QBugs: A Collection of Reproducible Bugs in Quantum Algorithms and a Supporting Infrastructure to Enable Controlled Quantum Software Testing and Debugging Experiments

José Campos†, André Souto†§
†LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal, and §Instituto de Telecomunicações, Lisboa, Portugal
jcmcampos@fc.ul.pt, ansouto@fc.ul.pt

***Abstract*—Reproducibility and comparability of empirical results are at the core tenet of the scientific method in any scientific field. To ease reproducibility of empirical studies, several benchmarks in software engineering research, such as Defects4J, have been developed and widely used. For quantum software engineering research, however, no benchmark has been established yet. In this position paper, we propose a new benchmark—named QBugs—which will provide *experimental subjects* and an experimental infrastructure to ease the evaluation of new research and the reproducibility of previously published results on quantum software engineering.**

***Index Terms*—Software engineering, Quantum software testing, Software bugs**

I. Problem Statement

Despite an endless number of fields where quantum computing could overpass traditional computing (e.g., factoring numbers [1], perform unstructured search [2], optimization algorithms [3]), solve linear equations [4]), quantum software—as *classic* software—needs to be specified, developed, and tested by human developers. On one hand, these tasks have been well studied by researchers and innumerous approaches have been proposed and evaluated in *classic* software. On the other hand, in the quantum world, the execution of these tasks implies novel research in several directions as well as the development of new approaches and prototypes [5]. Quantum software engineering research is still in its early days, as it is the empirical evaluation of novel approaches in quantum computing. To begin with

> *How would one evaluate their own novel quantum software engineering research or, reproduce others research on quantum software engineering?*

In its classic counterpart, research ideas and prototypes have been evaluated and reproduced through *controlled* empirical experiments, which usually require:

- *Experimental subjects*: software artifacts in the form of source code, test cases, documentation, bug reports, and commit history.
- *Experimental infrastructure*: configuration files, scripts, and tools.

To ease the burden, to some extent, of (1) finding representative and diverse *experimental subjects*, (2) (re-)developing an *experimental infrastructure*, and (3) reproducing others research, several databases of software artifacts have been developed [6, 7, 8, 9, 10].

Empirical experiments in quantum software engineering are still at its infancy [5] and, to the best of our knowledge, no database of quantum software artifacts have been presented. The lack of reusable artifacts and datasets on quantum software engineering may hold the research community back and reduce confidence in empirical results. In fact, we (community) are already witnessing the same *reproducibility* issue we have been trying to address in its classic counterpart. Recently, Liu et al. [11] empirical evaluated the effectiveness and performance of their novel compiler-level optimization approach of quantum programs, on a benchmark that is no longer available[1] and (to the best of our knowledge) cannot be re-created. Thus, how are others supposed to reproduce the study?

Lack of reproducibility goes beyond missing artifacts. Setting up a local environment similar to the environment other researchers used in the past to run a particular experimental analysis is, unfortunately, the most frustrating experience of being a researcher in software engineering. Inconsistent program versions, long installations steps that failed through the process, and obscure setup instructions are among the most common problems faced by researchers when they try to reproduce an experimental analysis. Virtual machines and docker containers only partially have solved this problem. Although both allow one to easily re-run an experimental setup on another computer, they do not provide a way to know how all the different modules of an experimental setup are connected, how to augment the defined setup, or how to run a pre-defined experiment with different inputs, etc.

Reproducibility (or the lack of it) of others' research is an increasing concern in software engineering [12], as shown by papers with conflicting results and by the recent establishment of artifact evaluation committees on top-tier conferences (e.g.,

[1] https://sites.google.com/site/qbenchmarks

POPL[2], ICSE[3]). We foresee that reproducibility will also be an issue in quantum software engineering research if, we (as community), do not establish and provide *experimental subjects and infrastructures*.

Thus, in this position paper, we propose the development of a novel framework named QBugs for quantum software testing and debugging research which aims to provide:

1) A catalog of open-source quantum algorithms.
2) A catalog of reproducible bugs in quantum algorithms.
3) Supporting infrastructure to enable controlled empirical experiments.

In the following we discuss the challenges to develop QBugs and research opportunities that could be built on top of QBugs.

II. Challenges

This section describes the main challenges to develop QBugs.

A. Quantum Programming Languages

First main challenge a framework such as QBugs would have to address is the support for different quantum programming languages [13], e.g., Q# [14], OpenQASM [15], Cirq [16], Quipper [17], and Scaffold [18]. In order to build the most diverse collection of reproducible bugs in quantum algorithms, we will provide support for the most adopted and supported quantum programming languages.

B. Open-Source Implementations of Quantum Algorithms

Coding quantum algorithms is very difficult and requires experts to do it. As opposed to its classic counterpart, finding open-source projects that implement quantum algorithms is a rather difficult task but crucial for the success of QBugs. With no open-source implementations of quantum algorithms there are no bugs to mine from, and therefore no *experimental subjects or infrastructure*.

To address this challenge and build a prototype of QBugs, we will rely on the open-source implementations of quantum algorithms that live in the quantum framework repositories. To the best of our knowledge, there are at least three sources of mature implementations of quantum algorithms / programs that we could consider.

- ProjectQ's framework repository[4] includes the implementation of 12 quantum algorithms.
- Qiskit-Aqua's repository[5] includes the implementation of 29 quantum algorithms developed in Qiskit, including the successful and well known Shor [1], Grover [2], and HHL [4] algorithms.
- Repository[6] of the book "Programming Quantum Computers" from O'Reilly [19] includes, overall, 166 quantum exercises and correspondent solutions (some implementing quantum algorithms) written in 6 different quantum frameworks: 10 algorithms written in Cirq, 4 in DWave, 29 in OpenQASM, 54 in QCEngine, 40 in Q#, and 29 in Qiskit.

[2]https://popl21.sigplan.org/track/popl-2021-artifact-evaluation
[3]https://conf.researchr.org/track/icse-2021/icse-2021-artifact-evaluation
[4]https://github.com/ProjectQ-Framework/ProjectQ/tree/c15f3b2/examples
[5]https://github.com/Qiskit/qiskit-aqua/tree/a8ab494/qiskit/aqua/algorithms
[6]https://github.com/oreilly-qc/oreilly-qc.github.io/tree/1a4c2cc/samples

C. Bugs Mining

The second challenge is related to the bug mining procedure. Given the implementation of a quantum algorithm, how could we automatically identify which bugs have been reported (thus real bugs), and what was the fix?

To address this challenge, we will adopt the same procedure Just et al. [7] and Gyimesi et al. [9] used to create the Defects4J database and the BugsJS database, respectively. In detail, we will first build a procedure to automatically map each bug report, that was labeled as "bug" or "issue" in the issue platform (e.g., Github issues), to a commit message in order to find the commit that fixed the bug. Then, we will identify the bug commit as being the commit right before the bug fixing commit and add that bug to our catalog. Others have started studying which bugs may raise from some quantum algorithms implementations [20, 21]. We, on the other hand, will *automatically* extract bugs from the repository's history.

A preliminary investigation of this challenge reveals, for example, that 93 issues were labeled as "type: bug" and closed in the Qiskit-Aqua's repository. For example, issue #928[7] reported a bug in the implementation of the QAOA algorithm [22], which is then addressed in commit 5695f9f[8] with the message: "Fix #928". Note that as repositories such as ProjectQ's framework repository, Qiskit-Aqua's repository, among others, also include the source code of the quantum language, framework, or simulator, some bug reports marked as "bug" might not be related to a bug in any quantum algorithm, but related to a bug in the quantum framework. For example, issue #1324[9] (also labeled as "bug") was related to a bug in the quantum framework. The proposed fix[10] only modified files under `qiskit/aqua/operators/` and the source code of quantum algorithms lives under `qiskit/aqua/algorithms/`. QBugs' bug mining module will have to be able to distinguish between framework's bugs and quantum algorithm's bugs in order to create an accurate catalog.

D. Bugs Reproducibility

Another challenge in QBugs is the reproducibility of bugs. Quantum programs are not by design deterministic. Thus, given a buggy version of a quantum algorithm, is it possible to automatically reproduce/trigger the buggy behavior in a deterministic way?

According to a recent study conducted by Fingerhuth et al. [23], automated tests of quantum algorithms appear to be very popular: 23 open-source quantum projects out of 26 have tests in place. Furthermore, Fingerhuth et al. [23] also concluded that the ratio of code exercise by the tests is slightly above the industry-expected standard (87% vs. 85%). Research on statistical test oracles for quantum algorithms have been proposed [21] which may further leverage the use of software tests in quantum computing.

[7]https://github.com/Qiskit/qiskit-aqua/issues/928
[8]https://github.com/Qiskit/qiskit-aqua/commit/5695f9f
[9]https://github.com/Qiskit/qiskit-aqua/issues/1324
[10]https://github.com/Qiskit/qiskit-aqua/pull/1340/files

One possible avenue to address this challenge is, therefore, to use the project's tests to run a buggy version of a quantum algorithm and to assess whether a bug is reproducible. For quantum projects with no tests in place, e.g., the exercises in the book "Programming Quantum Computers" from O'Reilly [19], another methodology would have to be defined (e.g., runtime oracles [24, 25, 26]).

III. Opportunities

This section describes the several research and teaching opportunities that could be developed on top of QBugs.

A. Research opportunities

Research areas that will immediately benefit from QBugs include: quantum software testing (e.g., regression testing, mutation testing, automatic test generation) and quantum debugging (e.g., fault localization), program comprehension, automated program repair, software evolution, mining software repositories, and machine learning [27] in quantum software engineering.

For instance, once a prototype of QBugs is available, it would be interesting to investigate how are bugs introduced in quantum algorithms, type of quantum bugs, and how are quantum bugs fixed. This could leverage research on novel static/dynamic tools tailored to identify quantum bugs on quantum programs. The catalog of real bugs will also allow researchers to develop and evaluate the effectiveness of, e.g., fault localization techniques at identifying the components that are more likely to be faulty, or the effectiveness of test generators at generating tests that trigger the faulty behavior.

In summary, QBugs will:

- Enable researchers to perform realistic experiments on real quantum software and on real quantum bugs.
- Allow researchers to focus on research ideas by freeing them from the task of (re-)developing an experimental infrastructure.
- Foster reproducibility and comparability of empirical studies by providing reusable artifacts and datasets.

B. Teaching opportunities

We also foresee that QBugs will be useful for software testing and quantum classes at undergraduate or graduate level. Instructors would have a large pool of artifacts from which they can select the best ones that might support the learning goals of their software testing or quantum-related course. Students will be able to explore how quantum algorithms have been developed and tested, experiment previously proposed quantum software engineering research, reproduce previously published results, and try new ideas/prototypes.

IV. Related Work

For the quantum software engineering's classical counterpart, several catalogs of real and artificial software faults [6, 7, 8, 9, 10] for different programming languages have been proposed and widely accepted by the research community. For example, the Software-artifact Infrastructure Repository (SIR) [6] is considered the first attempt to provide a database of bugs to enable reproducibility in software testing research. SIR provides 81 programs written in Java, C, C++, and C# and most of the faults are hand-seeded or obtained from program mutation. Defects4J [7] is another well known dataset that contains 835 real bugs from 17 Java projects, and it has been used for benchmarking and comparing (1) automatic test generation approaches [28], (2) fault localization techniques [29], (3) automatic program repair techniques [30]; and it also been used to study the properties of the bugs and their characteristics [31]. Besides providing real bugs, Defects4J also provides uniform access to the bugs through its own API by abstracting away the version control system (e.g., git) and build system.

We, on the other hand, aim to (1) gather a significant number of real bugs that have been reported to open-source quantum algorithms and therefore are quantum-based, and (2) integrate into our framework previously proposed tools / techniques on quantum software testing and debugging to ease future evaluations and comparisons (e.g., the mutation testing tool for quantum computing MTQC [32]).

V. Conclusion

A core tenet of the scientific method is reproducibility of experiments. To reproduce an empirical study in, e.g., software testing, one would require to use the same *experimental subjects*, i.e., same software projects and/or same catalog of software bugs, and the same *experimental infrastructure*, e.g., scripts and tools. Although the former and the latter have been investigated in detail in software engineering, there is not yet a well defined benchmark to ease reproducibility in quantum computing. Given the lack of reproducibility in software engineering [12], we foresee that such concern will also be valid in quantum computing in the near future. Thus, in this position paper, we propose a framework which will provide a catalog of quantum bugs and an infrastructure to conduct empirical experiments. We envisage that our framework will have a large potential for future research in the field, in particular reproducibility and comparability of empirical studies. For instance, researchers will be able to easily evaluate and compare the performance of different techniques under a common setup.

Acknowledgment

This work was supported by the Instituto de Telecomunicações (IT) Research Unit, ref. UIDB/EEA/50008/2020, granted by FCT/MCTES; by the Foundation for Science and Technology (FCT), project QuantumMining ref. POCI-01-0145-FEDER-031826 and project QuantumPrime ref. PTDC/EEI-TEL/8017/2020; by the FEDER through the COMPETE 2020 programme and by the Regional Operational Program of Lisboa, project Predict ref. PTDC/CCI-CIF/29877/2017; by the EU, project ref. UIDB/50008/2020-UIDP/50008/2020 (action QuRUNNER); and by the LASIGE Research Unit, ref. UIDB/00408/2020 and ref. UIDP/00408/2020.

References

[1] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM Review*, vol. 41, no. 2, pp. 303–332, 1999.

[2] L. K. Grover, "A Fast Quantum Mechanical Algorithm for Database Search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 212–219.

[3] E. Farhi and A. W. Harrow, "Quantum supremacy through the quantum approximate optimization algorithm," 2019.

[4] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum Algorithm for Linear Systems of Equations," *Phys. Rev. Lett.*, vol. 103, p. 150502, Oct 2009. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.103.150502

[5] J. Zhao, "Quantum Software Engineering: Landscapes and Horizons," 2020.

[6] H. Do, S. Elbaum, and G. Rothermel, "Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact," *Empirical Software Engineering*, vol. 10, no. 4, pp. 405–435, 2005. [Online]. Available: https://doi.org/10.1007/s10664-005-3861-2

[7] R. Just, D. Jalali, and M. D. Ernst, "Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ser. ISSTA 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 437–440. [Online]. Available: https://doi.org/10.1145/2610384.2628055

[8] R. K. Saha, Y. Lyu, W. Lam, H. Yoshida, and M. R. Prasad, "Bugs.Jar: A Large-Scale, Diverse Dataset of Real-World Java Bugs," in *Proceedings of the 15th International Conference on Mining Software Repositories*, ser. MSR '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 10–13. [Online]. Available: https://doi.org/10.1145/3196398.3196473

[9] P. Gyimesi, B. Vancsics, A. Stocco, D. Mazinanian, A. Beszédes, R. Ferenc, and A. Mesbah, "BugsJS: a Benchmark of JavaScript Bugs," in *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, 2019, pp. 90–101.

[10] S. H. Tan, J. Yi, Yulis, S. Mechtaev, and A. Roychoudhury, "Codeflaws: A Programming Competition Benchmark for Evaluating Automated Program Repair Tools," in *Proceedings of the 39th International Conference on Software Engineering Companion*, ser. ICSE-C '17. IEEE Press, 2017, p. 180–182. [Online]. Available: https://doi.org/10.1109/ICSE-C.2017.76

[11] P. Liu, S. Hu, M. Pistoia, C. R. Chen, and J. M. Gambetta, "Stochastic Optimization of Quantum Programs," *Computer*, vol. 52, no. 6, pp. 58–67, 2019.

[12] L. A. Barba, "Terminologies for Reproducible Research," *CoRR*, vol. abs/1802.03311, 2018. [Online]. Available: http://arxiv.org/abs/1802.03311

[13] B. Heim, M. Soeken, S. Marshall, C. Granade, M. Roetteler, A. Geller, M. Troyer, and K. Svore, "Quantum programming languages," *Nature Reviews Physics*, pp. 1–14, 2020. [Online]. Available: https://doi.org/10.1038/s42254-020-00245-7

[14] K. Svore, A. Geller, M. Troyer, J. Azariah, C. Granade, B. Heim, V. Kliuchnikov, M. Mykhailova, A. Paz, and M. Roetteler, "Q#: Enabling Scalable Quantum Computing and Development with a High-Level DSL," in *Proceedings of the Real World Domain Specific Languages Workshop 2018*, ser. RWDSL2018. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3183895.3183901

[15] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open Quantum Assembly Language," 2017.

[16] G. A. Q. Team. (2018, July) Cirq: an open source framework for programming quantum computers. [Online]. Available: https://quantumai.google/cirq

[17] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron, "Quipper: A Scalable Quantum Programming Language," in *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 333–342. [Online]. Available: https://doi.org/10.1145/2491956.2462177

[18] A. J. Abhari, A. Faruque, M. J. Dousti, L. Svec, O. Catu, A. Chakrabati, C.-F. Chiang, S. Vanderwilt, J. Black, F. Chong, M. Martonosi, M. Suchara, K. Brown, M. Pedram, and T. Brun, "Scaffold: Quantum Programming Language," Princeton University, Tech. Rep., June 2012, technical Report TR-934-12. [Online]. Available: https://www.cs.princeton.edu/research/techreps/TR-934-12

[19] M. G.-S. Eric R. Johnston, Nic Harrigan, *Programming Quantum Computers*. O'Reilly Media, Inc, 2019.

[20] Y. Huang and M. Martonosi, "QDB: From Quantum Algorithms Towards Correct Quantum Programs," 2018.

[21] ——, "Statistical Assertions for Validating Patterns and Finding Bugs in Quantum Programs," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 541–553.

[22] E. Farhi, J. Goldstone, and S. Gutmann, "A Quantum Approximate Optimization Algorithm," 2014.

[23] M. Fingerhuth, T. Babej, and P. Wittek, "Open source software in quantum computing," *PLOS ONE*, vol. 13, no. 12, pp. 1–28, 12 2018.

[24] G. Li, L. Zhou, N. Yu, Y. Ding, M. Ying, and Y. Xie, "Projection-Based Runtime Assertions for Testing and Debugging Quantum Programs," *Proceedings of the*

ACM on Programming Languages, vol. 4, no. OOPSLA, Nov. 2020. [Online]. Available: https://doi.org/10.1145/3428218

[25] J. Liu, G. T. Byrd, and H. Zhou, "Quantum Circuits for Dynamic Runtime Assertions in Quantum Computation," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1017–1030. [Online]. Available: https://doi.org/10.1145/3373376.3378488

[26] J. Liu and H. Zhou, "Systematic Approaches for Precise and Approximate Quantum State Runtime Assertion," in *27th IEEE International Symposium on High-Performance Computer Architecture*, ser. HPCA '21, 2021.

[27] S. B. Ramezani, A. Sommers, H. K. Manchukonda, S. Rahimi, and A. Amirlatifi, "Machine learning algorithms in quantum computing: A survey," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8.

[28] S. Shamshiri, R. Just, J. M. Rojas, G. Fraser, P. McMinn, and A. Arcuri, "Do Automatically Generated Unit Tests Find Real Faults? An Empirical Study of Effectiveness and Challenges," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 201–211.

[29] S. Pearson, J. Campos, R. Just, G. Fraser, R. Abreu, M. D. Ernst, D. Pang, and B. Keller, "Evaluating and Improving Fault Localization," in *Proceedings of the 39th International Conference on Software Engineering*, ser. ICSE '17. IEEE Press, 2017, p. 609–620. [Online]. Available: https://doi.org/10.1109/ICSE.2017.62

[30] M. Martinez, T. Durieux, J. Xuan, R. Sommerard, and M. Monperrus, "Automatic Repair of Real Bugs: An Experience Report on the Defects4J Dataset," 2015.

[31] V. Sobreira, T. Durieux, F. Madeiral, M. Monperrus, and M. de Almeida Maia, "Dissection of a bug dataset: Anatomy of 395 patches from Defects4J," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018, pp. 130–140.

[32] J. Pellejero, "MTQC: Mutation Testing for Quantum Computing," https://javpelle.github.io/MTQC, 06 2020, visited on 2021-03-12.