Quantum Software and its Engineering

Michael Epping[*], Michael Felderer[*†], Vlad Gheorghiu[‡§], Tao Yue[¶]
[*]German Aerospace Center (DLR), Institute of Software Technology, Cologne, Germany
[†]University of Cologne, Cologne, Germany
[‡]softwareQ Inc., Waterloo, Ontario, N2L 0C7, Canada
[§]Institute for Quantum Computing, University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada
[¶]School of Computer Science and Engineering, Beihang University, Beijing, China

***Abstract*—Quantum computing is an exciting new computing paradigm that promises significant advantages for many hard problems. As the field moves from theoretical considerations to practical applications, the need for high-quality software increases. More and more software developers are becoming involved in this exciting development. This brief primer provides an overview of quantum software, starting with the basic ideas of quantum computing and delving into the fundamental concepts of quantum programming, the role of quantum algorithms, and the tools needed to develop quantum applications. It serves as an introduction to the IEEE Software special issue on *Quantum Software and its Engineering*. Particular attention is given to combinatorial optimization as a particularly promising application, new developments in programming, and aspects of quality assurance and software engineering.**

***Index Terms*—Quantum Computing, Quantum Software Engineering**

I. Introduction

Quantum computing (QC) is gaining considerable attention from industry and academia alike. It has been making great progress in recent years and quantum computers are increasingly available to a larger community. And it is foreseeable that the technology will become established for specific use cases in optimization, machine learning or material simulation. Researchers and companies are looking into how QC can benefit them. To meet such expectations, not only quantum hardware but also software is required. Quantum software is a key enabler for QC [1], just as (classical) software is for classical computing. Quantum software covers algorithms, system software, application software, entire ecosystems, and hybrid systems. State-of-the-art quantum software is closer to the system than software for classical software systems. For instance, new compilers are needed that allow to transform programs into sequences of elementary operations ("quantum circuits") which match the specific limitations and opportunities of a given quantum hardware platform.

Quantum software also requires the adoption of software engineering concepts and partially also novel methods and techniques for software analysis, design, implementation, testing, deployment and maintenance. It is imperative to transfer concepts of software engineering to quantum and hybrid settings early on in the development and field test them to tackle the complexity of quantum software and ensure its reliability. There is active research on all phases of the quantum software lifecycle and many open questions remain.

Structure: Section II introduces QC. Section III presents key areas of this special issue along with challenges and opportunities. Section IV concludes the article.

II. Basics of Quantum Computing

QC was first introduced by Richard Feynman in 1982, who proposed using quantum systems to simulate physical phenomena that are computationally intractable for classical computers. Feynman's original focus was on simulating quantum mechanical systems, rather than tackling general computational problems. A pivotal breakthrough occurred in 1994 when Peter Shor developed Shor's algorithm capable of efficiently factoring large integers and solving discrete logarithms in large Abelian groups, which form the basis of most modern public-key cryptographic systems. This marked the first example of an exponentially faster quantum algorithm. Meanwhile, Lov Grover introduced Grover's algorithm, which provides a quadratic speedup for unstructured search problems and Boolean predicate evaluations. For several years, QC was considered conceptually promising but practically infeasible. A major challenge arose from the "No Cloning Theorem," which prevents cloning unknown quantum states, leading to doubts about the feasibility of quantum error correction (QEC). However, in 1995, Peter Shor showed that robust QEC codes exist, which sparked intense research into quantum information theory and fault-tolerant QC.

Today, we are at the threshold of a new computing era. Current quantum devices, i.e. Noisy Intermediate-Scale Quantum (NISQ) computers, contain between 100 to 1,000 qubits. These systems are difficult to simulate with conventional computers, so they must be studied experimentally. Although large-scale, fault-tolerant quantum computers capable of running Shor's algorithm remain a distant goal, NISQ devices offer a testbed for investigating potential applications.

A quantum computer consists of a collection of qubits, which can be controlled and manipulated with a degree of precision constrained by inherent noise and error rates. A qubit is a two-level quantum system, often exemplified by the spin states of a spin-1/2 particle. The evolution of qubits is governed by the Schrödinger equation, meaning operations applied to qubits must be unitary.

A typical quantum computation involves three key stages:

- Initialization: All qubits are set to a defined initial state, typically denoted as $|0\rangle$.

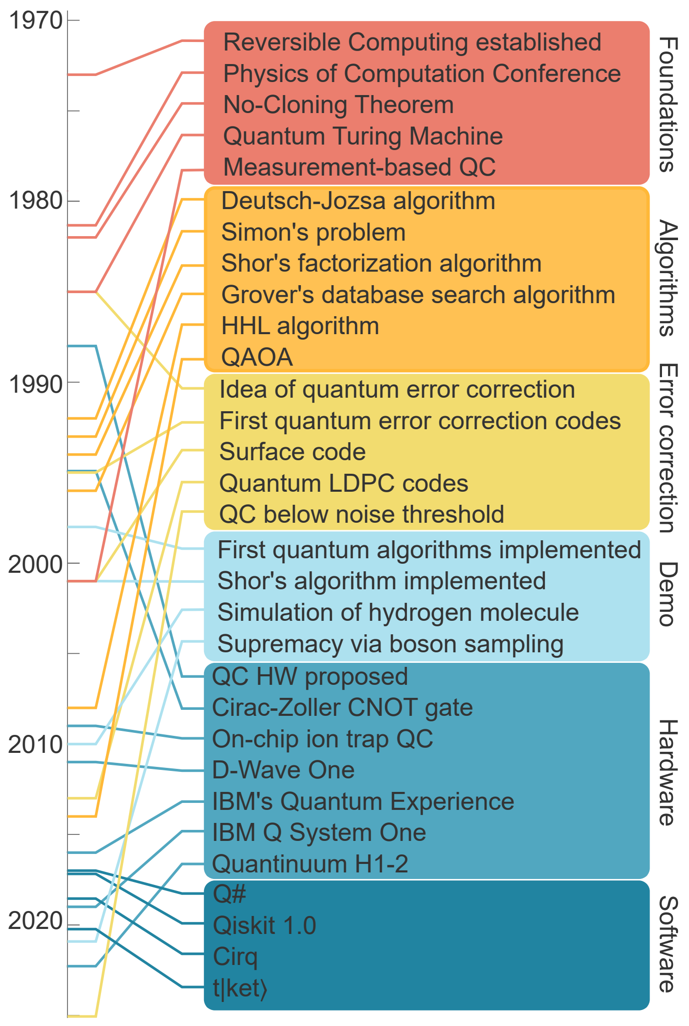

Fig. 1. (Side box) A selective timeline of QC. The theoretical foundations were laid in the 70's and 80's. Most well-known algorithms were designed in the 90's, followed by proof-of-principle implementations. Despite significant progress made in the 1980s and 1990s, error correction is only now becoming more relevant in practice. This is in part due to hardware development, which really took off in the 2010s. Consequently, the demand for programming tools and software engineering has steadily increased since the late 2010s. For larger-scale quantum software developments, advances in this area are now essential.

- Unitary Evolution: The qubits undergo a sequence of unitary operations, represented by a unitary matrix U.
- Measurement: The final state of the qubits is measured, collapsing the quantum state into classical results. This process may be repeated multiple times with results post-processed classically, as is often done in hybrid algorithms like the Variational Quantum Eigensolver.

The three primary principles that underlie QC are as follows.

- Superposition: A qubit can exist in a complex linear combination of its basis states, $|0\rangle$ and $|1\rangle$.
- Entanglement: Quantum entanglement is a phenomenon in which qubits become correlated in ways that are impossible with classical systems. Measurements of entangled qubits exhibit correlations that cannot be explained by shared randomness alone.
- Interference: Quantum states can interfere constructively or destructively, allowing algorithms to amplify correct solutions while cancelling out incorrect ones.

Unlike classical probabilistic systems, which operate with real-valued probabilities, the amplitudes describing quantum states are complex numbers. Consequently, quantum states are represented in a Hilbert space L^2, as opposed to the normed vector spaces L^1 that represent probabilistic classical computing. This seemingly subtle mathematical distinction is fundamental to the computational power of quantum systems.

III. Areas of Quantum Software and its Engineering

Engineering quantum software needs to systematically design, develop, test, and maintain quantum software, which, unlike engineering classical software, must account for quantum-specific challenges [2], such as dealing with probabilistic outcomes, handling quantum hardware constraints at the early stage, considering the inevitable demand on integration with classical systems, effectively dealing with noise-imposed restrictions on circuit depth and algorithm scalability, etc.

In this special issue, we present ten articles that advance four pivotal domains in quantum software and its engineering: quantum optimization (Section III-A), quantum programming (Section III-B), quality assurance (Section III-C), and classical software engineering (CSE) for quantum (Section III-D).

A. Quantum Optimization

Solving optimisation problems is one of the most promising applications of QC for the near future. In particular, combinatorial optimisation problems are often easy to describe but hard to solve, allowing the potential of quantum computers to be exploited without the need to transfer large amounts of data to or from the quantum computer, two bottlenecks that can negate the advantage of quantum computers for other applications. Another useful feature of combinatorial optimisation problems is the ability to formulate many different applications in the same mathematical framework. Reducing the degree of the cost polynomial by introducing slack variables and replacing constraints with penalty terms leads to quadratic unconstrained binary optimization (QUBO) problems. Software tools exist to perform these transformations, including the open source libraries quark and quapps, which are described by E. Lobe et al. in this issue [3]. The resulting form can be solved directly on a quantum annealer or on a gate-based quantum computer using the quantum approximate optimisation algorithm (QAOA). This allows practitioners to focus on their specific optimisation problem without having to worry about the tedious work of the necessary transformations.

Many applications that benefit from the approach described can be found in all industries and fields, including logistics, financial modelling, machine learning and materials science. And many examples can be found already in the vast literature on solving optimisation problems with quantum computers. Here we would like to highlight two such applications: Vehicle Route Optimisation and Test Case Minimisation.

Good solutions to hard vehicle routing problems, including the ride pooling problem, promise huge cost savings and a positive impact on the climate and the environment. They can be formulated as QUBO problems as well. This allows B. Rosendo et al. to investigate the prospects of using QAOA and quantum annealing for such problems [4]. It is highly recommended reading for anyone who wants to understand the possibilities and potential of QC using a practical example.

Another hard optimisation problem that can be formulated as a QUBO is test case minimisation. Reducing the number of tests in a test suite while still satisfying all testing objectives has many practical benefits, such as faster testing, reduced resource requirements, and improved maintainability, to name a few. BQTmizer, a tool for exploiting quantum annealers for test case minimisation, is introduced and validated by M. Roth et al. in this issue [5]. This can be a good entry point for many readers because they can relate to the problem that is solved.

Besides advertising the exciting potential of quantum computers for solving optimization problems, we also consider it appropriate to point out the challenges to the reader who is new to this topic. One notable obstacle is the problem of Barren plateaus, where the optimization landscape becomes flat as the system scales, making it increasingly difficult to find optimal solutions using the QAOA, for example. This issue needs to be addressed when trying to solve large instances that arise in economically relevant optimization problems. Moreover, despite great advances, quantum hardware remains in its infancy, with qubit decoherence, gate errors, and limited qubit numbers posing additional barriers.

Future research in quantum optimization is rich with possibilities. More and more practical case studies showcasing experience with quantum algorithms for relevant use-cases will follow. At the same time advances in algorithm design, in particular hybrid quantum-classical approaches, as well as tailored QEC methods, will allow more efficient use of scarce quantum resources. Further progress on the scalability of quantum hardware, combined with new developments in quantum software engineering (QSE), will push the boundary of instance sizes that can be tackled with quantum algorithms for combinatorial optimization, such that industrially relevant problem sizes become feasible. Therefore quantum optimization will have a major impact and it is a good candidate for a significant quantum advantage in the near future.

B. Quantum Programming

Quantum programming is a broad term encompassing all processes that enable interaction with QC hardware or facilitate specific tasks for quantum computers. These tasks include but are not limited to *quantum compiling*, where a quantum algorithm is transformed into an equivalent quantum circuit executable on specific hardware, and *quantum circuit optimization*, an additional compilation pass that reduces gate counts while preserving the circuit's computational equivalence [6]. However, quantum software extends far beyond just compiling and optimization to *quantum simulation*, where classical computers emulate quantum systems to study their behaviour, *quantum machine learning*, which explores hybrid quantum-classical models for data analysis and optimization; *QEC*, which focuses on mitigating noise in quantum computations, and *quantum resource estimation* [7], which assesses hardware requirements for executing quantum algorithms efficiently.

A decade ago, quantum software was in its infancy. Today, it has evolved into a rich ecosystem with a wide array of (open-source) quantum programming tools. These range from quantum compilers and simulators to QEC frameworks, quantum programming languages, and cloud-based QC platforms. Given the extensive number of available tools, a comprehensive list can be found, e.g., in the following resources, the list being non-exhaustive: https://quantiki.org/wiki/list-qc-simulators, https://github.com/desireevl/awesome-quantum-computing, and https://github.com/qosf/awesome-quantum-software.

The current special issue includes two articles on quantum programming.

In [8] J. Maletic et al. apply software engineering techniques to quantum programming by introducing methodologies for static program analysis and refactoring. They extend the pre-existing software infrastructure (srcML) to support OpenQASM, enabling quantum program exploration and manipulation. The research aims to address various challenges in quantum programming by leveraging static analysis for potential future tooling.

M. Roth et al. introduce sQUlearn [9], a NISQ-compatible Python library designed for quantum machine learning (QML). It integrates seamlessly with classical ML tools like scikit-learn and supports both quantum kernel methods and quantum neural networks. The library provides automated execution handling, customizable data encoding, and specialized regularization techniques. With support for Qiskit and PennyLane, it facilitates easy transitions between quantum simulation and physical hardware execution.

C. Quality Assurance

Software quality assurance, as a systematic process to ensure that software products meet predefined quality standards and fulfill customer requirements, involves code reviews, testing and the adherence to best practices or standards, etc. In QSE, significant attention has been paid to validate and verify quantum software. For instance, Murillo et al. [10] recently summarized key QSE areas, among which quality assurance received the highest number of publications.

This issue contains three articles that fall into quality assurance. One is about benchmarking hybrid classical-quantum system analyzability. The second is about algorithm-hardware co-design strategies enabling quantum polar code deployment on superconducting processors. The third work employs machine learning models to predict temporal error rate variations in cloud-based quantum systems. These three articles address quantum software quality assurance through distinct but complementary approaches.

Hybrid quantum software is currently the dominant trend as NISQ-era devices lack error correction, have limited qubits, and require classical systems for control, optimization and error mitigation. With this observation in mind, Diaz et al. [11] presented a list of properties and metrics, along with tool support, to facilitate the evaluation of the analyzability of hybrid classical-quantum software. Here, the analyzability is measured with metrics such as classical cyclomatic complexity, quantum cyclomatic complexity, method size, circuit depth, and auxiliary qubits. By demonstrating these metrics with a concrete example and providing tool support, their work marks an initial effort to systematize quality assessment in hybrid quantum-classical software.

Quantum polar codes, a class of quantum error-correcting codes inspired by classical channel polarization, mitigate noise in quantum systems (e.g., decoherence, gate errors) to enable fault-tolerant computation. Our special issue article from Kurniawan et al. [12] demonstrated a full-stack implementation unifying noise-aware quantum compilation with polar code integration. This framework spans hardware-optimized circuit synthesis (state preparation, gate transpilation) and classical decoding software for syndrome extraction and error correction, bridging design with superconducting processor constraints to advance scalable quantum error resilience.

Current noise-aware circuit compilation methods likely depend on calibration data that could become obsolete due to the high error variability in quantum systems themselves, as well as delays caused by cloud-based quantum job queues. Motivated by this observation, Rodríguez-Soriano et al. [13] proposed a novel methodology to predict temporal variations in error rates for superconducting quantum processors in the context of cloud-based quantum systems. The methodology, equipped with five different machine learning models, optimizes qubit mapping and routing by selecting the least noisy qubits and gates based on the anticipated conditions at the time of circuit execution. The authors also conducted an empirical study to evaluate the proposed methodology, on a simulator and a quantum hardware. Based on the results of the empirical study, the best machine learning models were recommended.

The literature presents numerous quantum software validation methods, including adapted classical techniques such as metamorphic testing and search-based testing. This special issue's three articles described above further contribute by advances in quality assessment and circuit-level error suppression. To accelerate progress toward reliable quantum software, future work need to develop standardized benchmarks, curate open datasets, and foster cross-disciplinary integration.

D. *Classical Software Engineering for Quantum Computing*

Building practical QC applications requires the implementation of quantum algorithms as software. Learning from the classical computing realm, developing dependable software entails following a software development life cycle, which typically includes requirements engineering, architecture and design, development, testing, debugging, and maintenance phases [2]. In addition, QC can benefit from classical system-level software concepts like compilers as well as classical simulation concepts.

Liu et al. [14] presented an approach to quantum compiler optimization, which is essential in the NISQ era for the practical feasibility of QC. This is the case because quantum gate synthesis and mapping often increase circuit size and errors, making performance enhancement crucial. Mainstream compilers like Qiskit, Tket, and Cirq typically use fixed or user-defined optimization techniques to execute strategies, which may not fully exploit their potential. The authors proposed an automated tuning method for quantum compilers based on the Double Dueling Deep Q-Network, framing the sequence of optimization techniques as a decision-making problem with the aim to reduce the number of quantum gates. Experimental results show that the method outperforms Qiskit. The approach provides a novel pathway for optimizing quantum compilers, effectively enhancing circuit optimization performance.

Velmurugan et al. [15] presented an approach to fast classical simulation of qubit-qudit hybrid systems. Simulating quantum circuits is a computationally intensive task that relies heavily on tensor products and matrix multiplications, which can be inefficient. Recent advancements eliminate the need for tensor products and matrix multiplications, offering significant improvements in efficiency and parallelization. Extending these optimizations, the authors adopt a block simulation methodology applicable to qubit-qudit hybrid systems, which interprets the state vector as a collection of blocks and applies gates without computing the entire circuit unitary. The method utilizes this block-simulation method, thereby gaining major improvements over state-of-the-art simulators for simulating multi-level quantum systems with various benchmark circuits.

IV. CONCLUSION

In this article we outlined the key concepts, milestones and challenges in quantum software and it's engineering. We teased various exciting topics along the quantum software lifecycle, from programming and simulation to analysis and testing, and along the quantum software stack, from the application to compilation and error correction. To realize the full potential of quantum computers, significant advances in the engineering of quantum software are mandated. Promising applications will require complex software that manages resources at various levels of abstraction, interacts with classical computing power, and must use advanced methods to leverage quantum computers available in the medium term. Without strong tool support for software development, we risk quickly reaching our limits here.

This is necessarily an interdisciplinary challenge, for which physicists describe the phenomena of quantum physics, computer scientists contribute their expertise and experience with conventional computers, engineers tackle immense technical challenges and domain experts contribute the applications to make the development worthwhile. With this in mind, we invite readers to take a closer look at the topics described. The sheer scale of the endeavor opens up many opportunities and there is a great need for bright minds, so please get involved!

V. AUTHOR INFORMATION

Michael Epping leads an interdisciplinary QC research group at the Institute of Software Technology at the German Aerospace Center (DLR) with a focus on software and research to make DLR's quantum computers as effective as possible. This requires developing the quantum software

stack, taking into account topics like error mitigation and optimized compilation. Michael Epping is a physicist with a strong background in quantum information theory. His previous research has contributed to the development of the foundations of quantum networks and the understanding of quantum phenomena such as Bell nonlocality.

Michael Felderer is the Director of the Institute of Software Technology at the German Aerospace Center (DLR) and a full professor at the University of Cologne, Germany. His research interests include software and systems engineering for digital twin, artificial intelligence (AI), and quantum technologies, software quality assurance, model-based software engineering, and research software engineering. Michael Felderer has coauthored more than 150 publications and received 14 best-paper awards. He is recognized by the Journal of Systems and Software (JSS) as one of the twenty most active established software engineering researchers worldwide in the period 2013 to 2020.

Vlad Gheorghiu is the CEO and President of softwareQ Inc., a quantum software and quantum information processing company in Waterloo, ON Canada. Vlad is also an Affiliate of the Institute for Quantum Computing at the University of Waterloo, ON Canada, working on quantum information and computation. Vlad holds a PhD in Theoretical Physics from Carnegie Mellon University, USA. His current interests lie in quantum software and quantum architectures, quantum-resistant cryptography, QEC, quantum cryptanalysis and resource estimation for realistic implementations of quantum algorithms, entanglement theory, as well as applications of machine learning techniques to the quantum domain. Vlad has more than 20 years of experience in quantum information and computation, and more than 1800 citations in peer-reviewed articles.

Tao Yue is a full professor at Beihang University, China. Her research interests include both classical and quantum software engineering. She has co-authored more than 170 publications and is the Editor in Chief of the Continuous Special Section of Quantum Software Engineering at TOSEM.

Acknowledgments

ME acknowledges funding by the DLR Quantum Computing Initiative via project ALQU and the QuantERA project EQUIP via DFG grant 491784278. VG acknowledges support from softwareQ Inc. TY acknowledges support from the State Key Laboratory of Complex & Critical Software Environment (SKLCCSE, grant No. CCSE-2024ZX-01).

References

[1] I. Exman et al., *Quantum Software: Aspects of Theory and System Design*. Springer Nature, 2024.
[2] 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.
[3] E. Lobe et al., "Quantum optimization applications with quark & quapps." in this issue.
[4] B. Rosendo et al., "Quantum approaches for vehicle routing optimization on nisq platforms." in this issue.
[5] X. Wang et al., "Bqtmizer: A tool for test case minimization with quantum annealing." in this issue.
[6] A. Javadi-Abhari et al., "Quantum computing with qiskit," arxiv:2405.08810.
[7] V. Gheorghiu et al., "Quantum resource estimation for large scale quantum algorithms," *Future Generation Computer Systems*, vol. 162, p. 107480, 2025.
[8] J. Maletic et al., "Static analysis and transformation for quantum programming languages." in this issue.
[9] M. Roth et al., "squlearn – a python library for quantum machine learning." in this issue.
[10] J. M. Murillo et al., "Quantum software engineering: Roadmap and challenges ahead," *ACM Trans. Softw. Eng. Methodol.*, 2025.
[11] A. Diaz et al., "Environment for the evaluation of hybrid (classical-quantum) systems analyzability." in this issue.
[12] H. Kurniawan et al., "Implementing quantum polar codes in a superconducting processor: From state preparation to decoding." in this issue.
[13] L. Rodríguez-Soriano et al., "Predicting the temporal variability of error rates in superconducting quantum processors." in this issue.
[14] Y. Liu et al., "Quantum compiler optimization automated tuning via d3qn." in this issue.
[15] A. Saha et al., "Fast classical simulation of qubit-qudit hybrid systems." in this issue.