A Roadmap for Quantum Software Engineering: applying the lessons learned from the classics

Enrique Moguel, Javier Berrocal, Jose García-Alonso and Juan Manuel Murillo

SPILab - Social and Pervasive Innovation Lab. Quercus Software Engineering Group. Escuela Politécnica. Avenida de la Universidad s/n. University of Extremadura. 10004 - Cáceres. Spain

Abstract

Quantum Computing is one of the emerging areas of computing that currently generates more expectations. However, there are many doubts about its actual future projection. On the one hand, the industry shows reluctance to invest in it. The main reasons are the high costs of the hardware, together with the fact that current commercial quantum computers offer a potential that goes little beyond experimentation. On the other hand, there is controversy in the research community about the feasibility of creating and programming powerful and reliable quantum computers. The possibility of having reliable hardware with a reasonable number of Qubits seems still distant. Finally, current quantum programming tools are still at almost logic gate level, which limits the possibility of creating real complex quantum software systems. If we look back in time, this situation is reminiscent of the Software Crisis experienced by classical computing in the 60's. This talk starts from this analogy and, analyzing the advances and the lessons learned in the field of Software Engineering in the last 60 years, raises the directions that could help to develop the future Quantum Software Engineering.

Keywords

Quantum Computing, Future Quantum Software Engineering, Quantum Software Crisis,

1. Introduction

Since the principles of quantum mechanics were established in the early 20th century by Max Planck and Niels Bohr [1, 2, 3], their disruptive approaches have led to innovations in many fields of science such as chemistry, optics, microelectronics, telecommunications or computing.

In the field of computing, the construction of the first quantum computer [4] capable of implementing Qubits as well as the concepts of superposition and entanglement enabled the ability of managing all possible states of a computer at the same time. From the point of view of computation, this increase in computing power not only enables the resolution of problems in much less time than with classical computers, but also to tackle problems that, due to their complexity, are out of their capabilities [5] [6]. Thus, the Theory of Computation is creating new

1st Quantum Software Engineering and Technology Workshop. Q-SET'20 co-located with the IEEE Quantum Week. IEEE International Conference on Quantum Computing and Engineering (QCE20) , Tuesday, October 13, 2020

enrique@unex.es (E. Moguel); jberolm@unex.es (J. Berrocal); jgaralo@unex.es (J. García-Alonso); juanmamu@unex.es (J. M. Murillo)

http://www.spilab.es/ (E. Moguel); http://www.spilab.es/ (J. Berrocal); http://www.spilab.es/ (J. García-Alonso); http://www.spilab.es/ (J. M. Murillo)

0000-0002-4096-1282 (E. Moguel); 0000-0002-1007-2134 (J. Berrocal); 0000-0002-6819-0299 (J. García-Alonso); 0000-0003-4961-4030 (J. M. Murillo)

categories in which to include those problems that, not having a reasonable resolution using classical computing, can be solved by quantum computing [7]. In particular, the capabilities of these computers allows the segregation of a new category within the class of P-SPACE problems called ***BQP*** - Bounded-error Quantum Polynomial time [8] that groups those problems that are solvable by a quantum computer in polynomial time with a probability of error limited by 1/3.

Thus, Quantum computers, in any of its models (quantum circuit model, adiabatic quantum computer, etc.) open new horizons for Software Engineering [9]. Programming the algorithms needed to solve ***BQP*** problems in quantum computers will require appropriate tools. One may feel inclined to directly bring the usual tools and procedures of classical software engineering into the quantum realm. However, quantum non-Von Neumann architectures will require new techniques and tools to be programmed [10]. Most of the usual techniques make no sense in the new scenario. For example, Using while-type loops as they are build in sequential programming would not make sense since inspecting the variable to detect the end of the loop would collapse the system. On the other hand, quantum computers and the principles on which they are based introduce new concepts such as *Superposition*, *Entanglement* or *Decoherence* that do not exist in classical software engineering.

Designing the new techniques needed in the field of Quantum Software Engineering (QSE) requires taking into account all these new concepts along with all the lessons learned in classical software engineering development. During the last sixty years [11] the classical software engineering generated advances, even overcoming several crises [12], that made possible the construction of complex software systems that deeply penetrated all aspects of today's society. The advances that have made such a huge development possible range from the introduction of abstractions in programming languages such as objects or microservices to the development of management disciplines that make the investment in software construction profitable. This article takes a look back at the evolution of classical software engineering techniques, their motivations and their foundations, to outline the lines that could be followed in the development of quantum software development techniques. The aim is to provide general guidelines to shorten the path towards the incorporation of quantum software in everyday life.

The rest of the paper is structured as follows. Section two introduces the motivations for carrying out the reflections made in this paper. Section three outlines the directions in which quantum software engineering should be developed. The paper ends with the conclusions and the future works.

2. Crossroads of Quantum Computing

There are many expectations placed on Quantum Computing to contribute to almost any area of science. For example, exact calculations of molecular properties are currently intractable because their computational cost grows exponentially with the number of atoms. However, the power of quantum computer can make it a tractable problem [13]. Complex mathematical processes behind finance such as Monte Carlo or Bayesian Inference could be lightened to provide real-time solutions [14]. Machine learning techniques that are applied today in multiple facets of our lives will increase their precision and speed so they can be used in problems that today are out of reach [15].

However, despite all these expectations, there are also well-founded doubts about the future projection of quantum computing. Somehow, one can recognize aspects similar to those experienced by classical computing in the late fifties and early sixties that were so well captured by Edsger W. Dijkstra in his ACM Turing Lecture in 1972 [16] and that finally led to the software crisis.

First of all, there is the cost of the hardware. D-Wave markets its D-Wave 2000Q, pending the launch of its new Advantage, for $15M [17]. Google's Sycamore chip [18] or IBM System Q [19] are not commercially available. Fortunately, manufacturers made them available in the cloud and they can be accessed through APIs. In all cases, the hardware is bulky and must be in a very special physical condition. All this is reminiscent of those unique, single-copy computers such as the ENIAC whose size and cost aroused the general interest of the press in 1947 [20]. Then, it was $400,000 and it had taken 200.000 man-hours to set up. Although the power of those computers was very little -compared to today's computers, it was enough to solve very concrete tasks in very controlled conditions much faster than a person would.

The work of programmers was very tightly linked to the machine they were programming for. Programmers, knowing the characteristics of the machine they were working on, developed their own set of tricks to make it work in the right way. Programs were difficult to export to other machines and difficult to use by other programmers. It was what later became known as Spaghetti Code [21]. This situation is again reminiscent of the work done today by programmers approaching quantum computers. With the limitations in qubits of current hardware, they develop mostly experiments for a specific machine and show how small, very concrete problems can be solved by a quantum computer. At the same time, this let programmers learn how quantum computers work.

Those mainframes were very unreliable mainly due to the technology they were based on, vacuum tubes. This made their results suspicious and often a human had to double-check the calculations. Something similar is happening today with quantum computers. The fragile nature of qubits subject to decoherence [22] and other sources of noise, causes errors that must be corrected by techniques called Quantum Error Correction (QEC) [23] [24]. This makes the reliability of the results provided by quantum computers dependent on statistics. Thus, While one of the great challenges of quantum computing today is to achieve more stable qubits and reduce the need for error correction, there are doubts that this can be achieved [25].

Surprisingly, the work of the programmers was bearable while the power of the hardware remained contained. But as soon as the power of the machines began to grow the situation became untenable. Just like Edsger Dijkstra said when talking about the Software Crisis *"the major cause is... that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming had become an equally gigantic problem"* [16]. Today we know that, if that fact was the trigger of the software crisis, the cause was the absolute lack of appropriate engineering tools and procedures to build software.

All of the above can lead one to assume that we are on the verge of a potential Quantum Software Crisis. Software Engineering must pay attention to these signals in order to anticipate it. Fortunately today we have the experience accumulated over the past sixty years thanks to the work of an eminent scientific community. Identifying the crossroads and the good decisions scientist took in each of them can contribute to take safe steps in the evolution of QSE. The next section identifies some of these crossroads that QSE might face as well as what might be the best decisions based on what proved to be good decisions in the field of classical computing.

3. Directions for Quantum Software Engineering

The fact that quantum computing is reaching the degree of maturity that claims for a Quantum Software Engineering is evidenced by the considerable amount of research works on this subject that has been developed in recent years. The most complete and recent survey on this topic is maybe the one developed by Prof. Jianjun Zhao [26]. It provides a definition of QSE that could be widely accepted by the community: *"Quantum software engineering is the use of sound engineering principles for the development, operation, and maintenance of quantum software and the associated document to obtain economically quantum software that is reliable and works efficiently on quantum computers".*

This section does not attempt to exhaustively review existing advances but simply to outline some of them in order to launch reflections that point to some interesting research directions in the field of Quantum Software Engineering.

3.1. Quantum Software Processes and Methodologies

Nowadays, there is no doubts about the interest that Quantum Software Engineering is arousing [27]. One of the main lessons learned during the evolution of classic software engineering is that software needs processes and processes need methodologies that help carry out each of the activities of the processes whether it is requirements specification, architectural design, detailed design, implementation or testing. Regarding the process for quantum software development, it could be assumed that the basic strategies for classical software development could be valid. They provide a general guide to approach the problem with the aim of getting a piece of software that supports the problem. In any case, quantum software processes, as commonly accepted for classical software today, should be as iterative, incremental and agile as possible.

However, the methodologies for carrying out each activity of the process should be reviewed to address the needs of quantum software development. The reason being that the techniques designed for classical computing are based on an underlying model of computing in which a sequence of instructions manipulates a set of data. The final state of the data is the output of the program. For example, the way in which a design is produced from a requirement specification is tightly aligned with such computational model. Nevertheless, such model is radically different to the quantum computational model. We have not such sequence of instructions but a system with the ability of having a set of possible states and being in all of them at the same time. Computation stops when a certain subset of the system state is in a desired state, which will define the state of the whole system by collapse. There are authors who have already approached some methodologies to deal with some activities such as design [28] or even re-engineering [29].

3.2. Abstractions for Quantum Software

A key step for having new techniques and methodologies is defining new abstractions for the quantum computing world. Let's focus now on the mechanisms of abstraction that we use in the design of classic software.

For example, when we include a class model in an UML class diagram we are representing a real-world object. This object exists in the system with the ability to interact with other entities. The ways in which entities interact is modeled in a sequence diagram.

We use classes because they proved to be much richer than the Abstract Data Types (ADT) that we used previously. They (ADT) basically allowed data structures to be modeled along with the set of operations to manipulate them. The reason to have them was that giving each programmer the responsibility of implementing the operations of a data structure introduced not only duplicate efforts but semantic variations in the structures that made them difficult to reuse. Both, the structures and the operations were hardly portable from one program to another. Moreover, the functionality code ended up being tangled with the code that implemented the operations on the data structures. Software Engineering Researchers realized then that a data structure was meaningless without the set of operations to manipulate it directly.

Before ADTs, we simply used procedures and functions to model pieces of functionality that the programmer could focus on, program, maintain, and evolve in isolation. In any case, all those abstractions correspond to pieces of software functionality as sequences of instructions that eventually manipulate data and that interact with each other by means of a protocol of control transfer from one to another. Again, this is due to classic computers being based on Von Neumann's machine model that expects to execute instructions in a certain sequence and our models must represent reality on that computer model.

It is worth remembering at this point what happened with the code written by those programmers born in procedural programming when they went on to the use of classes. It was common to find classes that modeled data structures (the TADs they were used to), classes grouping different sets of modules and a main class which collected the program control flow. This was the case even when object-oriented programming books systematically started explaining the differences between procedural and OOP programming. It was very hard then for programmers to change their way of conceiving software systems even when the underlying computer model was the same. Thus, the new change we are facing in which the underlying computer model is completely different is expected to be even harder.

When we think about taking our abstractions into the quantum environment, many questions arise: For example, Does it make sense to have a class in quantum computing? Is that a proper abstraction? What is the interest of having a class instance in all its possible states? What is the point of modeling a class in which part of its state is entangled with that of another class? What are the best abstractions for doing this? No doubt, we will feel inclined to replicate our techniques and abstractions from classical computing to quantum computing. But then, just as it happened to programmers who moved from procedural to object oriented programming, we may end up using them improperly wanting to do something that is intrinsically different. We will provide programmers with the possibility of generating classic computing solutions that will run on quantum computers, but we will not have generated the opportunity to "think and model in quantum".

With all the above, maybe one of the most interesting directions to be addressed by quantum software engineering research is to propose adequate abstractions for modelling, designing and building quantum programs. When thinking about these abstractions it is also interesting to think that the type of problems that will be solved with quantum computing are those included in the ***BQP*** class. If the typology of these problems can be categorized, domain-specific modeling languages could be designed for them.

3.3. Quantum Structured Programming

Currently, quantum programming languages [30, 31, 32, 33] work almost at the quantum circuit level. Its description is very reminiscent of the wiring work needed by the ENIAC to execute programs.

One of the greatest advances towards modern programming languages was the development of structured programming that led to the basic structures for computing: sequence, bifurcation, iteration. It opened the doors to the diverse programming paradigms we have today in classical computing.

As with abstractions, we could wonder what should be the basic structures and patterns in the development of quantum programs. Again, to do this, one must assume that the nature of a quantum computer is very different to Von Newman's architecture model. There are researchers who are already concerned about this aspect: [34, 35, 36].

4. Conclusions

This paper has approached the motivations and needs for quantum software engineering. Starting from them, and taking as a base the evolution followed by classical software engineering during the last sixty years, some directions have been pointed out that could direct the development of future quantum software engineering.

As if it were a new spin of the cycle, Quantum Computing is at the same evolution state as Classic Computing was in the 60's. It is still unknown what will come next, but we can anticipate similarities in what has happened in the last sixty years. Problems similar to those occurred in the past with classical computing will come and actions will have to be taken. It will be necessary to propose new technologies (languages, routines, operating systems, interaction protocols...), new architectures, new methodologies, etc..

A key strategy for all the above challenges will be to involve companies, in particular those willing to use quantum computing technology in the future. This will maximize the opportunities for the penetration of quantum computing in society.

Without wishing to be exhaustive, this paper collects some reflections that hopefully can be of help and inspiration for other researchers. There are many aspects that this paper does not cover that are of great interest such as, for example, the coexistence of classical software with quantum software and how this can affect quantum software due to an Amdahl's law-like relationship, how should be the base software for a quantum computer or how DevOps techniques could be extended to develop quantum solutions and the opportunity to made it. As future work, the authors of this paper are especially interested in the research of new basic abstractions new abstractions to enable the modelling of the quantum nature of software systems.

Acknowledgments

This work was supported by (0499_4IE_PLUS_4_E) funded by the Interreg V-A España-Portugal (POCTEP) 2014-2020 program, by the project RTI2018-094591-B-I00 (MCIU/AEI/FEDER, UE) and FPU17/02251 grant, by project IB18030 funded by the Government of Extremadura.

References

[1] M. J. Klein, Max planck and the beginnings of the quantum theory, Archive for History of Exact Sciences 1 (1961) 459–479.

[2] W. Heisenberg, The development of the interpretation of the quantum theory, 1955.

[3] W. Pauli, Niels Bohr and the Development of Physics; Essays Dedicated to Niels Bohr on the Occasion of His Seventieth Birthday, New York, McGraw-Hill, 1955. URL: https://books.google.es/books?id=YIXFzQEACAAJ.

[4] I. L. Chuang, N. Gershenfeld, M. Kubinec, Experimental implementation of fast quantum searching, Phys. Rev. Lett. 80 (1998) 3408–3411. URL: https://link.aps.org/doi/10.1103/PhysRevLett.80.3408. doi:10.1103/PhysRevLett.80.3408.

[5] E. Farhi, J. Goldstone, S. Gutmann, J. Lapan, A. Lundgren, D. Preda, A quantum adiabatic evolution algorithm applied to random instances of an np-complete problem, Science 292 (2001) 472–475.

[6] S. Aaronson, The limits of quantum, Scientific American 298 (2008) 62–69.

[7] E. Bernstein, U. Vazirani, Quantum complexity theory, SIAM Journal on computing 26 (1997) 1411–1473.

[8] S. Aaronson, Bqp and the polynomial hierarchy, in: Proceedings of the forty-second ACM symposium on Theory of computing, 2010, pp. 141–150.

[9] M. Piattini, G. Peterssen, R. Pérez-Castillo, J. L. Hevia, M. A. Serrano, G. Hernández, I. G. R. de Guzmán, C. A. Paradela, M. Polo, E. Murina, L. Jiménez, J. C. Marqueño, R. Gallego, J. Tura, F. Phillipson, J. M. Murillo, A. Niño, M. Rodríguez, The talavera manifesto for quantum software engineering and programming, in: M. Piattini, G. Peterssen, R. Pérez-Castillo, J. L. Hevia, M. A. Serrano (Eds.), Short Papers Proceedings of the 1st International Workshop on the QuANtum SoftWare Engineering & pRogramming, Talavera de la Reina, Spain, February 11-12, 2020, volume 2561 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020, pp. 1–5. URL: http://ceur-ws.org/Vol-2561/paper0.pdf.

[10] T. M. Conte, E. P. DeBenedictis, P. A. Gargini, E. Track, Rebooting computing: The road ahead, Computer 50 (2017) 20–29.

[11] P. Naur, B. Randell, Software Engineering: Report of a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO, 1969.

[12] M. S. Mahoney, The history of computing in the history of technology, Annals of the History of Computing 10 (1988) 113–125.

[13] B. P. Lanyon, J. D. Whitfield, G. G. Gillett, M. E. Goggin, M. P. Almeida, I. Kassal, J. D. Biamonte, M. Mohseni, B. J. Powell, M. Barbieri, et al., Towards quantum chemistry on a quantum computer, Nature chemistry 2 (2010) 106–111.

[14] R. Orus, S. Mugel, E. Lizaso, Quantum computing for finance: overview and prospects, Reviews in Physics 4 (2019) 100028.

[15] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, S. Lloyd, Quantum machine learning, Nature 549 (2017) 195–202.

[16] E. W. Dijkstra, The humble programmer, Commun. ACM 15 (1972) 859–866. URL: https://doi.org/10.1145/355604.361591. doi:10.1145/355604.361591.

[17] D-Wave Systems, The d-wave 2000q™. quantum computer technology overview, 2017. URL: https://www.dwavesys.com/sites/default/files/D-Wave%202000Q%20Tech%20Collateral_0117F_0.pdf.

[18] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell, et al., Quantum supremacy using a programmable superconducting processor, Nature 574 (2019) 505–510.

[19] IBM, IBM Q System One, 2017. URL: https://www.ibm.com/quantum-computing/.

[20] The New York Times, Electronic computers flashes answers, may speed engineering, 1946. URL: https://www.computerhistory.org/revolution/birth-of-the-computer/4/78/323.

[21] G. L. Steele Jr, Macaroni is better than spaghetti, ACM SIGPLAN Notices 12 (1977) 60–66.

[22] P. W. Shor, Scheme for reducing decoherence in quantum computer memory, Physical review A 52 (1995) R2493.

[23] J. Kelly, R. Barends, A. G. Fowler, A. Megrant, E. Jeffrey, T. C. White, D. Sank, J. Y. Mutus, B. Campbell, Y. Chen, et al., State preservation by repetitive error detection in a superconducting quantum circuit, Nature 519 (2015) 66–69.

[24] C. K. Andersen, A. Remm, S. Lazar, S. Krinner, N. Lacroix, G. J. Norris, M. Gabureac, C. Eichler, A. Wallraff, Repeated quantum error detection in a surface code, Nature Physics (2020) 1–6.

[25] M. Dyakonov, When will useful quantum computers be constructed? not in the foreseeable future, this physicist argues. here's why: The case against: Quantum computing, IEEE Spectrum 56 (2019) 24–29.

[26] J. Zhao, Quantum software engineering: Landscapes and horizons, arXiv preprint arXiv:2007.07047 (2020).

[27] M. Piattini, G. Peterssen, R. Pérez-Castillo, Quantum computing: A new software engineering golden age, SIGSOFT Softw. Eng. Notes 45 (2020) 12–14. URL: https://doi.org/10.1145/3402127.3402131. doi:10.1145/3402127.3402131.

[28] C. A. Pérez-Delgado, 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.

[29] L. Jiménez-Navajas, R. Pérez-Castillo, M. Piattini, Reverse engineering of quantum programs toward KDM models, in: M. J. Shepperd, F. B. e Abreu, A. R. da Silva, R. Pérez-Castillo (Eds.), Quality of Information and Communications Technology - 13th International Conference, QUATIC 2020, Faro, Portugal, September 9-11, 2020, Proceedings, volume 1266 of *Communications in Computer and Information Science*, Springer, 2020, pp. 249–262. URL: https://doi.org/10.1007/978-3-030-58793-2_20. doi:10.1007/978-3-030-58793-2_20.

[30] S. J. Gay, Quantum programming languages: Survey and bibliography, Mathematical Structures in Computer Science 16 (2006) 581.

[31] D. A. Sofge, A survey of quantum programming languages: History, methods, and tools,

in: Second International Conference on Quantum, Nano and Micro Technologies (ICQNM 2008), IEEE, 2008, pp. 66–71.

[32] J. A. Miszczak, Models of quantum computation and quantum programming languages, arXiv preprint arXiv:1012.6035 (2010).

[33] S. Garhwal, M. Ghorani, A. Ahmad, Quantum programming language: A systematic review of research topic and top cited languages, Archives of Computational Methods in Engineering (2019) 1–22.

[34] F. Leymann, Towards a pattern language for quantum algorithms, in: International Workshop on Quantum Technology and Optimization Problems, Springer, 2019, pp. 218–230.

[35] E. Knill, Conventions for quantum pseudocode, Technical Report, Los Alamos National Lab., NM (United States), 1996.

[36] M. Ying, Y. Feng, Quantum loop programs, Acta Informatica 47 (2010) 221–250.