Neurocomputing 538 (2023) 126199

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Let's do it right the first time: Survey on security concerns in the way to quantum software engineering

Danel Arias [a,*], Ignacio García Rodríguez de Guzmán [b], Moisés Rodríguez [b], Erik B. Terres [a], Borja Sanz [a], José Gaviria de la Puerta [a], Iker Pastor [a], Agustín Zubillaga [c], Pablo García Bringas [a]

[a] University of Deusto, Avda. de las Universidades, 24, Bilbao 48007, Spain
[b] Universidad de Castilla-La Mancha, Rectorado UCLM, C. Altagracia, 50, Ciudad Real 13001, Spain
[c] Vicomtech, Mikeletegi Pasealekua, 57, San Sebastián 20009, Spain

ARTICLE INFO

Article history:
Received 26 July 2022
Revised 15 December 2022
Accepted 28 March 2023
Available online 31 March 2023

Keywords:
Quantum computing
Quantum software engineering
Quantum software testing
Quantum program security

ABSTRACT

Quantum computing is no longer a promise of the future but a rapidly evolving reality. Advances in quantum hardware are making it possible to make tangible a computational reality that until now was only theoretical. The proof of this is that development languages and platforms are appearing that bring physical principles closer to developers, making it feasible to begin to propose, in different areas of society, solutions to problems that until now were unsolvable. However, security vulnerabilities are also emerging that could hinder the progress of quantum computing, as well as its transition and development in industry. For this reason, this article proposes a review of some of the first artefacts that are emerging in the field of quantum computing. From this analysis, we begin to identify possible security issues that could become potential vulnerabilities in the quantum software of tomorrow. Likewise, and following the experience in classical software development, the testing technique is analysed as a possible candidate for improving security in quantum software development. Following the principles of Quantum Software Engineering, we are aware of the lack of tools, techniques and knowledge necessary to guarantee the development of quantum software in the immediate future. Therefore, this article aims to offer some first clues on what would be a roadmap to guarantee secure quantum software development.

1. Introduction

Over the last few decades, more and more is heard about how the new quantum computing paradigm will revolutionize the current state of our society due to its impact on such crucial areas as chemical, defence, pharmacy, medical, industry, and so on. Quantum computing technology is currently an evolving technology with solid fundamentals: (i) physics and computing theory is almost developed and put into practice on (ii) increasing production of quantum computers which are now in a continuous cycle of evolution and improvement, with the power to approach the "quantum supremacy" level (a moment "when a universal quantum computer performs a computational task that is beyond the capability of any classical computer") [1].

The prospects for quantum computing are indeed exciting, and extraordinary expectations are now fuelling a global effort to perfect quantum computing [2]. The most important companies (Google, IBM, Microsoft, Intel, Atos, Alibaba, etc.) are investigating how to make the most of this new technology in their businesses. Also, many countries (China, U.S., Japan, Russia, U.K., etc.) are investing huge quantities of money in quantum technology. As evidenced by the introduction of the National Quantum Initiative Act in the United States, the funding of the Institute for Quantum Computing by the Canadian government, or the European Union's "Quantum Manifesto and Quantum Technologies Flagship" initiative, the involvement of governments is of paramount importance.

A more than significant advance has been made in hardware, and there are already quantum computers available and ready to use (IBMQ, IonQ, D-Wave, Microsoft Quantum, Google Quantum, Honeywell, etc.). However, despite of the computational advantage that computing hardware represents, it is quantum software that will benefit the most from that hardware and solve those problems that are currently out of the scope of classic computing. There are already some quantum algorithms that exploit the quantum characteristics of these computers, such as Shor [3], Grover [2],

* Corresponding author.
E-mail addresses: danel.arias@opendeusto.es (D. Arias), ignacio.grodriguez@uclm.es (I. García Rodríguez de Guzmán), moises.rodriguez@uclm.es (M. Rodríguez), e.terres@deusto.es (E.B. Terres), jgaviria@deusto.es (J. Gaviria de la Puerta), iker.pastor@deusto.es (I. Pastor), azubillaga@vicomtech.org (A. Zubillaga), pablo.garcia.bringas@deusto.es (P. García Bringas).

https://doi.org/10.1016/j.neucom.2023.03.060
0925-2312/

Deutsch–Jozsa [4], or Teleportation [5], but many others are to be developed to solve current problems out of the scope of classic computers.

In recent years, there have been many proposals for quantum languages, software development kits (Forest [6], Qiskit [7], QDK [8], Orquestra [8], Cirq) and platforms (Quantum Inspire [9], IQ Experience, Quantum Playground [10], Forge [11], LIQUi|⟩ [12]), but quantum computing is still under development.

As The Talavera Manifesto for Quantum Software Engineering and Programming [13] states: "Given the recent rapid advances in quantum hardware, it is urgent that we step up our efforts in quantum software", stressing the relevance of quantum software, and reaffirming the importance of ensuring the quality and security of quantum software. This manifesto has subsequently been signed by some four hundred researchers worldwide. It is necessary to go a step further and raise awareness of the need for Quantum Software Engineering (QSE) that will enable the production of quantum software at the right quality and productivity levels [7]. As [11] states "software engineers should start the process of bringing SE (Software Engineering) practices into the domain of QCs (Quantum Computers)". The problem comes when we realise that in order to boost large-scale production of quantum software, an adequate quality level is required [13], so that society can really benefit from the promising quantum applications that exist in the various domains. It is important to consider that in a quantum information system, there are several factors that influence the quality of the results [4]: the quality of the quantum hardware, the quality of the quantum software platform (development and operational) and the quality of the quantum software itself [14].

Therefore, our aim in this paper is to set the focus on an unexplored area of Quantum Software Engineering: the security concerns that could potentially affect quantum software. Although software security is a broad concept that may involve many areas of software development, the scope that we propose in this paper states the first of the steps that should be considered on the generation of techniques and tools intended to develop quantum software programs with an acceptable security level (as it is done with classic software, and is of concern to designers, developers and users).

Security is an important consideration when deploying any form of software product or service. The code is executed for a purpose and if the code is well-written, it should reach that purpose. Robin Miller provided the following slogan: a well-typed program cannot "go wrong" [15]. The appropriate formalization of this slogan depends on the style of formal semantics used for a particular language. In the case of quantum programming languages is not different. That is why it is critical to focus on a certain type of quantum programming language (QPL) so that an in-depth and targeted study may be conducted.

So, it is desirable to reach a maturity point in quantum software development when "it integrates software security concepts in a software engineering design course" [16] (as classic software development states and puts into practice); we are now however starting from scratch. First of all, current vulnerabilities and security issues need to be identified from the available material for quantum software development (i.e. quantum programming languages, current and most popular quantum algorithms and quantum hardware), and techniques such as testing should be explored as one of the best drivers to identify security problems in existing quantum software [17–19], exploiting the current testing techniques that are now emerging [20,21]. Once all this valuable knowledge is built, model-based, design-patterns and architectures could be proposed in order to improve (or create) techniques and tools to develop quantum secure software in a sustainable way.

The organization of the paper is as follows: Section 2 presents a brief introduction to the main concepts of quantum software regarding quantum circuits; Sections 3 and 4 present a brief analysis of the most important quantum programming languages and quantum algorithms, which are the basis to start analysing the potential security threats; Section 5 presents to what extent, current hardware limitations could affect quantum software security; Section 6 shows to what extent testing techniques could be helpful to identify security issues in quantum software code; Section 7 outlines some conclusions and presents the next steps to be carried out in the context of this research.

2. Introduction to quantum states and quantum operators

A quantum circuit is both a visual representation of the steps required to perform a quantum calculus and a high-level view of a quantum program; in fact, a quantum circuit can be translated into a quantum program and vice versa. The circuit consists of a set of horizontal lines, each one representing the qubit that is operated by the gates placed on that very same wire. Thus, quantum gates affecting a qubit are drawn on its corresponding wire.

Fig. 1 shows an excerpt from the Bell Inequality Test circuit (taken from https://algassert.com/quirk). It has five qubits; each one being affected by the gates placed on their respective wires. The first qubit, for example, is affected by a Hadamard gate on the first run and then a CNOT gate is applied to qubits 1 and 5.

Besides the "operative" gates (which perform a modification on the qubit related to the desired calculus), the circuit also has a set of measurement gates: these collapse the qubit and allow its state to be read as a classical, two-states, bit. Measuring a qubit is irreversible (i.e., the qubit cannot be returned to its previous state).

Although not measured, the qubit is always in a state that is often represented as a particle placed in a given position – x, y, z – of the Bloch Sphere (Fig. 2).

Thus, a particle placed in the North pole of the sphere is in the state $|0\rangle$. Rotating it through the Z axis moves it to the South pole, which corresponds to the $|1\rangle$ state. The changes in the state are performed by the quantum gates, which can be represented as matrixes. The process of applying a gate to a qubit consists in calculating the product of the matrix that represents the input qubit (or qubits) by the gate's matrix. As an example, Eq. 1 describes the application of the Hadamard gate to a qubit in state $|0\rangle$. H rotates the qubit particle π radians over X and $\pi/2$ radians over Y.

$$\mathrm{H}|0\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\ 0\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\ 1\end{pmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{1}$$

The end result of the Eq. 1, $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, is said to be in a superposition state, since it is representing neither the state $|0\rangle$ nor the state $|1\rangle$ alone, but a combination of the two states probabilistically. Measuring the state would collapse it to one of them with a probability based on the square of their coefficient, in this case, $\frac{1}{2}$.

If we had more qubits, the entanglement phenomenon could occur, which happends when a quantum state cannot be separated into a product of individual qubit states. For example the state $\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$ is separable and equal to

$$\left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right)\left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right)$$

On the other hand, the state

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

is not separable, and hence, it is an entangled state. This state "entangles" the qubits in the sence that a measurement in one qubit gives us information about the other qubit. In this case, if we were

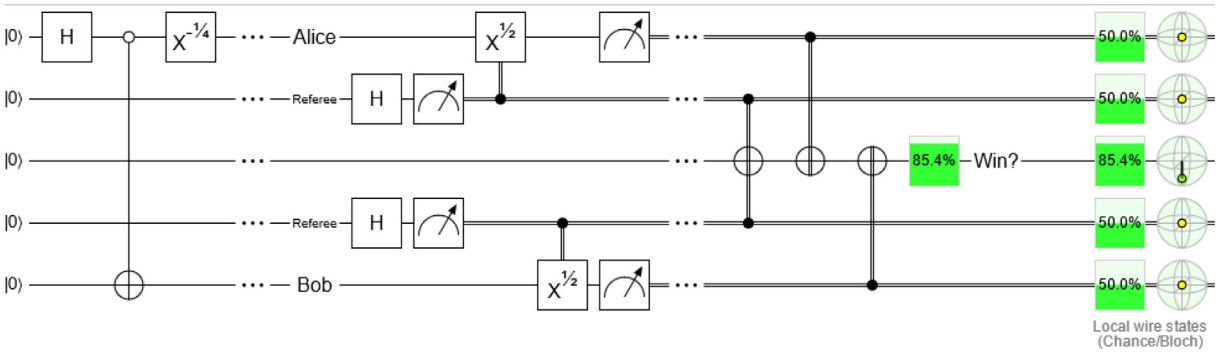

Fig. 1. An excerpt of a quantum circuit.

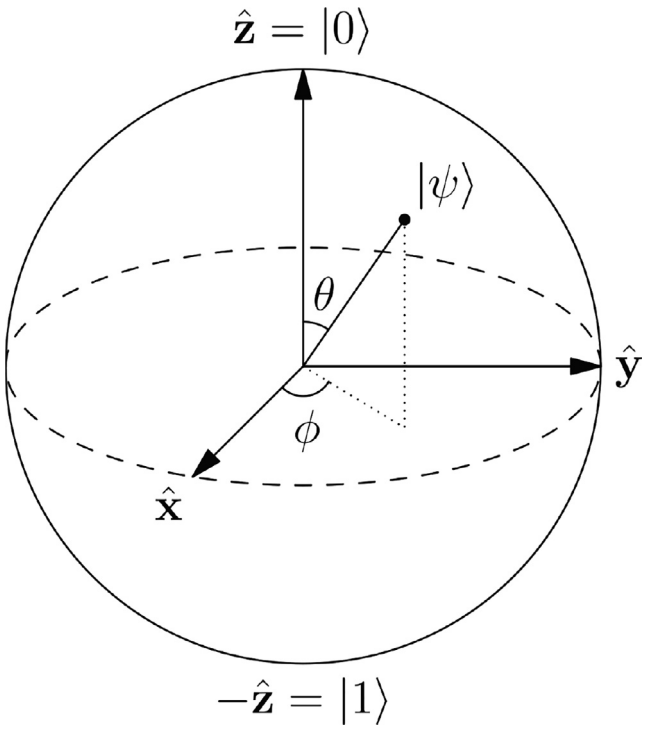

Fig. 2. Qubit representation with a Bloch Sphere.

to measure the first qubit and we read a zero, we would know that the only possible state for the second qubit would be a zero.

Some of the differences between quantum circuits and classical ones are that in quantum circuits loops are not allowed, wires cannot be joined together, and multiple outputs cannot be obtained from a single input.

It is important to consider that a quantum circuit is a high-level view of a quantum program that could be implemented in any particular platform, given that it is possible to translate any quantum circuit to a quantum program.

There is a set of primitive quantum gates that perform relatively simple operations on qubits. All the gates modify the qubit state by changing the position of its associated particle in the Bloch Sphere. Just as it is possible to call a subroutine in classical computing, it is also possible to integrate one predefined quantum circuit in another one, so handling it as if it was a primitive gate.

3. Quantum programming languages (QPL)

In this section we make a summary of the quantum programming languages that currently exist and we make a brief analysis of possible dangers that may exist or that we believe could come up. Since QPLs are the heart of quantum software, a brief analysis is important to understand future security concerns.

Quantum languages, as the classical ones, can be categorized into 2 groups: imperative and functional languages. Nonetheless, the latter is not strictly true as there are language proposals that do not fall within this classification, such as the work of Freedman, Kitaev, and Wang [22] that proposes a radically different direction in the semantics of quantum computation exploiting connections between quantum computation and topological quantum field theories (TQFTs).

Since the creation of the first functional quantum language (QFC [23]), much progress has been made in the development of this programming paradigm and new languages have been proposed (QML [24], LIQUi|⟩ and Quipper [25]). Moreover, the improvements of the linear lambda calculus for quantum computation described by van Tonder have lead to a better structured functional logic. [26,27] Still, the most advanced and recognized are the imperative languages for their involvement in the hardware model.

3.1. Imperative languages

The imperative language development thread was started by Knill (1996) [28] who proposed a set of conventions for quantum pseudo-code that would be implemented in a QRAM (quantum random access machine). Knill's proposal significantly influenced future imperative language developments, although the proposed pseudo-code is not precise enough to be implemented in the form of a language directly.

3.1.1. QRAM model

Since this model is the basis for subsequent languages, it is necessary to mention how this works. The QRAM is made up of a quantum memory and a quantum ALU (qALU), which in turn is made up of quantum logic gates. This model is controlled by classical instructions, which gives rise to a hybrid system in which the classical machine provides the code for quantum operations to the quantum subsystem and this, as a black box, returns the results after the operations, acting with a slave role. The role of the quantum subsystem is to decode the classical instructions and carry out the pertinent operations through quantum gates at the indicated qubits (see Fig. 3).

The development of quantum languages is not easy because although certain specific phenomena of quantum particles such as superposition or entanglement are the basis of the power of quantum computing, they present a problem when it comes to reaching the level of abstraction necessary for a high-level programming language. For this reason, the most developed QPLs present a way of defining quantum circuits for a quantum algorithm to be described by deconstructing it into instructions of the QASM

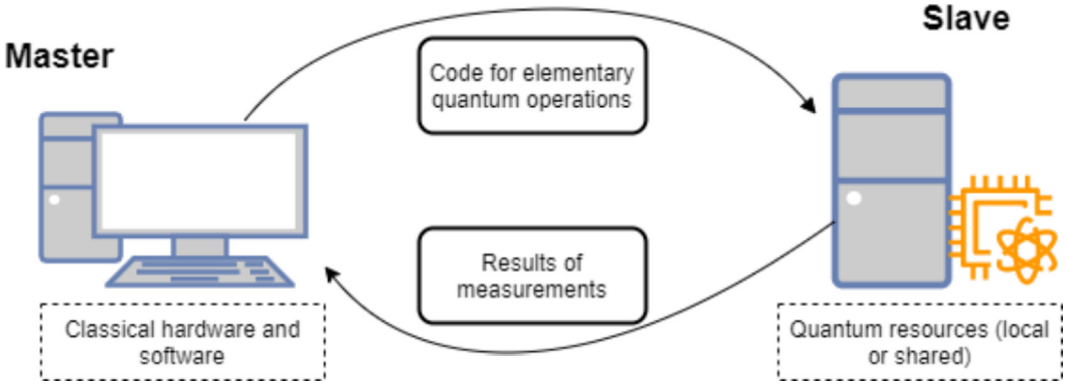

Fig. 3. Simplified scheme of a QRAM machine. The classical hardware drives the quantum resources in a master–slave configuration.

language [29] that then will be sent to the QRAM. The definition of these circuits is done through a succession of quantum gates. Instruction by instruction it is indicated, sequentially, which gate would apply to each qubit or qubits. Measurements are indicated this way as well. In the end, the circuit is defined and ready to be executed in the hardware.

3.1.2. QPLs currently

During the years following Knill's proposal, Ömer (2002, 2003) [30,31] developed what is considered the first quantum programming language, QCL, which has a syntax based on C. Bettelli et al. (2003) [32] proposed a high-level language based on C++ that was implemented as a library for the latter. Subsequently, Sanders & Zuliani [33] explored non-deterministic and mixed-state programming with the qGCL language. All these languages were the basis of the languages that have proliferated today and the precursors in their development. Some examples are Q# [34], Q|SI⟩ [35], Qiskit[36] and Scaffold [37].

Microsoft's Q# and IBM's Qiskit are two of the most advanced and widely used QPLs based on the QRAM model today. These languages, however, are still in their early stages of development; Qiskit is now at version 0.36, and Microsoft Quantum Development Kit is at version 0.24. Both languages can be called extensions of classical languages since they are not only based on these, but Qiskit, for example, works as a Python library. On the other hand, Q# is based on F#, C# and Python. This foundation in classical languages makes sense because they will be used to define the behaviour of our quantum circuit, as the quantum behaviour of the hardware is not directly applied to it.

3.2. Potential threads

When analysing quantum programming languages, we have observed two main types of problems: base language problems and QLP implementation problems.

The base language problems are those security problems that arise based on the characteristics of the language on which the implementation of quantum software is done. Python's intrinsic problems are inherited by Qiskit as it is implemented on top of it. On the other hand, the quantum language Q# has its own compiler, which cannot be assured to be fully secure. The implementation may contain certain classical bugs, which could be exploited in order to perform classical or even quantum attacks. It is crucial that compiler developers ensure the security of their software before releasing production-ready public versions.

This type of security concerns could include bufferoverflows, executed in a classical way, such as trying to maliciously compile specific circuits to abuse the exponential character of quantum circuits to overflow the memory. Another security concern that we believe may arise in the future stems from the lack of precision of numerical values in classical computers. When trying to encode an input value using methods such as Amplitude Encoding[38], the number of state coefficients grows exponentially, while their sum of squares remains fixed at one. This can cause some information to be lost due to precision errors, which can lead to unexpected numerical or computational errors.

On the other hand, those languages implemented as frameworks on top of other languages need to be carefully considered with regard to the dependencies they use. In May 2022, a vulnerability was detected in some versions of log4j that allowed malicious agents to execute code remotely.[39] This type of incident could happen again in dependencies from libraries such as Qiskit or Cirq. Making use of updated versions of the software is a must for every software developer, both classical and quantum.

QLP implementation problems refer to implementations bugs that arise from the libraries themselves, such as poorly implemented algorithms. In the quantum libraries case, such bugs may arise from misplaced quantum gates, mathematical errors in simulations, or mismeasured results. These bugs, derived from human errors, may exist and not have been detected by the library developers' tests. They could be abused by malicious agents to obtain unwanted outputs from our circuits and exploit them. On the other hand, depending on the severity of these bugs in our quantum libraries, we could receive serious classical attacks to our software, so, as mentioned in the previous paragraph, it is essential to update the quantum frameworks to its latest versions.

4. Quantum algorithms

As shown in Section 3, the programming of these quantum computers is done by defining specific concrete quantum circuits. Since the quantum computer works as a black box for the classical part, the circuits are defined to apply the gates for the corresponding qubits according to the algorithm and it ends with a measurement to obtain the expected specific result. These quantum algorithms are previously defined theoretically and then programmed by defining the corresponding quantum circuit.

Due to this theoretical background, most of the current quantum algorithms do not have vulnerabilities by themselves. The attacks or security flaws that this type of circuits could have mainly arise from physical attacks or implementation errors. In this sense, it could be understood that most of the current quantum algorithms are as secure as other classical algorithms that only follow a mathematical background, such as the Dijkstra's algorithm or the Euclidean algorithm. In this section we will make an exposition of quantum algorithms which currently are vulnerable to attacks, for which quantum software developers will have to implement security measures.

4.1. Quantum teleportation and superdense coding

Quantum teleportation [40] is a technique that is used to transport an arbitrary quantum state between two distant observers A and B, even in the absence of a quantum communications channel linking the sender of the quantum state to the recipient. To perform this, the technique builds on the quantum entanglement.

Superdense Coding (SDC) is a quantum communication protocol to communicate a number of classical bits of information by only transmitting a smaller number of qubits, under the assumption of sender and receiver pre-sharing an entangled resource [41,42]. Superdense coding is the core premise of secure quantum secret coding. Eavesdroppers are less likely to intercept transmissions since both qubits are required to decode the information being conveyed [43].

Because quantum teleportation and superdense coding are so closely related, it is important to distinguish the two of them. Quantum teleportation is a method of transmitting the quantum state of a qubit from one location to another using two bits of classical communication and a Bell pair. In other words, it is a protocol that destroys the quantum state of a qubit in one location and recreates it on a qubit at a distant location, with the help of shared entanglement. Superdense coding, on the other hand, is a method of sending two classical bits to another party with only a single qubit of communication.

Both algorithms are vulnerable to physical attacks, where malicious actors could induce noise to alter the states which are being transmitted or to drop the information[44,45]. To reduce physical attacks in an environment where information is not as easy to send as in classical computing, protection techniques are mandatory. Either asserting that the connection is not being attacked by sending some test quantum state in order to validate it or implementing Error Correcting Techniques must be tools in every quantum software developer's portfolio.

4.2. Quantum key distribution

QKD is a secure communication protocol which implements a cryptographic protocol involving components of quantum mechanics. It enables two parties to produce a shared random secret key known only to them, which can then be used to encrypt and decrypt messages. Although theoretically secure, several papers have pointed out serious security flaws such as man-in-middle attacks [46], time-shift attacks [47], faked states attacks [48] or wavelength-dependent attacks [49]. Most of this attacks are hardware-based, but a good software implementation could help to detect some of these attacks. Quantum software developers should take these attacks into account when designing testing and assertion patterns for their software.

4.3. Algorithms for quantum machine learning

The ability of quantum states to be in superposition can thereby lead to a substantial speedup of a computation in terms of complexity since operations can be executed on many states at the same time, which is why people consider the possibilities of the combination of quantum computing and machine learning. This way, Quantum Machine Learning has become one of the most researched branches of Quantum computation. There have been some survey papers which mainly overview general ideas of different machine learning algorithms in quantum version [50,51]. The most important and influential algorithms used as subroutines in QML algorithms are Grover's algorithm, the Swap-test, QPE and the HHL algorithm [52]. HHL is used in Quantum Support Vector Machine (QSVM) [53] algorithm and in the 2 main Quantum Dimensionality Reduction algorithms: Quantum principal component analysis (QPCA) and Quantum linear discriminant analysis (QLDA). The rest, however, are used in the Quantum K-means Clustering. Not to forget that neural networks have also been in the spotlight of the search for their quantum counterparts [54,55].

Unlike previous algorithms, which relied on a mathematical basis, quantum machine learning has to deal with parameterized circuits that change throughout the learning phase. This uncertainty about the circuit implies greater difficulty in testing the validity of the implementation. Although this limitation also occurs in classical machine learning algorithms, the quantum world has physical limitations that could affect the correct operation of the software. The decoherence effect, in which states will become zeros once enough time has elpased, would imply a bias towards certain categories.[56] This bias, while not related to computer security, does affect social security, as measurements from a biased network can affect real-world use cases with erroneous predictions.

Adversarial attacks are also a major struggle in the field of quantum neural networks.[57,58] These attacks consist in injecting pre-calculated data into the input values in order to confuse the network into predicting incorrect values. Quantum neural networks are no better than classical ones in robustness, which implies having to deal with possible classical adversarial attacks and also physical adversarial attacks, where an agent would try to deteriorate the qubits state to modify the prediction. At present, there are no papers examining physical quantum adversarial attacks, but they could be a major problem if they become feasible. New architectures or techniques are needed to improve robustness.

A last but not least important topic is explicability on QML. Explicability is the field of machine learning that studies how we can infer the logic behind black models by obtaining details or reasons that are clear or easy to understand.[59] There is no work on this topic at present, which mean that our understanding of quantum neural networks is that they are completely black boxes. New techniques will be needed to test whether quantum neural networks are working as intended and to improve resilience against adversarial attacks.

5. Limitations of the hardware

As mentioned in the introduction, several new hardware developments are underway. However, the current state of the art in terms of quantum hardware still suffers some core shortcomings, which include a small number of qubits, restrictive connectivity, or limited native instructions. Moreover, compared to classical bits, quantum bits (qubits) are much more fragile, unstable, error-prone and deciduous. Quantum circuits are, of course, affected by these errors, which keeps the quality of the generated outputs in low levels, and it does imply high levels of expert supervision. These devices are termed as noisy intermediate-scale quantum (NISQ) computers [60,61].

All these flaws can lead to faulty software if not accounted for. An unstable set of qubits passing through a deep enough circuit will result in erroneous measurements, which, if undetected, can lead to bugs in production code. Since most quantum algorithm circuits are by the quantum programming languages themselves, coders will have to assert that those implementation will run on the quantum computer of their choice with enough stability to yield correct measurements.

Based on this premises, it is key to identify the errors and their causes:

1. Quantum bits suffer from a short lifetime as well as unstable gates and measure operations. On the one hand, the relaxation problem implies that a qubit in state $|1\rangle$ spontaneously can lose energy and end with state $|0\rangle$. Besides, the de-phasing phenomena can produce a similar effect. On the other hand, measurements and gate dependencies can suffer alterations when read out, even accumulating errors in parallel operations.[62]
2. Limited native set of (small) gates, so every quantum program needs to be decomposed in a relatively high number of gates, which increases the complexity of the circuit, the need for a long life of the qubits, and -in general- the probability of error.
3. Coupling constraint, and limited connectivity of the qubits, which introduces the need to provide additional connection instructions (swap operations), of course increasing the runtime and the number of gates of the quantum program, and -again- the possibility of error.
4. Quantum hardware is far for being a home commodity due to its high cost and needs (ultra-cold temperature [63,64], shielded environment, and complex wiring for control). Therefore, cloud-based access to quantum computers is the logical path forward where the hardware is hosted in a remote location. This infrastructure comes with its own problems, as running quantum code in the cloud may involve waiting for a queue of jobs from other developers to access the computer, as is currently the case with IBM systems. Testing and debugging code, as well as ensuring stability, will not be as fast as it is currently with classical programs, so good frameworks and programming paradigms are needed.
5. The specific way of the encoding of a quantum circuit can provide enough clues to infer sensitive information of the problem.
6. Given the essential need of optimization that quantum circuits have [65], it is also mandatory to consider third party compilers that are able to perform this improvement of the circuits. These compilers should be taken into consideration in a trusted way; deficiencies or shortcomings in the compilers would open the door of the quantum program to tamper and exploit attacks, and also to reverse engineering.

Therefore, noise-resilient methods are of paramount importance. Due to this, besides of engineering better qubits, several other independent approaches like parameter optimization in variational algorithms, noise-aware mapping, and scheduling are pursued to make quantum programs resilient to noise. The theory of Quantum Error Correction (QEC) [41,66–69] enables fault tolerant computation [70,71] using redundant qubits, but full fault tolerance is still too expensive for current noisy devices. However, much effort is being made to implement feasible countermeasures (listed in [72]) directly into the hardware or to the compiler that might reduce error occurrence.

Even a well-typed program might be affected by these errors turning it into unsafe code. Some of these problems, at the stage where QPLs are today, could be handled by the programmer or, at least, be considered in an analysis of the program. This enforces the idea of either a prior analysis of the quantum code or, if possible, a dynamic analysis in which the undesired behaviour of the software due to a noise error can be detected.

6. Quantum software testing

In this section, a brief analysis of quantum software testing techniques is presented. Software testing is a powerful tool for asserting the correct workflow of code, and some classic techniques can be extrapolated to quantum code to ensure both security and quality. Based on a recent review of the software testing state of the art, we extend its analysis to add security concerns and implications of testing in quantum software in the security domain.[73].

6.1. Dealing with quantum bugs

Bug patterns are incorrect code idioms or poor coding techniques that have been repeatedly shown to fail. They are typically brought on by a misunderstanding of the features of a programming language, the use of incorrect design patterns, or simple errors that have similar behaviours.

6.1.1. Bug types and patterns

In order to support the debugging of quantum software, Huang and Martonosi [74,75] investigated the different bug types for specific quantum programs. They identified various bug types unique to quantum software and provided defence measures for each type of bug based on their experiences building some quantum algorithms. These bug categories include incorrect classical input parameters, incorrect deallocation of qubits, incorrect quantum initial values, incorrect operations and transformations, incorrect composition of operations using iteration, incorrect composition of operations using recursion and incorrect composition of operations using mirroring. They also suggested some bug-specific defence tactics.

In order to give readers a clear understanding of the different types of bugs that might appear in quantum programs and how to spot them, Zhao et al. [76] identified and categorized various bug patterns in the quantum programming language Qiskit. Their research on bug patterns primarily focuses on the symptoms, underlying causes, treatments, and preventions of bug patterns. They also give an example of the pattern's symptoms for each bug pattern. Finding these similarities in a quantum programming language can increase programmers' productivity in locating problems and lower the cost of software maintenance. Despite the fact that this research focuses on Qiskit, many of these issues can be applied to other languages based on gates because, despite the differing syntax, they are similar in most aspects.

6.1.2. Bug benchmarks

Campos and Souto [77] proposed QBugs, which is a collection of reproducible bugs in quantum algorithms for supporting controlled experiments for quantum software debugging and testing. In order to help the evaluation and comparison of new research as well as the reproducibility of published research findings on quantum software engineering, QBugs offers some preliminary suggestions for developing a benchmark. Zhao et al. [78] proposed Bugs4Q, a benchmark of thirty-six real Qiskit bugs that had been manually verified in addition to test cases for simulating bug-related behaviour. Bugs4Q facilitates downloading and running test cases for quantum software testing and gathers repeatable problems in Qiskit programs. The fixes for every actual bug are made accessible for public investigation. For the purpose of categorizing existing bugs and experimentally evaluating isolated bugs, Bugs4Q offers a database that contains an analysis of bug types.

6.2. Testing approaches for quantum software

Testing is the process that executes a program with the intent to find errors, which is a critical process for supporting quality and security assurance during software development. In this section, we review state of the art of testing quantum software.

1. Fuzz-testing. Fuzz testing is a testing technique that inputs invalid or random data called fuzz into the software system to discover coding errors and security loopholes. Wang et al. [79] adapted the existing classical technique coverage-guided

fuzzing (CGF) to test quantum software. They proposed QuanFuzz, a search-based test input generator for quantum software. The basic idea of QuanFuzz is to define some quantum sensitive information to evaluate the test inputs for quantum software and use a matrix generator to generate test cases with higher coverage. Fuzz testing would allow the developer to find possible inputs that could lead to unexpected outputs which, in addition, could affect the classical section of the program inducing bufferoverflows or bad error handling.
2. Property-based testing. Property-based testing uses specifications of essential properties to produce testing criteria and procedures which focus on these properties in a systematic manner. It has been shown to be a promise tool for generating test cases that reveal program faults in classical software. As an example, Honarvar et al. [80] presented QSharpCheck, a property-based testing approach for quantum software written in Q#.
3. Search-based testing. Search-Based Software Testing (SBST) is the use of optimizing search techniques, to solve problems in software testing. Recently, SBST is showing promising in classical software testing, such as test case generation, test case prioritization, and test suite minimization. Wang et al. [81] proposed a search-based approach, called QuSBT (Quantum Search-Based Testing), to test quantum programs.

6.3. Debugging quantum programs

As with any programming environment, debugging methodologies and techniques are essential for quantum software security. This section provides an overview of four debugging techniques for quantum programs, which allow the developer to analyse existing bugs, problems in the code and possible edge cases for certain programs.

6.3.1. Debugging quantum processes

It is challenging to debug quantum programs through program monitoring because measuring a quantum system could result in its state collapsing. Li and Ying [82] proposed a method to debug quantum processes by observing measurements as a solution to this issue. The basic idea of the approach is to create a protocol that will enable monitoring-based debugging of quantum processes without affecting their current states. This debugging approach, however, can only handle the debugging of a subset of quantum processes and the coverage to all of them is still an open problem [82].

6.3.2. Assertion-based debugging

Several assertion-based approaches have recently been proposed for debugging quantum programs. One approach is the proposal by Huang and Martonosi [75] which focuses on carrying out a statistical study of the quantum program. They begin by defining three categories of assertions: classical state, superposition, and entanglement, with their respective operating hypotheses. Multiple measures are used in these statistical tests to determine if the hypothesis should be rejected. With enough measurements, a statistical test can determine if an assertion is not true, indicating an error in the program. However, as stated in the paper, these operations are exceedingly costly and thus cannot be employed as effective assertion checks with this method.

It was found that a crucial flaw in Huang and Martonosi's assertion-based approach was that each measurement made during debugging required stopping the program, and that assertions required aggregates of runs to measure the actual results of the computation. To overcome this, the proposal of Liu et al. [83] is to implement certain ancilla qubits (some additional qubits) that could be measured without interrupting the execution by indirectly verifying the desired condition regarding the qubits under test. Thus, they propose certain quantum circuits that could perform dynamic assertion not only for classical values but also for superposition and entanglement states. Subsequently, after the measurement of those ancilla qubits, Quantum Phase Estimation (QPE) (explained in Section 4) is carried out. This algorithm estimates the value of the qubit. This approach is predicated on the assumption that these quantum circuits are error-free, as faults could damage not only the measurement but also the qubits under test.

6.3.3. Language support for debugging

Although the work on tackling bugs in quantum programs through assertion checking shown in Section 6.3.2 is encouraging, it usually involves checking assertions dynamically during runtime, which could waste the quantum computing (or simulation) resources. To address this, Singhal and Reppy [84,85] proposed encoding assertions into a static type of system of a quantum programming language to support programmers in writing correct programs from the beginning. This allows programmers to specify some of the semantic properties they want their programs to have in their code, which a data-type checker can be used to verify if some of those properties are correct at compilation time.

6.4. Analysis

Recently, a number of studies for program analysis of quantum software have been conducted, which can help with bug discovery for quantum systems.

1. Entanglement analysis. ScaffCC [86] is a compiler for the Scaffold language [37] that integrates code analysis tools. This compiler receives the code in Scaffold as input and returns the representation in quantum assembly (QASM) [87] as output. One analysis that ScaffCC supports is called entanglement analysis, which can conservatively identify each possible pair of qubits that might be entangled in the program. Such entangle information can help a programmer assert that the output of the program is correct and that phase decoherence does not separate an entangled state, which could lead to an unexpected (and possibly harmful) output.
2. Robustness analysis. Hung et al. [88] presented a semantics for describing quantum computation with errors and an analysis that bounds the distance between the result of a noisy program and its corresponding ideal program on the same input. This analysis could actually be useful in asserting that a quantum computer is not being physically attacked to increase noise in the environment in order to produce noisy outputs. The analysis could be run periodically or before each batch of runs.

7. Conclusions

Quantum computing is a computational paradigm that is here to stay, offering society many opportunities for improvement through the possibility of tackling problems and challenges that, until now, could not be solved with classical computing [89]. However, the current level of development in its different fields (programming languages, platforms, hardware, quantum software engineering techniques) means that, among many potential problems, there is one in particular that should be of particular concern: security. The aspects that compromise security at the moment are:

1. Quantum hardware is advancing, however current quantum computers are very sensitive to "noises" or interferences [90] that affect the performance of algorithm calculations, and that could be exploited by attackers to create security breaches that, according to the industry, could be very critical.
2. There are problems in how the languages themselves are defined which, together with the current impossibility of generalising the classical methodology of code analysis, encourages the development of possible attacks based on the weaknesses of the programming languages themselves.

Thus, in light of the background presented, it is reasonable to propose a new research hypothesis: the study and experimentation on the state of the art in stability and security approaches from the conventional to the quantum realm. The aim would be to produce a Quantum Software Security Model useful to prevent the malicious impact of malware-related knowledge in quantum programs. These models (to be created) may, in the future, serve for the development of techniques and tools, as well as processes and knowledge, that facilitate secure quantum software development for engineers.

In particular, static code analysis would be essential to ensure safety during quantum software development, and as an analytical basis for structuring other dynamic analyses that could eventually be proposed when debugging the execution of quantum programs becomes feasible. Furthermore, given the nature of quantum software programming, the task of collecting and preparing an archive of debugging tools, known common bugs, stability monitoring methods and testing techniques is crucial. If implemented together in a library, they would pave the way for the definition of more complex programs and reduce security risks.

This article therefore presents a first identification of security issues present in the current available artefacts. This is the first step on a long road towards considering security in quantum software development as an aspect to be included in the software design itself, rather than a vertical aspect, by taking advantage of the existing knowledge on security in classical computing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors would like to give thanks to professors Mario Piattini, Ignacio García, Juan Manuel Murillo and José Manuel García, for their generosity and wealthy collaboration in the preparation of the Q-Serv 1.0 (Quantum Service Engineering: development, quality, testing & security of quantum microservices, PID2021-124054OB-C33) R&D project proposal. The authors would also like to acknowledge the partial finantial support by Ministry of Science (project QSERV-UD, PID2021-124054OB-C33), and also to the Basque Government (projects TRUSTIND - KK-2020/00054, and REMEDY - KK-2021/00091).

References

[1] A.W. Harrow, A. Montanaro, Quantum computational supremacy, Nature 549 (7671) (2017) 203–209, https://doi.org/10.1038/nature23458.
[2] L.K. Grover, A fast quantum mechanical algorithm for database search, in: STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing, pp. 212–219. doi: 10.1145/237814.237866.
[3] P.W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM J. Comput. 26 (5) (1997) 1484–1509, https://doi.org/10.1137/S0097539795293172.
[4] S. Gulde, M. Riebe, G.P.T. Lancaster, C. Becher, J. Eschner, H. Häffner, F. Schmidt-Kaler, I.L. Chuang, R. Blatt, Implementation of the deutsch–jozsa algorithm on an ion-trap quantum computer, Nature 421 (6918) (2003) 48–50, https://doi.org/10.1038/nature01336.
[5] D. Bouwmeester, J.-W. Pan, K. Mattle, M. Eibl, H. Weinfurter, A. Zeilinger, Experimental quantum teleportation, Nature 390 (6660) (1997) 575–579, https://doi.org/10.1038/37539.
[6] Rigetti, Welcome to the docs for the forest sdk!, Report (2019). https://pyquil-docs.rigetti.com/en/v2.7.2/.
[7] Qiskit: Open-source quantum development (2022) [cited accessed on 28 of November 2022]. https://qiskit.org.
[8] Q# y el kit de desarrollo de quantum (2022) [cited accessed on 28 of November 2022]. https://azure.microsoft.com/es-es/resources/development-kit/quantum-computing/#overview.
[9] The multi hardware quantum technology platform (2022) [cited accessed on 28 of November 2022]. https://www.quantum-inspire.com.
[10] Quantum computing playground (2022) [cited accessed on 28 of November 2022]. https://www.quantumplayground.net/#/home.
[11] Qcware force (2022) [cited accessed on 28 of November 2022]. https://forge.qcware.com.
[12] D. Wecker, K.M. Svore, Liqui—⟩: A software design architecture and domain-specific language for quantum computing, arXiv (2014). doi:10.48550/ARXIV.1402.4467.
[13] M. Piattini, M. Serrano, R. Perez-Castillo, G. Petersen, J.L. Hevia, Toward a quantum software engineering, IT Professional 23 (1) (2021) 62–66, https://doi.org/10.1109/MITP.2020.3019522.
[14] J. Verdugo, M. Rodríguez, M. Piattini, Software quality issues in quantum information systems.
[15] R. Milner, A theory of type polymorphism in programming, J. Comput. Syst. Sci. 17 (3) (1978) 348–375, https://doi.org/10.1016/0022-0000(78)90014-4.
[16] A.A.R. Angulo, X. Yang, Q. Niyaz, S. Paheding, A.Y. Javaid, A secure software engineering design framework for educational purpose, in: 2022 IEEE International Conference on Electro Information Technology (eIT), pp. 375–381. doi:10.1109/eIT53891.2022.9837112.
[17] S.M. Dhlamini, T. Mwakabaga, M.O. Kachienga, A holistic test procedure for security systems software: An experience report, in: AFRICON 2007, pp. 1–7. doi:10.1109/AFRCON.2007.4401471.
[18] G. McGraw, Software security: Building security, in: 2006 17th International Symposium on Software Reliability Engineering, pp. 6–6. doi:10.1109/ISSRE.2006.43.
[19] M.R. Stytz, S.B. Banks, Dynamic software security testing, IEEE Secur. Privacy 4 (3) (2006) 77–79, https://doi.org/10.1109/MSP.2006.64.
[20] A. García de la Barrera, I. García-Rodríguez de Guzmán, M. Polo, M. Piattini, Quantum software testing: State of the art, J. Software: Evol. Process (2021).
[21] A. García de la Barrera, M. Serrano, I. García-Rodríguez de Guzmán, M. Polo, M. Piattini, Automatic generation of test circuits for the verification of quantum deterministic algorithms, in: 1st International Workshop on Quantum Programming for Software Engineering (QP4SE 2022), pp. 1–6.
[22] M.H. Freedman, A. Kitaev, Z. Wang, Simulation of Topological Field Theories by Quantum Computers, Commun. Math. Phys. 227 (3) (2002) 587–603, https://doi.org/10.1007/s002200200635.
[23] P. Selinger, Towards a quantum programming language, Mathematical Structures in Computer Science 14 (4) (2004) 527–586, publisher: Cambridge University Press. doi:10.1017/S0960129504004256. https://www.cambridge.org/core/journals//mathematical-structures-in-computer-science/article/abs//towards-a-quantum-programming-language//54D5BCF28724CA6BE38F98DC4B6803DF.
[24] T. Altenkirch, J. Grattage, QML: Quantum data and control, Tech. rep. (2005).
[25] A.S. Green, P.L. Lumsdaine, N.J. Ross, P. Selinger, B. Valiron, Quipper, ACM SIGPLAN Notices 48 (6) (2013) 333–342, https://doi.org/10.1145/2499370.2462177.
[26] A. van Tonder, A Lambda Calculus for Quantum Computation, SIAM Journal on Computing 33 (5) (2004) 1109–1135, arXiv: quant-ph/0307150. doi:10.1137/S0097539703432165. http://arxiv.org/abs/quant-ph/0307150.
[27] A. van Tonder, M. Dorca, Quantum Computation, Categorical Semantics and Linear Logic, arXiv:quant-ph/0312174ArXiv: quant-ph/0312174 (May 2011). http://arxiv.org/abs/quant-ph/0312174.
[28] E. Knill, Conventions for quantum pseudocode, Tech. Rep. LA-UR-96-2724, Los Alamos National Lab. (LANL), Los Alamos, NM (United States) (Jun. 1996). doi:10.2172/366453. https://www.osti.gov/biblio/366453.
[29] K. Svore, A. Aho, A. Cross, I. Chuang, I. Markov, A layered software architecture for quantum computing design tools, Computer 39 (1) (2006) 74–83, conference Name: Computer. doi:10.1109/MC.2006.4.
[30] B. Ömer, Procedural Quantum Programming, AIP Conference Proceedings 627 (1) (2002) 276–285, publisher: American Institute of Physics. doi:10.1063/1.1503695. https://aip.scitation.org/doi/abs/10.1063/1.1503695.
[31] B. Ömer, Structured Quantum Programming (2003).
[32] S. Bettelli, T. Calarco, L. Serafini, Toward an architecture for quantum programming, Eur. Phys. J. D 25 (2) (2003) 181–200, https://doi.org/10.1140/epjd/e2003-00242-2.
[33] J.W. Sanders, P. Zuliani, Quantum Programming, Mathematics of Program Construction, Springer, Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 80–99.
[34] Azure Quantum Documentation, QDK & Q# API Reference. https://docs.microsoft.com/en-us/azure/quantum/.

[35] S. Liu, X. Wang, L. Zhou, J. Guan, Y. Li, Y. He, R. Duan, M. Ying, Q—Sl⟩) A Quantum Programming Environment, arXiv:1710.09500 [quant-ph]ArXiv: 1710.09500 (Oct. 2017). http://arxiv.org/abs/1710.09500.

[36] Qiskit documentation. https://qiskit.org/documentation/.

[37] A.J. Abhari, A. Faruque, M.J. Dousti, L. Svec, O. Catu, A. Chakrabati, C.-F. Chiang, S. Vanderwilt, J. Black, F. Chong, Scaffold: Quantum programming language (Tech. rep.), Princeton Univ NJ Dept of Computer Science, 2012.

[38] D. Ventura, T. Martinez, Initializing the amplitude distribution of a quantum state, Foundations Phys. Lett. 12 (6) (1999) 547–559, https://doi.org/10.1023/A:1021695125245.

[39] R. Hiesgen, M. Nawrocki, T.C. Schmidt, M. Wählisch, The race to the vulnerable: Measuring the log4j shell incident (2022). doi:10.48550/ARXIV.2205.02544.

[40] C.H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, W.K. Wootters, Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels, Phys. Rev. Lett. 70(13) (1993) 1895–1899, publisher: American Physical Society. doi:10.1103/PhysRevLett.70.1895.

[41] M.A. Nielsen, I.L. Chuang, Quantum Computation and Quantum Information: 10th Anniversary Edition, 10th ed., Cambridge University Press, USA, 2011.

[42] C.H. Bennett, S.J. Wiesner, Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states, Phys. Rev. Lett. 69 (20) (1992) 2881–2884, publisher: American Physical Society. doi:10.1103/PhysRevLett.69.2881.

[43] C. Wang, F.-G. Deng, Y.-S. Li, X.-S. Liu, G.L. Long, Quantum secure direct communication with high-dimension quantum superdense coding, Phys. Rev. A 71(4) (2005) 044305, publisher: American Physical Society. doi:10.1103/PhysRevA.71.044305.

[44] S.N. Molotkov, On a new attack on quantum key distribution: Joint unambiguous measurements of probe states and the pns attack on information states, JETP Lett. 112 (6) (2020) 383–392, https://doi.org/10.1134/S0021364020180095.

[45] T. Satoh, S. Nagayama, S. Suzuki, T. Matsuo, M. Hajdusek, R.V. Meter, IEEE Trans. Quantum Eng. 2 (2021) 1–17, https://doi.org/10.1109/tqe.2021.3094983.

[46] Y.-Y. Fei, X.-D. Meng, M. Gao, H. Wang, Z. Ma, Quantum man-in-the-middle attack on the calibration process of quantum key distribution, Scientific Rep. 8 (1) (2018) 4283, https://doi.org/10.1038/s41598-018-22700-3.

[47] Y. Zhao, C.-H.F. Fung, B. Qi, C. Chen, H.-K. Lo, Quantum hacking: Experimental demonstration of time-shift attack against practical quantum-key-distribution systems, Phys. Rev. A 78 (4) (2008).

[48] V.M. *, D.R. Hjelme, Faked states attack on quantum cryptosystems, J. Modern Opt. 52(5) (2005) 691–705. doi:10.1080/09500340410001730986.

[49] N. Jain, B. Stiller, I. Khan, D. Elser, C. Marquardt, G. Leuchs, Attacks on practical quantum key distribution systems (and how to prevent them), Contemporary Phys. 57 (3) (2016) 366–387, https://doi.org/10.1080/00107514.2016.1148333.

[50] I. Oshurko, Quantum Mach. Learn. (2020), https://doi.org/10.1142/9781786348210_0010.

[51] M. Schuld, N. Killoran, Quantum machine learning in feature Hilbert spaces, Phys. Rev. Lett. 122(4) (2019) 040504, arXiv: 1803.07128. doi:10.1103/PhysRevLett.122.040504.

[52] Y. Zhang, Q. Ni, Recent advances in quantum machine learning, Quantum Engineering 2 (1) (2020) e34, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/que2.34. doi:10.1002/que2.34. https://onlinelibrary.wiley.com/doi/abs/10.1002/que2.34.

[53] V. Havlicek, A.D. Córcoles, K. Temme, A.W. Harrow, A. Kandala, J.M. Chow, J.M. Gambetta, Supervised learning with quantum enhanced feature spaces, Nature 567 (7747) (2019) 209–212, https://doi.org/10.1038/s41586-019-0980-2, arXiv: 1804.11326.

[54] E.C. Behrman, J.E. Steck, A quantum neural network computes its own relative phase, arXiv:1301.2808 [quant-ph]ArXiv: 1301.2808 (Jan. 2013). http://arxiv.org/abs/1301.2808.

[55] S. Gupta, R.K.P. Zia, Quantum Neural Networks, arXiv:quant-ph/0201144ArXiv: quant-ph/0201144 (Jan. 2002). http://arxiv.org/abs/quant-ph/0201144

[56] N.H. Nguyen, E.C. Behrman, J.E. Steck, Quantum learning with noise and decoherence: a robust quantum neural network, Quantum Mach. Intell. 2 (1) (2020) 1, https://doi.org/10.1007/s42484-020-00013-x.

[57] S. Lu, L.-M. Duan, D.-L. Deng, Quantum adversarial machine learning, Phys. Rev. Res. 2 (3) (2020), https://doi.org/10.1103/physrevresearch.2.033212.

[58] H. Liao, I. Convy, W.J. Huggins, K.B. Whaley, Robust in practice: Adversarial attacks on quantum machine learning, Phys. Rev. A 103 (4) (2021), https://doi.org/10.1103/physreva.103.042427.

[59] A.B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins, R. Chatila, F. Herrera, Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai (2019). doi:10.48550/ARXIV.1910.10045.

[60] J. Preskill, Quantum Computing in the NISQ era and beyond, Quantum 2 (2018) 79, publisher: Verein zur Förderung des Open Access Publizierens in den Quantenwissenschaften. doi:10.22331/q-2018-08-06-79. https://quantum-journal.org/papers/q-2018-08-06-79/.

[61] C.G. Almudever, L. Lao, X. Fu, N. Khammassi, I. Ashraf, D. Iorga, S. Varsamopoulos, C. Eichler, A. Wallraff, L. Geck, A. Kruth, J. Knoch, H. Bluhm, K. Bertels, The engineering challenges in quantum computing, in: Design, Automation Test in Europe Conference Exhibition (DATE), 2017, 2017, pp. 836–845, iSSN: 1558–1101. doi:10.23919/DATE.2017.7927104.

[62] M. Schlosshauer, Quantum decoherence, Phys. Rep. 831 (2019) 1–57, https://doi.org/10.1016/j.physrep.2019.10.001.

[63] P. Krantz, M. Kjaergaard, F. Yan, T.P. Orlando, S. Gustavsson, W.D. Oliver, A quantum engineer's guide to superconducting qubits, Appl. Phys. Rev. 6(2) (2019) 021318, publisher: American Institute of Physics. doi:10.1063/1.5089550.

[64] C.D. Bruzewicz, J. Chiaverini, R. McConnell, J.M. Sage, Trapped-ion quantum computing: Progress and challenges, Appl. Phys. Rev. 6 (2) (2019) 021314, publisher: American Institute of Physics. doi:10.1063/1.5088164.

[65] R. Iten, R. Moyard, T. Metger, D. Sutter, S. Woerner, Exact and practical pattern matching for quantum circuit optimization (2019). doi:10.48550/ARXIV.1909.05270.

[66] S.J. Devitt, W.J. Munro, K. Nemoto, Quantum error correction for beginners, Rep. Prog. Phys. 76(7) (2013) 076001, publisher: IOP Publishing. doi:10.1088/0034-4885/76/7/076001.

[67] J. Preskill, Lecture notes for physics 229: Quantum information and computation, California Institute of Technology 16 (1) (1998) 1–8.

[68] J. Preskill, Reliable quantum computers, Proc. R. Soc. London Ser. A 454 (1969) (1998) 385–410, publisher: Royal Society. doi:10.1098/rspa.1998.0167.

[69] J. Roffe, Quantum error correction: an introductory guide, Contemporary Phys. 60 (3) (2019) 226–245, https://doi.org/10.1080/00107514.2019.1667078, publisher: Taylor & Francis _eprint:.

[70] E.T. Campbell, B.M. Terhal, C. Vuillot, Roads towards fault-tolerant universal quantum computation, Nature 549 (7671) (2017) 172–179, bandiera_abtest: a Cg_type: Nature Research Journals Number: 7671 Primary_atype: Reviews Publisher: Nature Publishing Group Subject_term: Information theory and computation;Quantum information;Qubits Subject_term_id: information-theory-and-computation;quantum-information;qubits. doi:10.1038/nature23460. https://www.nature.com/articles/nature23460.

[71] J. Preskill, Fault-tolerant quantum computation, in: Introduction to quantum computation and information, World Scientific, 1998, pp. 213–269.

[72] A.A. Saki, M. Alam, K. Phalak, A. Suresh, R.O. Topaloglu, S. Ghosh, A Survey and Tutorial on Security and Resilience of Quantum Computing, arXiv:2106.06081 [quant-ph]ArXiv: 2106.06081 (Jun. 2021). http://arxiv.org/abs/2106.06081.

[73] A. García de la Barrera, I. García-Rodríguez de Guzmán, M. Polo, M. Piattini, Quantum software testing: State of the art, Journal of Software: Evolution and Process n/a (n/a) e2419. doi: 10.1002/smr.2419.

[74] Y. Huang, M. Martonosi, QDB: From Quantum Algorithms Towards Correct Quantum Programs (2019) 14 pages doi:10.4230/OASIcs.PLATEAU.2018.4.

[75] Y. Huang, M. Martonosi, Statistical assertions for validating patterns and finding bugs in quantum programs, in: Proceedings of the 46th International Symposium on Computer Architecture, ISCA '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 541–553. doi:10.1145/3307650.3322213.

[76] P. Zhao, J. Zhao, L. Ma, Identifying Bug Patterns in Quantum Programs, arXiv:2103.09069 [quant-ph]ArXiv: 2103.09069 (Mar. 2021). http://arxiv.org/abs/2103.09069.

[77] J. Campos, A. Souto, QBugs: A Collection of Reproducible Bugs in Quantum Algorithms and a Supporting Infrastructure to Enable Controlled Quantum Software Testing and Debugging Experiments (Mar. 2021). doi:10.48550/arXiv.2103.16968. http://arxiv.org/abs/2103.16968.

[78] P. Zhao, J. Zhao, Z. Miao, S. Lan, Bugs4Q: A Benchmark of Real Bugs for Quantum Programs, IEEE Comput. Soc. (2021) 1373–1376, https://doi.org/10.1109/ASE51524.2021.9678908, URL: https://www.computer.org/csdl/proceedings-article//ase/2021/033700b373/1AjThPvMJqM.

[79] J. Wang, M. Gao, Y. Jiang, J. Lou, Y. Gao, D. Zhang, J. Sun, QuanFuzz: Fuzz Testing of Quantum Program (Oct. 2018). doi:10.48550/arXiv.1810.10310.

[80] S. Honarvar, M.R. Mousavi, R. Nagarajan, Property-based Testing of Quantum Programs in Q#, in: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, Association for Computing Machinery, New York, NY, USA, 2020, pp. 430–435. doi: 10.1145/3387940.3391459.

[81] X. Wang, P. Arcaini, T. Yue, S. Ali, Generating Failing Test Suites for Quantum Programs With Search, in: Search-Based Software Engineering: 13th International Symposium, SSBSE 2021, Bari, Italy, October 11–12, 2021, Proceedings, Springer-Verlag, Berlin, Heidelberg, 2021, pp. 9–25. doi:10.1007/978-3-030-88106-1_2.

[82] Y. Li, M. Ying, Debugging Quantum Processes Using Monitoring Measurements, Phys. Rev. A 89 (4) (2014), https://doi.org/10.1103/PhysRevA.89.042338.

[83] J. Liu, G.T. Byrd, 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, ASPLOS '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 1017–1030. doi:10.1145/3373376.3378488.

[84] K. Singhal, Quantum Hoare Type Theory (Master's thesis), University of Chicago, Chicago, IL, Dec. 2020, URL: https://ks.cs.uchicago.edu/publication/qhtt-masters/.

[85] K. Singhal, J. Reppy, Quantum Hoare Type Theory: Extended Abstract, Electronic Proceedings in Theoretical Computer, Science 340 (2021) 291–302, https://doi.org/10.4204/EPTCS.340.15, http://arxiv.org/abs/2109.02198.

[86] A.J. Abhari, S. Patil, D. Kudrow, J. Heckey, A. Lvov, F.T. Chong, M. Martonosi, ScaffCC: a framework for compilation and analysis of quantum computing programs, in: Proceedings of the 11th ACM Conference on Computing Frontiers, CF '14, Association for Computing Machinery, New York, NY, USA, 2014, pp. 1–10. doi:10.1145/2597917.2597939.

[87] A.W. Cross, L.S. Bishop, J.A. Smolin, J.M. Gambetta, Open Quantum Assembly Language, arXiv:1707.03429 [quant-ph]ArXiv: 1707.03429 (Jul. 2017). http://arxiv.org/abs/1707.03429.
[88] S.-H. Hung, K. Hietala, S. Zhu, M. Ying, M. Hicks, X. Wu, Quantitative robustness analysis of quantum programs, Proceedings of the ACM on Programming Languages 3 (POPL) (2019) 31:1–31:29. doi:10.1145/3290344.
[89] M. Piattini, G. Peterssen Nodarse, R. Pérez-Castillo, J.L. Hevia Oliver, M. Serrano, G. Hernández González, I. Guzmán, C. Andrés Paradela, M. Polo, E. Murina, L. Jiménez Navajas, J. Marqueño, R. Gallego, J. Tura, F. Phillipson, J. Murillo, A. Niño, M. Rodríguez, The Talavera Manifesto for Quantum Software Engineering and Programming, 2020.
[90] T. Patel, D. Tiwari, Veritas: Accurately estimating the correct output on noisy intermediate-scale quantum computers, in: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, 2020, pp. 1–16. doi:10.1109/SC41405.2020.00019.