researchers and practitioners are getting more and more interested in it. Due to this reason, the objective of this chapter was to provide an initial set of research challenges and opportunities for QSA after examining a set of research questions and also put light on future research directions. These discussions are preliminary and need to be further enhanced with more thorough investigations, such as systematic literature reviews, empirical evaluations, and surveys with industry and researchers.

References

1. M. Weigold, J. Barzen, F. Leymann, and D. Vietz, "Patterns for hybrid quantum algorithms," in *Service-Oriented Computing*, J. Barzen, Ed. Cham: Springer International Publishing, 2021, pp. 34–51.
2. A. Callison and N. Chancellor, "Hybrid quantum-classical algorithms in the noisy intermediate-scale quantum era and beyond," *Phys. Rev. A*, vol. 106, p. 010101, Jul 2022. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.106.010101
3. K. Bharti, A. Cervera-Lierta, T. H. Kyaw, T. Haug, S. Alperin-Lea, A. Anand, M. Degroote, H. Heimonen, J. S. Kottmann, T. Menke, W.-K. Mok, S. Sim, L.-C. Kwek, and A. Aspuru-Guzik, "Noisy intermediate-scale quantum algorithms," *Rev. Mod. Phys.*, vol. 94, p. 015004, Feb 2022. [Online]. Available: https://link.aps.org/doi/10.1103/RevModPhys.94.015004
4. A. Bayerstadler, G. Becquin, J. Binder, T. Botter, H. Ehm, T. Ehmer, M. Erdmann, N. Gaus, P. Harbach, M. Hess, J. Klepsch, M. Leib, S. Luber, A. Luckow, M. Mansky, W. Mauerer, F. Neukart, C. Niedermeier, L. Palackal, R. Pfeiffer, C. Polenz, J. Sepulveda, T. Sievers, B. Standen, M. Streif, T. Strohm, C. Utschig-Utschig, D. Volz, H. Weiss, and F. Winter, "Industry quantum computing applications," *EPJ Quantum Technology*, vol. 8, no. 1, 11 2021. [Online]. Available: https://epjquantumtechnology.springeropen.com/track/pdf/10.1140/epjqt/s40507-021-00114-x.pdf
5. T. Gabor, S. Zielinski, S. Feld, C. Roch, C. Seidel, F. Neukart, I. Galter, W. Mauerer, and C. Linnhoff-Popien, "Assessing solution quality of 3sat on a quantum annealing platform," *Proc. Int. Workshop on Quantum Technology and Optimization Problems (QTOP)*, 2019. [Online]. Available: https://arxiv.org/abs/1902.04703
6. M. Schönberger, S. Scherzinger, and W. Mauerer, "Ready to leap (by co-design)? join order optimisation on quantum hardware," in *Proceedings of ACM SIGMOD/PODS International Conference on Management of Data*, 2023.
7. K. Wintersperger, H. Safi, and W. Mauerer, "Qpu-system co-design for quantum hpc accelerators," in *Proceedings of the 35th GI/ITG International Conference on the Architecture of Computing Systems*. Gesellschaft für Informatik, 8 2022.
8. B. Sodhi, "Quality attributes on quantum computing platforms," *ArXiv*, vol. abs/1803.07407, 2018.
9. S. Ali, T. Yue, and R. Abreu, "When software engineering meets quantum computing," *Communications of the ACM*, vol. 65, no. 4, pp. 84–88, 2022.
10. H. Yao and Y. Ma, "An exploration for the software architecture description language of wright," *ICIC Express Letters*, vol. 8, pp. 3481–3487, 12 2014.
11. Z. A. Dahi and E. Alba, "Metaheuristics on quantum computers: Inspiration, simulation and real execution," *Future Generation Computer Systems*, vol. 130, pp. 164–180, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X21004969
12. M. Schönberger, M. Franz, S. Scherzinger, and W. Mauerer, "Peel — pile? cross-framework portability of quantum software," *QSA@IEEE International Conference on Software Architecture (ICSA)*, 2022.
13. T. Krüger and W. Mauerer, "Quantum annealing-based software components: An experimental case study with sat solving," *Q-SE@ICSE*, 2020. [Online]. Available: https://arxiv.org/abs/2005.05465
14. T. S. Humble, A. McCaskey, D. I. Lyakh, M. Gowrishankar, A. Frisch, and T. Monz, "Quantum computers for high-performance computing," *IEEE Micro*, vol. 41, no. 05, pp. 15–23, sep 2021.

15. M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2011.
16. E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," Nov. 2014.
17. A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'Brien, "A variational eigenvalue solver on a photonic quantum processor," *Nature Communications*, vol. 5, no. 1, p. 4213, Jul 2014. [Online]. Available: https://doi.org/10.1038/ncomms5213
18. J. Tilly, H. Chen, S. Cao, D. Picozzi, K. Setia, Y. Li, E. Grant, L. Wossnig, I. Rungger, G. H. Booth, and J. Tennyson, "The variational quantum eigensolver: A review of methods and best practices," *Physics Reports*, vol. 986, pp. 1–128, 2022, the Variational Quantum Eigensolver: a review of methods and best practices. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0370157322003118
19. M. Franz, L. Wolf, M. Periyasamy, C. Ufrecht, D. Scherer, A. Plinge, C. Mutschler, and W. Mauerer, "Uncovering instabilities in variational-quantum deep q-networks," *Journal of The Franklin Institute*, 8 2022.
20. L. Zhou, S.-T. Wang, S. Choi, H. Pichler, and M. D. Lukin, "Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices," *Phys. Rev. X*, vol. 10, p. 021067, Jun 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevX.10.021067
21. M. Streif and M. Leib, "Training the quantum approximate optimization algorithm without access to a quantum processing unit," *Quantum Science and Technology*, vol. 5, no. 3, p. 034008, may 2020. [Online]. Available: https://dx.doi.org/10.1088/2058-9565/ab8c2b
22. S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, "t—ket⟩: a retargetable compiler for nisq devices," *Quantum Science and Technology*, vol. 6, no. 1, p. 014003, nov 2020. [Online]. Available: https://dx.doi.org/10.1088/2058-9565/ab8e92
23. L. Henriet, L. Beguin, A. Signoles, T. Lahaye, A. Browaeys, G.-O. Reymond, and C. Jurczak, "Quantum computing with neutral atoms," *Quantum*, vol. 4, p. 327, Sep. 2020. [Online]. Available: https://doi.org/10.22331/q-2020-09-21-327
24. L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, ser. SEI series in software engineering. Addison-Wesley, 2003. [Online]. Available: http://books.google.fi/books?id=mdiIu8Kk1WMC
25. S. Dasgupta and T. S. Humble, "Characterizing the reproducibility of noisy quantum circuits," *Entropy*, vol. 24, no. 2, 2022. [Online]. Available: https://www.mdpi.com/1099-4300/24/2/244
26. W. Mauerer and S. Scherzinger, "1-2-3 reproducibility for quantum software experiments," *Q-SANER@IEEE International Conference on Software Analysis, Evolution and Reengineering*, 2022.
27. H. Pashayan, J. J. Wallman, and S. D. Bartlett, "Estimating outcome probabilities of quantum circuits using quasiprobabilities," *Phys. Rev. Lett.*, vol. 115, p. 070501, Aug 2015. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.115.070501
28. J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, pp. 74–80, 2013. [Online]. Available: http://cacm.acm.org/magazines/2013/2/160173-the-tail-at-scale/fulltext
29. A. Reuther, C. Byun, W. Arcand, D. Bestor, B. Bergeron, M. Hubbell, M. Jones, P. Michaleas, A. Prout, A. Rosa, and J. Kepner, "Scalable system scheduling for

hpc and big data," *Journal of Parallel and Distributed Computing*, vol. 111, pp. 76–92, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731517301983

30. R. Hilliard, "Using the uml for architectural description," in *International Conference on the Unified Modeling Language*. Springer, 1999, pp. 32–48.
31. C. A. Pérez-Delgado and H. G. Perez-Gonzalez, "Towards a quantum software modeling language," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 2020, pp. 442–444.
32. S. Ali and T. Yue, "Modeling quantum programs: Challenges, initial results, and research directions," in *Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software*, ser. APEQS 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 14–21. [Online]. Available: https://doi.org/10.1145/3412451.3428499
33. F. Gemeinhardt, A. Garmendia, and M. Wimmer, "Towards model-driven quantum software engineering," in *2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE)*. IEEE, 2021, pp. 13–15.
34. R. P. Feynman, "Space-time approach to quantum electrodynamics," *Phys. Rev.*, vol. 76, pp. 769–789, Sep 1949. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRev.76.769
35. B. Coecke and A. Kissinger, *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, 2017.
36. F. Leymann, "Towards a pattern language for quantum algorithms," in *International Workshop on Quantum Technology and Optimization Problems*. Springer, 2019, pp. 218–230.
37. M. Weigold, J. Barzen, F. Leymann, and M. Salm, "Encoding patterns for quantum algorithms," *IET Quantum Communication*, vol. 2, no. 4, pp. 141–152, 2021.
38. P. Zhao, J. Zhao, and L. Ma, "Identifying bug patterns in quantum programs," in *2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE)*. IEEE, 2021, pp. 16–21.
39. I.-D. Gheorghe-Pop, N. Tcholtchev, T. Ritter, and M. Hauswirth, "Quantum devops: Towards reliable and applicable nisq quantum computing," in *2020 IEEE Globecom Workshops (GC Wkshps*. IEEE, 2020, pp. 1–6.
40. X. Wang, P. Arcaini, T. Yue, and S. Ali, "Generating failing test suites for quantum programs with search," in *Search-Based Software Engineering*, U.-M. O'Reilly and X. Devroey, Eds. Cham: Springer International Publishing, 2021, pp. 9–25.
41. S. Ali, P. Arcaini, X. Wang, and T. Yue, "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)*, 2021, pp. 13–23.
42. X. Wang, P. Arcaini, T. Yue, and S. Ali, "Application of combinatorial testing to quantum programs," in *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2021, pp. 179–188.
43. E. Mendiluze, S. Ali, P. Arcaini, and T. Yue, "Muskit: A mutation analysis tool for quantum software testing," in *The 36th IEEE/ACM International Conference on Automated Software Engineering, Tool Demonstration*. IEEE/ACM, 2021.
44. K. Svore, A. Cross, A. Aho, I. Chuang, and I. Markov, "Toward a software architecture for quantum computing design tools," in *Proceedings of the 2nd International Workshop on Quantum Programming Languages (QPL)*, 2004, pp. 145–162.

45. E. Moguel, J. Rojo, D. Valencia, J. Berrocal, J. Garcia-Alonso, and J. M. Murillo, "Quantum service-oriented computing: current landscape and challenges," *Software Quality Journal*, pp. 1–20, 2022.