FEATURE ARTICLE: QUANTUM SOFTWARE

Toward a Quantum Software Engineering

Mario Piattini, *University of Castilla—La Mancha*
Manuel Serrano, *University of Castilla—La Mancha*
Ricardo Perez-Castillo, *University of Castilla—La Mancha*
Guido Petersen, *aQuantum*
Jose Luis Hevia, *aQuantum*

Nowadays, we are at the dawn of a new age, the quantum era. Quantum computing is no longer a dream; it is a reality that needs to be adopted. But this new technology is taking its first steps, so we still do not have models, standards, or methods to help us in the creation of new systems and the migration of current ones. Given the current state of quantum computing, we need to go back to the path software engineering took in the last century to achieve the new golden age for quantum software engineering.

If the 19th century was the machine era and 20th century was the information era, several experts agree that 21st century will be the quantum era. In fact, "The thing driving the hype is the realization that quantum computing is actually real. It is no longer a physicist's dream—it is an engineer's nightmare."[1]

Over the past three decades, our understanding of "quantum computers" has expanded drastically,[2] as the efforts to realize such an exotic computer have made steady, yet remarkable progress.[3] Using various "counterintuitive" principles of quantum mechanics, such as superposition and entanglement, quantum computers will provide faster computing speed,[4] providing high value for specific tasks in various important applications. For example, applications include, but are not limited to, the following:

- privacy and cryptography: certification of randomness and authentication;[5]
- supply chain and logistics: optimization problems in procurement, production and distribution, vehicle routing optimization, etc.;
- chemistry: simulations of complex molecules, discovery of new materials, advanced molecular design, etc.;
- economics and financial services: portfolio risk optimization and fraud detection, actual randomness for financial models, simulations and scenario analysis, etc.;
- energy and agriculture: production of ammonia, better distribution of resources, asset degradation modeling, etc.;
- medicine and health: protein folding and drug discovery, disease detection, noninvasive and high-precision surgeries, targeted drugs design, tailored medicine, improvement of the quality of life, prediction of therapeutic prescriptions, etc.;
- defense and national security programs.

In these areas, the quantum software engineer will implement "killer" software applications, in which the speed afforded by quantum computers will have real-world impact.[4]

As Humble and DeBenedictis[6] highlight, the prospects of quantum computing are exciting, and extraordinary expectations are now fueling a global effort to perfect quantum computing. Many countries (China, U.S., Japan, Russia, U.K., etc.) are investing huge quantities in quantum technology. Also, the most important companies (Google, IBM, Microsoft, Intel, Atos, Alibaba, etc.) are investigating how to take the most of this new technology for their businesses.

Stepney *et al.* from the University of York proposed a "Grand Challenge for Computing Research"[7] about Quantum Software Engineering—which covers the development of a full discipline of Quantum Software Engineering, ready to exploit the full potential of commercial quantum computer hardware, once it arrives. This challenge is to build the corresponding languages, tools and techniques for quantum software engineering." They emphasize the need to raise the level of thinking about quantum programs. Also, Van den Brink

1520-9202 © 2021 IEEE
Digital Object Identifier 10.1109/MITP.2020.3019522
Date of current version 29 January 2021.

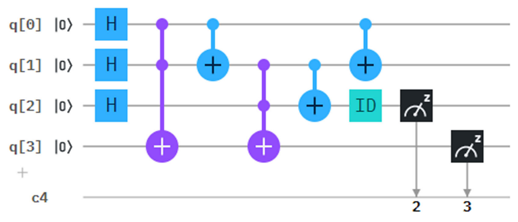

FIGURE 1. Quantum circuit for a full adder with carry.

et al.[8] pointed out the need of software engineering tooling for quantum programming. We revise the evolution of software engineering techniques and propose a set of priority subject areas to be addressed.

QUANTUM PROGRAMMING

Quantum computing changes the traditional bits, the smaller unit of information, with qubit. A qubit is usually represented with the electron spin or photons among other subatomic particles. A qubit is a multiple status quantum system, i.e., it is not only defined by zero or one as a classical bit, but possible values exist at the same time. So, a qubit can be zero or one with a certain probability (this is known as superposition and is the key for the high computational power). The actual value of a qubit is only known once it is measured, and then, the qubit is collapsed and cannot be used anymore without resetting. As a result, the philosophy of quantum programming is oriented toward exploring and searching optimal solutions in a probabilistic space.

The way in which quantum programmers work with qubits is through quantum circuits and quantum gates. In a similar way to the classical logic gate operations (AND, XOR, NOT), quantum logic gates compute the output states from arbitrary input states. Quantum gates are implemented by designing the time evolution of the input quantum states to achieve the desired output states, often by manipulating the external control signals. In other words, quantum gates alter the probability of being zero or one. Quantum circuits are collections of quantum gates interconnected by quantum wires, where qubit's states change consequently of those quantum gates. Parts of these circuits can be encapsulated into oracles or other quantum gates in order to reduce complexity.

Figure 1 provides an example of a quantum circuit for a full adder with carry for two numbers (represented by the two bottom qubits). Quantum circuits and gates can be represented graphically like in the example, but also through syntax-based notations that are provided by a wide variety of quantum programming languages (e.g., Q#, QASM, Cirq, pyquil, QCL, among many other). These languages usually adopt some control structures from imperative languages, or even functional programming facilities, to make quantum programming easier.

EVOLUTION OF SOFTWARE ENGINEERING

Boehm[9] perfectly summarizes the evolution of software engineering as a process of continuous thesis, antithesis, and synthesis. We must take also into account that, as it is reasonable, the evolution of software engineering was bottom-up. In fact, initially the hardware base is developed, remember the first computers in the late 1930s (Zuse's Z1) and 1940s (IBM's Mark I) built with tubes and capacitors, in the 1950s, the transistor-based computer, and in the 1960s, the appearance of computers with integrated circuit boards.

With the hardware came the machine languages and later also the first assembly programming languages. In the 1950s (for example, FORTRAN) and 1960s (COBOL), the first high-level programming languages were developed. With these languages arose the need of programming techniques, the most influential was the proposal of structured programming by Dijkstra in 1968[10] and the techniques proposed by Warnier and Jackson.

Over these concepts, in the 1970s, structured design techniques by Myers, Yourdon, and Constantine, or E/R model by Chen are proposed and, lately, structured analysis by Gane and Sarson, and DeMarco and Weinberg. Also, the C programming language was developed,[11] which gained a lot of popularity, specifically for developing operating systems' utilities. In the 1980s, complete methodologies (Merise, SSADM, Information Engineering, etc.) were suggested. This was the first golden age of software engineering.[12]

The same pattern was followed by object-oriented technologies. In the 1960s (Simula), 1970s (Smalltlak), and 1980s (C++, Objective-C, Eiffel), different object-oriented programming languages were tested. In the 1990s, object-oriented was considered "the" approach for developing information systems. Nearly 100 methodologies were proposed, first for object-oriented design and later for object-oriented analysis; the most important ones (Booch, Rumbaugh and Jacobson's) integrated in Unified Modeling Language and Unified Process. Booch[12] named this the "second golden age of software engineering."

Similar evolution was experienced by Aspect-Oriented Software Development. AspectJ was proposed in 2001, and then, design and analysis methodologies were created. Also, in the 2000s, the need of an empirical and evidence-based software engineering was recognized, and so several of the proposed techniques were subjected to validation.

In the 2010s, DevOps and several associated techniques have been producing another golden age.[12]

Quantum mechanics have been developed since the first 1900s, and nowadays, we have now several quantum computers that implements different layers

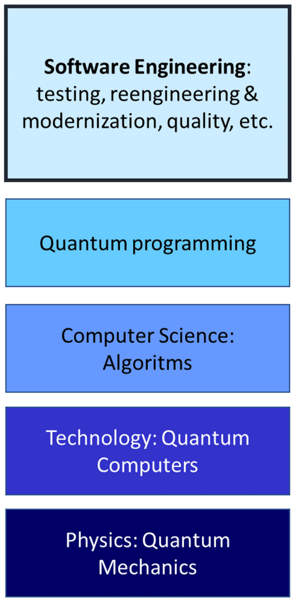
Software Engineering: testing, reengineering & modernization, quality, etc.
Quantum programming
Computer Science: Algoritms
Technology: Quantum Computers
Physics: Quantum Mechanics

FIGURE 2. Quantum technology and software engineering.

in the technology stack (see Figure 2), also several algorithms (https://quantumalgorithmzoo.org/) have been proposed, and different programming languages are already available. So, it is time to propose and validate software engineering techniques and to achieve a new golden age for quantum software engineering.

PRIORITY AREAS FOR QUANTUM SOFTWARE ENGINEERING

Quantum computing will affect all the areas of software engineering. As a matter of fact, most of the 14 areas in SWEBOK (Software Engineering Body of Knowledge)[13] should be updated to include quantum issues. As we previously stated in Talavera Manifesto for quantum software engineering and programming,[14] we believe that quantum computing will heavily influence: software design, software construction, software testing, software maintenance, and software quality; moderately in: software requirements, software engineering process, software engineering models and methods, and computing foundations; and has very little or no impact in: software configuration management, software engineering professional practice, software engineering economics, mathematical foundations, and engineering foundations.

For the first four SWEBOK areas (software design, software construction, software testing, and software maintenance), we propose some interesting themes:

- software design of quantum hybrid systems;
- testing techniques for quantum programs;
- quantum programs quality;
- re-engineering and modernization toward classical-quantum information systems.

There are also other areas of interest such as sustainability of quantum applications, as energy might be an issue in quantum algorithms, and high energy-consuming quantum systems should be tuned to be useful and viable.

Design of Quantum Hybrid systems

Design of quantum systems is related to, and includes, requirements engineering, conceptual modeling, among other challenges. In order to implement quantum programs, it is necessary to perform multiple steps that transform the current classical computing into hybrid systems that delegate CPU-intensive algorithms (some of them approximated in hybrid CPU-GPU computing platforms) to these new quantum systems, as well as other tasks that, from nature, could be more suited. These systems will have to be hybrid to take advantage of both worlds. It should be noted that dividing the tasks over quantum and classical computers can be a matter for all the software layers (see Figure 2).

However, this new kind of hybrid programming is neither easy nor trivial and requires not only knowledge and experience, but should be also supported by development methodologies, notations, medium and high-level languages or templates and stereotypes that allow the incorporation of quantum algorithms in the current computation scheme.

The needed development methodologies should be designed to work in a multilevel architecture (a specific characteristic of these hybrid systems) and should make it easy, as far as possible, the construction of new quantum programs that are capable of harnessing the power of new computers.[15] This task will be crucial, due to the huge complexity of programming of this new paradigm of program building.

Quantum Programs Testing

Given the inherent complexity of classical algorithms, their design and implementation are doomed to the appearance of multiple bugs and errors. In this way, it is essential to create a new paradigm of quantum algorithms testing to guarantee its correctness and alignment with the algorithm's functional requirements.

To achieve this goal, current testing techniques should be used or adapted, accommodating classical testing techniques and assertions. Since, most of quantum programs are oriented to search approximated solutions in a probabilistic space, testing of quantum programs should come up with a lot of innovation. For example, there are challenges like the definition of assertions and oracles or debugging among other. Debugging is one of the most important areas to achieve a practical verification of quantum programs. To achieve debugging, we might need the usage of specific simulators,[8] or quantum programming languages with specific characteristics that favor debugging.[16]

Additionally, quantum algorithms could take advantage of more advanced testing methods such as the use of mutation techniques to improve and optimize the testing of quantum applications.

Quantum Programs Quality Assurance

Quantum programs are no different from other software in terms of the quality and maintainability of the applications and their code. In addition, their complexity drives us to take special care in maintaining their quality at a high level.

Pursuing this goal can be as complex as seeking the Holy Grail but we cannot leave aside the quality assurance of these systems and we must apply the same techniques that we use in other types of software.

However, their different paradigm urges us to define new quality metrics and new methods of evaluation and quality assurance. This opens a new field of research in which new techniques for quantum programs must be developed, in which new quality metrics for modularity, maintainability, portability, among others, are needed. As a result, classical metrics have to be combined with others based on quantum gates and other quantum building blocks. Additionally, guidelines and good practices of programming should be defined as well.

Quantum Re-Engineering and Modernization

In order to harness the full power of the new quantum technology, it is necessary to be able to migrate applications, or part of them, to the new quantum computing paradigm. However, migrating from classical to quantum applications is no trivial task. In fact, given the incipient state of this technology, there are no standards for quantum languages or model-based design techniques.

To perform all these re-engineering processes, it is necessary to have standards, models, and systematic transformations that help to preserve business rules and facilitate migration among different classical and quantum environments.

This area is related to the software maintenance necessary for both, quantum and classical systems. Thus, this covers not only classical functionalities that need to be migrated toward quantum environments, but also new quantum applications that must be integrated in evolved versions of existing classical systems.

CONCLUSION

There is a certain urgency that we start developing and/or adapting the classical software engineering techniques to quantum programming characteristics. These proposals should be agnostic from the quantum computing technology.[14]

For researchers, following three important things must be considered.

1) Several quantum computer scientists do not know the software engineering principles and techniques, so several errors could be done again, and some expensive "rediscoveries" could happen.
2) We should adopt a more "agile" approach when proposing and developing software engineering quantum techniques, that is, do not wait until quantum programming languages are "stable" or "refined" in order to adapt existing techniques or create new ones, but develop them in parallel with the evolution of the quantum languages, starting from now.
3) Learning from the mistakes of the past, when proposing new software engineering quantum techniques, empirical validation should be a must.

For practitioners, it is advisable to start learning the basic concepts of quantum computing (for example, a first step could be the Brilliant's quantum computing course[17]) and working with some of the available environments (Microsoft's Quantum Development Kit, IBM's Q Experience and QISkit, Google's Cirq, Dwave and Rigetti's Forest, among other), in order to assess the benefits that this new paradigm could bring to their software applications.

For universities, we must follow the advice of Boehm:[9] "to keep courses and courseware continually refreshed and up-to-date, and to anticipate future trends and preparing students to deal with them" and so to incorporate in the curricula courses con quantum technologies, quantum computing, and quantum software engineering. Remembering that as Booch[12] remarks: "No matter the medium or the technology or the domain, the fundamentals of sound software engineering will always apply: craft sound abstractions," so we must dedicate a considerable amount of time to transmit these fundamentals. There is already an urgent demand for quantum software engineering.

In summary, all of us, as a software engineering community, we have a very interesting opportunity to bring a new golden age and contribute importantly to the advance of our society.

ACKNOWLEDGMENTS

This work was supported by the Spanish Ministry of Science and Innovation under Project SMOQUIN (PID2019-104791RB-I00).

REFERENCES

1. W. Knight, "Serious quantum computers are finally here. What are we going to do with them," *MIT Technol. Rev.*, vol. 30, 2018.
2. I. Bojanova, "The Digital Revolution: What's on the Horizon?" *IT Professional*, vol. 16, no. 1, pp. 8–12, 2014, doi: 10.1109/MITP.2014.11.
3. D. Maslov, Y. Nam, and J. Kim, "An outlook for quantum computing [point of view]," *Proc. IEEE*, vol. 107, no. 1, pp. 5–10, Jan. 2019.
4. L. Mueck, "Quantum software," *Nature*, vol. 549, no. 7671, 2017, Art. no. 171, doi: 10.1038/549171a.
5. L. O. Mailloux, I. C. D. Lewis, C. Riggs, and M. R. Grimaila, "Post-quantum cryptography: What advancements in quantum computing mean for IT professionals," *IT Professional*, vol. 18, no. 5, pp. 42–47, 2016, doi: 10.1109/MITP.2016.77.
6. T. S. Humble and E. P. DeBenedictis, "Quantum realism," *Computer*, vol. 52, no. 6, pp. 13–17, Jun. 2019.
7. S. Stepney *et al.*, "Journeys in non-classical computation II: Initial journeys and waypoints," *Int. J. Parallel, Emergent, Distrib. Syst.*, vol. 21, no. 2, pp. 97–125, 2006, doi: 10.1080/17445760500355454.
8. R. F. Van den Brink, F. Phillipson, and N. M. J. C. T. Neumann, "Visual on next level quantum software tooling," in *Proc. 10th Int. Conf. Comput. Logics, Algebras, Program., Tools, Benchmarking*, 2019, pp. 16–23.
9. B. Boehm, "A view of 20th and 21st century software engineering," in *Proc. 28th Int. Conf. Softw. Eng.*, 2006, pp. 12–29, doi: 10.1145/1134285.1134288.
10. E. Dijkstra, "Structured programming," in *Classics in Software Engineering*. Upper Saddle River, NJ, USA: Yourdon Press, 1979, pp. 41–48.
11. B. W. Kernighan and D. M. Ritchie, *The C Programming Language*. 1st ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 1978, p. 274.
12. G. Booch, "The history of software engineering," *IEEE Softw.*, vol. 35, no. 5, pp. 108–114, Sep./Oct. 2018.
13. P. Bourque and R. E. Fairley, Eds., *Guide to the Software Engineering Body of Knowledge, Version 3.0*. Washington, D.C., USA: IEEE Comput. Soc., 2014.
14. M. Piattini *et al.*, "The Talavera Manifesto for quantum software engineering and programming," in *Proc. Quantum Softw. Eng. Program.*, 2020, pp. 1–5.
15. H. Thapliyal and E. Muñoz-Coreas, "Design of quantum computing circuits," *IT Professional*, vol. 21, no. 6, pp. 22–26, 2019, doi: 10.1109/MITP.2019.2943134.
16. L. Zhou, N. Yu, and M. Ying, "An applied quantum Hoare logic," in *Proc. 40th ACM SIGPLAN Conf. Program. Lang. Des. Implementation*, 2019, pp. 1149–1162, doi: 10.1145/3314221.3314584.
17. brilliant.org, Quantum Comput*ing* Course. 2020. [Online]. Available: https://brilliant.org/courses/quantum-computing/

MARIO PIATTINI is currently a full professor of software engineering with the University of Castilla—La Mancha, Ciudad Real, Spain, a leader of the Alarcos Research Group, Ciudad Real, and the scientific director of AQCLab, S.L., Ciudad Real (first ENAC / ILAC accredited laboratory for software and data quality based on ISO 25000). He received the M.Sc. and Ph.D. degrees in computer science from Madrid Technical University (UPM), Madrid, Spain, in 1989 and 1994, respectively. Contact him at Mario.Piattini@uclm.es.

MANUEL A. SERRANO has been an associate professor with the University of Castilla—La Mancha, Ciudad Real, Spain, since 2000. He is also a vice-dean of the Department of Technologies and Information Systems. He received the M.Sc. and Ph.D. degrees in computer science from the University of Castilla—La Mancha. Contact him at Manuel.Serrano@uclm.es.

RICARDO PEREZ-CASTILLO is currently with the IT & Social Sciences School of Talavera, University of Castilla—La Mancha, Ciudad Real, Spain. He received the Ph.D. degree in computer science from the University of Castilla—La Mancha. He is the corresponding author of this article. Contact him at ricardo.pdelcastillo@uclm.es.

GUIDO PETERSEN is currently the director of R&D and Software Solutions of Alhambra-Eidos, Madrid, Spain, a leader of aQuantum, Madrid, a research, development, and consulting group in quantum software engineering and programming, and the operations director of M2i Formación, Madrid. Contact him at guido.peterssen@a-e.es.

JOSE L. HEVIA is currently the software architect, software solutions IT manager, and a quantum technology team leader with Alhambra-Eidos, Madrid, Spain. He has more than 25 years of experience in consulting, design of HA-FT Enterprise multilayer solutions and technical training, using state-of-the-art technologies. Contact him at jluis.hevia@a-e.es.