Embracing Iterations in Quantum Software: A Vision

Arif Ali Khan
University of Oulu
Oulu, Finland
arif.khan@oulu.fi

Mahdi Fahmideh
University of Southern Queensland
Brisbane, Australia
mahdi.fahmideh@usq.edu.au

Aakash Ahmad
Lancaster University Leipzig
Leipzig, Germany
a.ahmad13@lancaster.ac.uk

Muhammad Waseem
Wuhan University
Wuhan, China
m.waseem@whu.edu.cn

Mahmood Niazi
King Fahd University of Petroleum and Minerals, and Interdisciplinary Research Centre for Intelligent Secure Systems
Dhahran, Saudi-Arabia
mkniazi@kfupm.edu.sa

Valtteri Lahtinen
Quanscient
Tampere, Finland
valtteri.lahtinen@quanscient.com

Tommi Mikkonen
University of Jyväskylä
Jyväskylä, Finland
tommi.j.mikkonen@jyu.fi

ABSTRACT

In today's software engineering, iterations, affordable en masse, form an important part of just about any system. However, not all computing resources are cheap to consume. In High-Performance (HPC) and Quantum Computing (QC), executions can consume considerable amounts of energy and time, which is reserved and used even if the very first steps in the process fail. This means that developers must assume a different attitude towards programming, and aim at error-free software before its execution. This is commonly facilitated using simulators, which are commonplace for both HPC and QC. However the fashion developers advance from one tool to another is ad-hoc, with no established software engineering guidelines, and the final step from simulators to HPC/QC is still a leap of faith, comparable to releasing software.

In this paper, we propose a vision where developers can iterate in an agile fashion when developing quantum software. The iterations are defined such that when the solution is still vague in the beginning, computations are interactive and provide instant feedback, thus supporting conceptualization of the software and experimenting with new ideas. When the solution becomes more precise, more expensive computations such as quantum algorithm and hyperparameter optimization are executed in batches.

CCS CONCEPTS

• **Software and its engineering** → **Software development techniques**; **Software prototyping**;

KEYWORDS

Quantum software engineering, Quantum Computing, Quantum IDE

ACM Reference Format:
Arif Ali Khan, Mahdi Fahmideh, Aakash Ahmad, Muhammad Waseem, Mahmood Niazi, Valtteri Lahtinen, and Tommi Mikkonen. 2022. Embracing Iterations in Quantum Software: A Vision. In *Proceedings of the 1st International Workshop on Quantum Programming for Software Engineering (QP4SE '22), November 18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3549036.3562057

1 INTRODUCTION

The field of Quantum Computing (QC) has lately received growing research, industry, and policy interest to an ever-increasing extent. The disruptive solutions empowered with QC have opened the way for business development in various areas, such as cybersecurity, financial services, and aerospace. On the other hand, the aspect of development to achieve cost-effective and reliable quantum software applications has been slow. Solving this problem is critical to enabling QC to live up to its promising potential.

The word quantum in *Quantum Computing* implies quantum mechanics that the system uses to compute the output. In physics, quantum is the smallest unit of any physical entity, generally referring to atomic or subatomic particles, e.g., electrons, neutrons, and photons. QC is a fast-growing research field expected to bring revolution in various industrial areas [30]. QC uses quantum mechanics concepts to process the information and complete specific tasks much faster than classical computers. Quantum physics characteristics such as superposition, entanglement, and quantum interference are applied for computing purposes. Different studies have been published focusing on developing quantum algorithms [7, 12, 20]. Quantum computers outperform in two different ways as compared to classical computers:

- the problem needs large amounts of parallel computing, e.g. big data analysis [23, 30], data security [30], encryption [21], and machine learning [3];
- the problem requires precise and effective computation in natural science e.g. physics [4], material science ([11], and chemistry [18].

Analogically, these resemble the fashion of how GPUs and other special computing units accelerate computations. However, there is one extra twist in QC: computational results are typically statistical, and several executions are typically needed. Moreover, to make matters worse, the results are flaky because the same input can produce different outputs, some of which are true results and some false positives and false negatives. Hence, running them takes time and interpreting the results typically requires additional computations using classical software [24]. Moreover, many conventional tools such as debuggers and step-by-step execution render useless in QC development due to the statistical nature of quantum technologies.

This paper presents a vision for embracing iterations when developing QS. However, unlike present-day iterations, QS iterations are of increasing complexity, starting with the developer workstation, continuing in the cloud, running simulations with High-Performance Computing (HPC), and finally reaching the Quantum Computer. The vision is aligned with the Academy of Finland's recently accepted research proposal *Developer Experience in Iterative Quantum Software Engineering* (DEQSE, 2022-2026), where such a development approach will be experimented.

2 BACKGROUND AND MOTIVATION

Because of the flaky characteristics of QC, software developers need novel techniques, tools, processes, and methods that explicitly focus on developing software systems based on quantum mechanics. In particular, designing a quantum software algorithm is challenging because of fundamental quantum mechanics characteristics, including superposition and entanglement. New principles and methodologies for quantum software design are strongly demanded as the design is the most critical phase of developing the QS systems [30].

Following the conventional wisdom in software programming, which started from hardware-focused, hard-wired techniques in the 1950s and then evolved into today's agile, iterative development, Quantum Software Engineering (QSE) should eventually become agile, iterative, and incremental [19, 22].

However, conventional tools and methods may need to be examined to ensure that they meet the unique characteristics of quantum software [30]. Therefore, software developers face significant challenges when coding quantum programs due to switching to an entirely different programming mindset with counter-intuitive quantum principles [21]. While producing the actual code remains broadly similar to recent programming languages and libraries, testing and debugging require reconsideration, as discussed below.

Testing quantum programs requires addressing the following aspects:

- Defining and checking test oracles, since the state of a quantum program can be in superposition and reading the precise state in superposition is a challenge, and
- Efficient quantum test data generation, since potential values of quantum variables will be exponentially higher than classical variables.

Defects identified during the testing phase must be located, isolated, and patched; thus, debugging is needed. Developing effective debugging solutions must overcome the following three challenges [13]:

- When observations or simulations are available, quantum states are generally high dimensional and difficult to interpret, limiting their usefulness in debugging
- No evidence or guide exists for where and what to check when debugging quantum programs.
- Examining values of quantum variables in superposition is hard.

While one cannot redefine how actual quantum computers work, the situation can be considerably eased by offering tools and techniques for experimentation early on and, in general, agile development of quantum programs. Moreover, such tools enable an approach where the role of quantum computations focuses on the essentials, whereas any boilerplate part of programs is dealt with classical software.

For classical software, many problems related to the development, testing and debugging are dealt with by iterative development. This has been manifested for the last 20 years – ever since the rise of agile methods, development cycles have become shorter and shorter, up to the point where several releases are made per day [8]. Indeed, agile, iterative development has numerous positive outcomes for developers. Instead of waiting for weeks or months for bug reports from acceptance testing, they get feedback in a matter of seconds or minutes when a project is finally completed. This supports their flow state, which is an essential ingredient in modern-day software development [16]. Adopting an iterative approach for developing a QS system is the best choice in the present scenario [22]. We do not need to wait until the QSE techniques get stable or refined but develop them in parallel with the QSE evolution [22].

To summarize, iterations help find the bugs and other problems as early as possible because, at that point, the developers can fix them with straightforward actions. Moreover, since computing resources are generally cheap, iterations are affordable en masse. However, not all computing resources are reasonably cheap, as in HPC and QC, and executions can consume considerable energy and time. This means developers must assume a different attitude towards programming and aim at error-free software before its execution. This is commonly done using simulators, which are commonplace for both HPC and QC. However, the final step from simulators to HPC/QC is still a leap of faith, comparable to releasing software, which at that point is expected to be error-free.

3 THE VISION: IDE FOR A QUANTUM SOFTWARE DEVELOPER

Based on QC characteristics, our vision is to enable iterative development of quantum programs, with decreasing degree of interactivity and increasing degree of quantum-related infrastructure in the process: At the early stage of development, the developer uses a simulator to get interactive feedback instantly, which enables rapid REPL (Read-Eval-Print Loop [27]) cycles early on, similarly to the

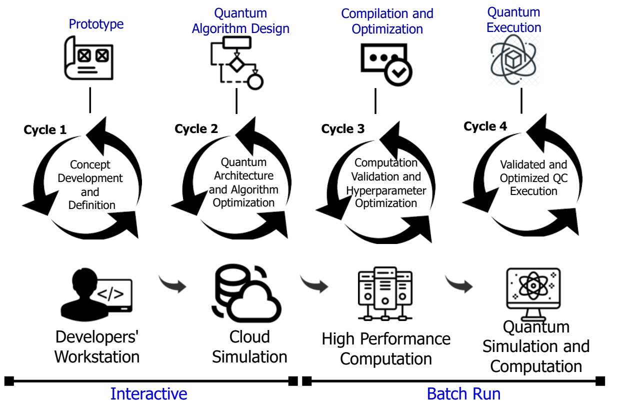
Prototype
Quantum Algorithm Design
Compilation and Optimization
Quantum Execution
Cycle 1
Concept Development and Definition
Cycle 2
Quantum Architecture and Algorithm Optimization
Cycle 3
Computation Validation and Hyperparameter Optimization
Cycle 4
Validated and Optimized QC Execution
Developers' Workstation
Cloud Simulation
High Performance Computation
Quantum Simulation and Computation
Interactive
Batch Run

Figure 1: Cycles and infrastructure for interactive and iterative development of quantum programs.

design of classical software. Then, as the program evolves, new infrastructure will be made available to enable the design and optimization of the quantum parts of the software. Moreover, one of the challenges pointed out by developers is that nowadays, the current IDE support is inadequate for quantum programming, hindering the development tasks [6]. This is an important step in programming in the quantum context because every qubit matters. In contrast, in classical computing, Moore's law has eliminated size-related problems from an average developer [30]. Finally, when everything is verified using HPC, an accurate QC is involved, using the most appropriate quantum algorithm and hyperparameters that guide its execution. In other words, the development starts with a simple implementation that can be run on a single computer, then advances to more complex systems using cloud and HPC, and finally reaches a true quantum computer.

The research will focus on understanding the quantum software development using four iterative cycles (Figure 1) inspired from [10], covering the following steps:

- **Cycle 1**: Concept development and definition using the developer's computer. The techniques we applied include software prototyping [5] and end-user programming [14]. Tool support such as wizards and help menus that help an inexperienced developer forward will be added so that the developer needs not to worry about boiler-plate code. As executions happen on a workstation, development is interactive.
- **Cycle 2**: Quantum architecture and algorithm optimization in a cloud environment. The techniques in this phase include adaptive mutation in genetic algorithms [17] and model/program transformations [29] to test different configurations and create some level of flakiness. Both fully automated and developer-supported tools will be considered and experimented with. The goal is to support both the quantum program and related infrastructure development. Our goal is to use Techila Distributed Computing Engine[1] for parallelizing the executions, resulting in near real-time interactive development without the complexity of traditional high-performance computing.

[1]http://www.techilatechnologies.com/

- **Cycle 3**: Computation validation and hyperparameter optimization in the HPC environment. Here, we rely on using existing techniques but adding a monitoring and rollback option to them to cancel executions as soon as it is clear that the simulation fails. The purpose is to enable faster recovery in case of errors. Here, we rely on existing resources simply because running applications typically becomes batch-oriented instead of interactive in HPC.
- **Cycle 4**: Live QC, using a real quantum computer.

Within each of these levels, iterations of different natures are available; in a personal computer, iterative work is typically rapid, but programs must be simple. In an actual QC, iterative development is next to impossible, but the full power of QC is available. In the intermediate levels, it is an open problem how iterations should be used for the best possible developer experience.

4 IMPLEMENTATION CONSIDERATIONS AND DISCUSSION

The bulk of the technical work needed for reaching the vision is composing classical software, including designing and implementing interfaces, tools, testware, and monitoring systems, such as dashboards. We plan to implement an Eclipse plugin that, on the one hand, provides a programming interface to the developers and, on the other hand, provides a connection to simulators and QC. The implementation will be based on an existing plugin PyDev[2], and it will be extended with features that connect it to different simulators, HPC and QC. To simplify using various infrastructures, we plan to implement a deployment pipeline and a monitoring dashboard to support the development experience similar to modern-day continuous software engineering facilities [9]. As the baseline, we will use QisKit[3] [28]. This pipeline will collect data from the development process, allowing us to analyze how developers behave while developing quantum programs.

As for designing quantum programs, our intuition is that programming a quantum computer is similar to algorithm design in general. Much of the learning from this field can be reused in the context of QC. However, some of the tasks in QC fall beyond algorithm design, including, in particular, the design and implementation of the subsystems that deal with statistical and flaky outcomes and validating the results. Hence, while it is known that heterogeneity is harmful to algorithm design in general [2], we expect that aspect of the work will require significant attention simply because it is expected that several different designs for QC will be proposed.

A feasible baseline design would be that we need an equivalent of Hardware Abstraction Layer (HAL), which we call Quantum Abstraction Layer (QAL), which will form the core of the interactive executions. However, as designing such QAL would require significant investment in design, we are initially taking a shortcut and relying on results presented already at [10, 26]. The approach relies on making quantum software accessible in a fashion where API gateways are used to interact with quantum computers. These gateways are similar in nature to any other gateway used on the Web, thus letting the developers enter the familiar territory in terms

[2]https://www.pydev.org/
[3]https://qiskit.org/

of infrastructure, with quantum computers being the backend only, using real-time containers as the underlying technology [25]. This also allows distribution to the cloud on a large scale. Moreover, different simulators can be embedded in containers as needed, following the observations of [1, 15].

Finally, the bulk of the technical work needed for reaching the vision is composing classical software, including designing and implementing interfaces, tools, testware, and monitoring systems (e.g., dashboards). In addition, the infrastructure includes data collection features that allow tracking of how developers use it to understand. Once we reach the level of a prototype implementation, we can start observing which resources the developers address during the development and deployment of quantum software and refine and redesign some of them based on developer feedback. In the long run, the goal is to shape the practice of quantum software engineering so that general guideline can be established.

5 CONCLUSIONS

The advantages of the iterative software development to mitigate risks and uncertainties have been widely acknowledged in the software engineering literature. Iterative development helps in spotting potential defects early in the process.

In this vision paper, we propose helping developers to iterate in the development of HPC and QC software. The iterations are defined such that in the beginning, when the solution is still vague, computations are interactive and provide instant feedback. When the solution becomes more precise, more expensive computations, executed in batches, are needed. Finally, when everything is verified using HPC, a real QC is involved, at a point when the algorithm and its hyperparameters are as precise as possible. It is acknowledged that to some extent, quantum programming resembles classical algorithmic design, but here also related infrastructure for interpreting and validating the results is being built in parallel as part of the effort.

Once our implementation advances, we plan to study the developer experience across all the phases to understand the applicability of the model. In the long run, we aim at consolidating guidelines for concrete development of quantum software in an iterative fashion, with iterations having different flavor at different points of the development.

ACKNOWLEDGMENTS

This work has been partially supported by the Academy of Finland (project 349945).

REFERENCES

[1] Aakash Ahmad, Arif Ali Khan, Muhammad Waseem, Mahdi Fahmideh, and Tommi Mikkonen. 2022. Towards Process Centered Architecting for Quantum Software Systems. In *Proceedings of the 1st International Conference on Quantum Software (QSW)*. IEEE, 26–31.

[2] Olivier Beaumont, Vincent Boudet, Arnaud Legrand, Fabrice Rastello, and Yves Robert. 2000. Heterogeneity considered harmful to algorithm designers. In *Proceedings IEEE International Conference on Cluster Computing. CLUSTER 2000*. Citeseer, 403–403.

[3] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. 2017. Quantum machine learning. *Nature* 549, 7671 (2017), 195–202.

[4] Andrew M Childs, Dmitri Maslov, Yunseong Nam, Neil J Ross, and Yuan Su. 2018. Toward the first quantum simulation with quantum speedup. *Proceedings of the National Academy of Sciences* 115, 38 (2018), 9456–9461.

[5] Alan M Davis. 1995. Software prototyping. In *Advances in computers*. Vol. 40. Elsevier, 39–63.

[6] Manuel De Stefano, Fabiano Pecorelli, Dario Di Nucci, Fabio Palomba, and Andrea De Lucia. 2022. Software engineering for quantum programming: How far are we? *Journal of Systems and Software* 190 (2022), 111326.

[7] David Deutsch. 1985. Quantum theory, the Church–Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 400, 1818 (1985), 97–117.

[8] Dror G Feitelson, Eitan Frachtenberg, and Kent L Beck. 2013. Development and deployment at Facebook. *IEEE Internet Computing* 17, 4 (2013), 8–17.

[9] Brian Fitzgerald and Klaas-Jan Stol. 2017. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software* 123 (2017), 176–189.

[10] Jose Garcia-Alonso, Javier Rojo, David Valencia, Enrique Moguel, Javier Berrocal, and Juan Manuel Murillo. 2021. Quantum software as a service through a quantum API gateway. *IEEE Internet Computing* 26, 1 (2021), 34–41.

[11] Harper R Grimsley, Sophia E Economou, Edwin Barnes, and Nicholas J Mayhall. 2019. An adaptive variational algorithm for exact molecular simulations on a quantum computer. *Nature communications* 10, 1 (2019), 1–9.

[12] Lov K Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 212–219.

[13] Yipeng Huang and Margaret Martonosi. 2019. Statistical assertions for validating patterns and finding bugs in quantum programs. In *Proceedings of the 46th International Symposium on Computer Architecture*. 541–553.

[14] Capers Jones. 1995. End user programming. *Computer* 28, 9 (1995), 68–70.

[15] Arif Ali Khan, Aakash Ahmad, Muhammad Waseem, Peng Liang, Mahdi Fahmideh, Tommi Mikkonen, and Pekka Abrahamsson. 2022. Software Architecture for Quantum Computing Systems-A Systematic Review. *arXiv preprint arXiv:2202.05505* (2022).

[16] Kati Kuusinen, Helen Petrie, Fabian Fagerholm, and Tommi Mikkonen. 2016. Flow, Intrinsic Motivation, and Developer Experience in Software Engineering. In *Agile Processes, in Software Engineering, and Extreme Programming*, Helen Sharp and Tracy Hall (Eds.). Springer International Publishing, Cham, 104–117.

[17] Stefano Marsili Libelli and P Alba. 2000. Adaptive mutation in genetic algorithms. *Soft computing* 4, 2 (2000), 76–80.

[18] Sam McArdle, Suguru Endo, Alán Aspuru-Guzik, Simon C Benjamin, and Xiao Yuan. 2020. Quantum computational chemistry. *Reviews of Modern Physics* 92, 1 (2020), 015003.

[19] Enrique Moguel, Javier Berrocal, José García-Alonso, and Juan Manuel Murillo. 2020. A Roadmap for Quantum Software Engineering: Applying the Lessons Learned from the Classics.. In *Q-SET@ QCE*. 5–13.

[20] Ashley Montanaro. 2016. Quantum algorithms: an overview. *npj Quantum Information* 2, 1 (2016), 1–8.

[21] Michele Mosca, Martin Roetteler, and Peter Selinger. 2019. Quantum Programming Languages (Dagstuhl Seminar 18381). In *Dagstuhl Reports*, Vol. 8. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[22] Mario Piattini, Manuel Serrano, Ricardo Perez-Castillo, Guido Petersen, and Jose Luis Hevia. 2021. Toward a quantum software engineering. *IT Professional* 23, 1 (2021), 62–66.

[23] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. 2014. Quantum support vector machine for big data classification. *Physical review letters* 113, 13 (2014), 130503.

[24] Amr Sabry. 2003. Modeling quantum computing in Haskell. In *Proceedings of the 2003 ACM SIGPLAN workshop on Haskell*. 39–49.

[25] Kilian Telschig, Andreas Schönberger, and Alexander Knapp. 2018. A real-time container architecture for dependable distributed embedded applications. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*. IEEE, 1367–1374.

[26] David Valencia, Jose Garcia-Alonso, Javier Rojo, Enrique Moguel, Javier Berrocal, and Juan Manuel Murillo. 2021. Hybrid classical-quantum software services systems: Exploration of the rough edges. In *International Conference on the Quality of Information and Communications Technology*. Springer, 225–238.

[27] L Thomas Van Binsbergen, Mauricio Verano Merino, Pierre Jeanjean, Tijs Van Der Storm, Benoit Combemale, and Olivier Barais. 2020. A principled approach to REPL interpreters. In *Proceedings of the 2020 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. 84–100.

[28] Robert Wille, Rod Van Meter, and Yehuda Naveh. 2019. IBM's Qiskit tool chain: Working with and developing for real quantum computers. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1234–1240.

[29] Manuel Wimmer, Michael Strommer, Horst Kargl, and Gerhard Kramler. 2007. Towards model transformation generation by-example. In *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*. IEEE, 285b–285b.

[30] Jianjun Zhao. 2020. Quantum software engineering: Landscapes and horizons. *arXiv preprint arXiv:2007.07047* (2020).