The Journal of Systems & Software 190 (2022) 111326

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

Software engineering for quantum programming: How far are we?[☆]

Manuel De Stefano[a,*], Fabiano Pecorelli[b], Dario Di Nucci[a], Fabio Palomba[a], Andrea De Lucia[a]

[a] SeSa Lab - University of Salerno, Italy
[b] Tampere University, Finland

ARTICLE INFO

Article history:
Received 23 October 2021
Received in revised form 5 April 2022
Accepted 10 April 2022
Available online 20 April 2022

Keywords:
Quantum computing
Software engineering for quantum programming
Empirical software engineering

ABSTRACT

Quantum computing is no longer only a scientific interest but is rapidly becoming an industrially available technology that can potentially overcome the limits of classical computation. Over the last years, all major companies have provided frameworks and programming languages that allow developers to create their quantum applications. This shift has led to the definition of a new discipline called *quantum software engineering*, which is demanded to define novel methods for engineering large-scale quantum applications. While the research community is successfully embracing this call, we notice a lack of systematic investigations into the state of the practice of quantum programming. Understanding the challenges that quantum developers face is vital to precisely define the aims of quantum software engineering. Hence, in this paper, we first mine all the GitHub repositories that make use of the most used quantum programming frameworks currently on the market and then conduct coding analysis sessions to produce a taxonomy of the purposes for which quantum technologies are used. In the second place, we conduct a survey study that involves the contributors of the considered repositories, which aims to elicit the developers' opinions on the current adoption and challenges of quantum programming. On the one hand, the results highlight that the current adoption of quantum programming is still limited. On the other hand, there are many challenges that the software engineering community should carefully consider: these do not strictly pertain to technical concerns but also socio-technical matters.

1. Introduction

The dream has come true (Knight, 2018): several physicians and computer scientists agree that the quantum technology is right around the corner (Knight, 2018; Hoare and Milner, 2005) and that the 21st century will be recalled as the *"quantum era"* (Piattini et al., 2021). Specific mechanic principles such as *superposition*, i.e., quantum objects may assume different states at the same time, and *entanglement*, i.e., quantum objects may be deeply connected without any direct physical interaction, promise to revolutionize program computation compared to classical computers (Mueck, 2017). Quantum computers could eventually lead to resolving NP-complete problems (Aaronson, 2005; Ohya and Volovich, 2008)—often referred to as *quantum supremacy* (Arute et al., 2019), namely the point in time when a programmable quantum device would be able to solve problems that no classical computer can solve in any feasible amount of time.

For this reason, all major software companies, like IBM and Google, are currently investing hundreds of millions of dollars every year to produce novel hardware and software technologies that can support the execution of quantum programs.[1] For instance, IBM Quantum[2] has developed its programming framework, which allows developers to design, implement, and execute quantum applications on cloud-based quantum computers. Companies and researchers have also been developing several quantum programming languages (Ömer, 2003; Q, 2021; Altenkirch and Grattage, 2005) and development toolkits (Aleksandrowicz et al., 2019; Broughton et al., 2020; Steiger et al., 2018) that provide developers with off-the-shelf instruments and APIs to create quantum programs.

While there have been already several promising applications of quantum programming to the resolution of various problems in the fields of machine learning (Biamonte et al., 2017), optimization (Guerreschi and Smelyanskiy, 2017), cryptography (Mailloux et al., 2016), and chemistry (Reiher et al., 2017), the development of large-scale quantum software seems to be still far from being

☆ Editor: Alexander Chatzigeorgiou.
* Corresponding author.
E-mail addresses: madestefano@unisa.it (M. De Stefano), fabiano.pecorelli@tuni.fi (F. Pecorelli), ddinucci@unisa.it (D. Di Nucci), fpalomba@unisa.it (F. Palomba), adelucia@unisa.it (A. De Lucia).

1 Boston Consulting Group report: .
2 IBM Quantum: https://www.ibm.com/quantum-computing/.

https://doi.org/10.1016/j.jss.2022.111326
0164-1212/© 2022 Elsevier Inc. All rights reserved.

a reality. In this respect, researchers such as Piattini et al. (2021, 2020b,a), Moguel et al. (2020), and Zhao (2020) advocated the need for a brand new scientific discipline able to rework and extend the classical software engineering into the quantum domain. This new field that should enable developers to design and develop quantum programs with the same confidence as classical programs is what we call *quantum software engineering* (QSE) (Zhao, 2020).

In response to the quantum software engineering call, our research community has proposed thematic workshops, like Q-Se[3] and Q-SET,[4] as well as devising novel processes (Barbosa, 2020), modeling techniques (Gemeinhardt et al., 2021), and debugging mechanisms (Li et al., 2020). Recognizing the initial effort spent by the research community, we notice a notable lack of empirical investigations to provide a more comprehensive overview of the *current state of the practice* on quantum software engineering. Therefore, there is a need to analyze how quantum programming is currently used and the key software engineering challenges developers face when programming quantum programs. An improved understanding of these critical aspects might shed light on the limitations of the state of the art and drive the next research steps that our community should invest in.

To the best of our knowledge, El aoun et al. (2021) have been the first to work along these lines. They conducted an empirical study on the questions asked by quantum developers on Stack Exchange forums, as well as the issues reported on GitHub. The authors performed qualitative coding analyses and automated topic modeling to uncover the topics in quantum software engineering-related posts and issue reports. According to the reported results, knowledge of quantum theory and quantum program comprehension represent key engineering aspects hindering the developer's capabilities to develop quantum programs.

In this paper, we aim at complementing and extending the work by El aoun et al. (2021). Rather than analyzing the traces left by developers over forums and issues, we approach the understanding of the current state of the practice on quantum programming by proposing a two-step investigation that includes a mining software repository analysis and a survey study. The ultimate goal of our study is to assess how far quantum software engineering is from effectively supporting developers in practice.

In particular, we first mine all the GitHub repositories that employ the three most widely used quantum programming frameworks implementing the quantum logic gate model, i.e., Qiskit (Aleksandrowicz et al., 2019), Cirq (Developers, 2021), and Q♯ (Quantum Development Kit, 2021). Our choice to focus on the emerging technologies enabling the universal quantum gate model is driven by the fact that this can be applied to a broader range of problems if compared to other models, e.g., quantum annealing (Finnila et al., 1994) or quantum adiabatic (Farhi et al., 2000). We conduct content analysis sessions (Lidwell et al., 2010) to elicit a taxonomy of purposes for which quantum programming is used. Afterwards, we run a survey among the contributors of the mined repositories, asking about their opinions and perspectives on the current adoption of quantum programming and the key challenges they face when engaging with the currently available quantum technologies.

On the one hand, our repository analysis highlights that quantum programming is currently mainly used for didactic purposes or for curiosity to experiment with quantum technologies. On the other hand, the survey study identifies five significant challenges related to the comprehension of quantum programs, the hardness of setting up hardware and software infrastructures, the implementation and code quality issues, the difficulty of building a quantum developer's community, and, perhaps more importantly, the lack of realism of the current quantum applications.

Based on these observations, we conclude that the road ahead for quantum software engineering is still long and concerns technical and socio-technical matters.

Structure of the paper. Section 2 overviews the basic concepts behind quantum computing and programming, while Section 3 summarizes the existing literature. Section 4 elaborates on the research questions, the context of the study, and the methodology of our empirical study. In Sections 5 and 6 we analyze the results addressing the two research questions of the study. Section 7 discusses the main findings and the implications that they have for researchers, practitioners, and tool vendors. The threats to the validity of the study are reported and discussed in Section 8. Finally, Section 9 concludes the paper and outlines our future research agenda.

2. Background

In the following, we describe the basics for understanding the universal gate model of quantum computing (Barenco et al., 1995; Feynman, 2017).

2.1. Quantum bits

In classical programming, the base unit of information is the bit, which can assume only binary values. In quantum computing, the base unit of information is the *quantum* bit (i.e., *qubit* or *qbit*). A qubit differs from a classical bit since its state is a linear combination of two bases in the quantum space, represented by a bidimensional vector (Kaye et al., 2007). Thus, the computational basis of the qubit is defined as:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{1}$$

Then, a generic qubit $|q\rangle$ can be represented as:

$$|q\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2}$$

where α and β are complex numbers subject to the *normalization condition*, i.e., $|\alpha|^2 + |\beta|^2 = 1$. This is necessary since $|\alpha|^2$ and $|\beta|^2$ indicates the probability of the qubit to be either in state 0 or 1. The fact that the qubit can be found in one of these states with a certain probability is called *superposition* (Kaye et al., 2007). In other words, a quantum computer consisting of Qubits is in many different states at the same time.

2.2. Quantum gates

In classical logical circuits, the state of a bit can be changed by applying a gate. The same concept applies to quantum circuits. In this case, such gates can be applied on a single (i.e., for the transition of a single quantum state) or multiple qubits (i.e., for the transition of multiple quantum states). The number of inputs and outputs of a gate should be equal to make the operation reversible. A *single qubit gate* is represented by a squared bidimensional matrix. The resulting quantum state is determined by multiplying the quantum state vector with the matrix. An example of a single qubit gate is the *NOT* gate, denoted in the following.

$$NOT = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{3}$$

[3] The Q-Se workshop: https://q-se.github.io/qse2021/.
[4] The Q-SET workshop: https://quset.github.io/qset2021/

Applying a *NOT* gate to a generic qubit $[\alpha, \beta]$ results in a new state vector whose components are inverted, as shown in the following:

$$NOT \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix} \tag{4}$$

Single qubit gates are fundamental to achieve the *superposition* condition described above.

Conditional logic operations are needed to change the state of a qubit given the state of another qubit. They require quantum gates with multiple inputs and outputs, namely *multiple qubits gates*, which are represented as square matrices having a higher dimension. An example is the *controlled not* (*CNOT*):

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{5}$$

This operation takes as input two qubits: the control qubit and the target qubit. When the control qubit has state $|0\rangle$, the target qubit remains unchanged, whereas when the control qubit has state $|1\rangle$, a NOT gate is applied to the target qubit. Please consider that in both cases, the control qubit remains unchanged. By leveraging multiple qubit gates, in particular the CNOT and the H gate, it is possible to achieve the *entanglement* (Kaye et al., 2007) among qubits, which is a condition in which the state of a set of qubits cannot be described anymore by considering the state of a single qubit, but considering the system as a whole. An entangled system is one whose quantum state cannot be factored as a product of the states of its local constituents; in other words, they are not discrete particles but rather an indivisible totality. If two constituents are entangled, one discrete particle cannot be properly defined without addressing the other. The state of a composite system may always be expressed as a sum, or superposition, of products of local constituent states; it is entangled if this sum cannot be represented as a single product term (Kaye et al., 2007). In these cases, changing one qubit will affect all other constituents. This particular situation can be achieved (for instance), if we put a CNOT gate on a two-qubit registry, and the control qubit is in superposition (to be precise the $|+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ state). Entanglement plays a crucial role for quantum computing since it is necessary for a quantum algorithm to offer an exponential speed-up over classical computations (Jozsa and Linden, 2003). Indeed, changing the state of an entangled qubit will immediately change the state of all paired qubits, speeding up the process.

2.3. Quantum measurement

The measurement, i.e., the process to obtain the information of the qubits, is one of the most important operations appliable to qubits. Measurements are irreversible and permanently force qubits to certain states (i.e., 0 or 1). To be more precise, the probability of measuring a state $|\psi\rangle$ in any qubit state $|x\rangle$ is given by the following equation:

$$p(|x\rangle) = |\langle x|\psi\rangle|^2 \tag{6}$$

This action has several implications. Any qubit in the state $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ has $|\alpha|^2$ probability to be found in the state $|0\rangle$, and $|\beta|^2$ probability to be found in the state $|1\rangle$. Measurements alter the magnitudes of α and β. For instance, if the result of the measurement is $|1\rangle$, α is changed to 0, and β is changed to the phase factor $e^{i\phi}$ that is no longer experimentally accessible. If a measurement is performed on an entangled qubit, it may collapse the state of the other entangled qubits.

2.4. Quantum programming

Combining qubits and gates, we obtain the *quantum (logical) circuit*, one of the most commonly used and general-purpose quantum computing models (Kaye et al., 2007). This allows to leverage on *superposition* and *entanglement* among qubits and achieve a *theoretical* advantage over classical computers in performing large-scale parallel computation (Kaye et al., 2007).

To determine the type of transformation the circuit performs, we need to analyze the structure of quantum circuits, the number and the type of gates, and the interconnection scheme. The result of a quantum circuit can be read out through *quantum measurements*.

```
from qiskit import QuantumCircuit
# Create a quantum circuit with two qubits and
    classical bits:
qc = QuantumCircuit(2,2)
# Apply H-gate to the first:
qc.h(0)
# Apply a CNOT:
qc.cx(0,1)
    # measure the qubits
    qc.measure(0,0)
    qc.measure(1,1)
```

Listing 1: Qiskit code for the bell state circuit.

Quantum programming is the process of designing and building executable code that a quantum computer will execute to obtain a particular result (Miszczak, 2012; Ying, 2016). A quantum program contains classical code blocks and quantum components. As a result, a typical quantum program contains two types of instructions (or statements), namely quantum and classical statements. On the one hand, classical instructions work with the state of classical bits and apply conditional expressions. On the other hand, quantum instructions operate on the state of qubits and measure qubit values. These operations, which can be applied to (*single qubit*) and (*multiple qubits*) and can be reversible, are mainly represented as quantum circuits that manipulate qubit register to perform quantum operations. At the end of the operation, classical bit registers are used to store *quantum measurements*.

Quantum programming, at the state of the practice, heavily leverages libraries and APIs (e.g., Qiskit) to define quantum circuits and run them on quantum machines. Listing 1 shows the code of a simple circuit, which puts the 2 qubits in an entangled state called *bell state*, which is a state in which the only two possible measures can be `00` or `11`, with a 50% probability each. As can be seen, to assemble a quantum circuit, the class `QuantumCircuit` is imported from the Qiskit library. Then, the quantum gates are applied on the qubits invoking some API defined on the same `QuantumCircuit` class.

3. Related work

Research in software engineering for quantum programming, or quantum software engineering (QSE), is in its infancy. During the first International Workshop on Quantum Software Engineering, researchers and practitioners proposed a QSE manifesto, known as the "Talavera Manifesto", which defines the set of fundamental principles of this new discipline (Piattini et al., 2020b,a). Some of these principles include (i) agnosticism towards specific quantum technologies, (ii) coexistence of classical and quantum programming, (iii) support for developing and maintaining quantum software. Since then, several studies have been presented

Table 1
Comparison between the state-of-the art work which explicit challenges in quantum software engineering.

Paper title	Methodology	Main results
Toward a Quantum Software Engineering (Piattini et al., 2021)	Analysis of the impact that the rules defined in the Talavera Manifesto (Piattini et al., 2020b) can have in the fields of software engineering.	Proposal of Some *hot topics* to focus on. In particular, some relevant advice is given to practitioners, researchers and universities.
Quantum Software Engineering: Landscapes and Horizons (Zhao, 2020)	Non-Systematic survey of the current literature regarding quantum software engineering	Definition of a set of challenges divided by software engineering areas, i.e., quantum requirements analysis, quantum software design, and quantum software testing.
Understanding Quantum Software Engineering Challenges: An Empirical Study on Stack Exchange Forums and GitHub Issues (El aoun et al., 2021)	Qualitative analysis of QSE-related questions raised by developers on Stack Exchange forums. Automated topic modeling to uncover the QSE-related Stack Exchange posts and GitHub issue reports.	Highlighting some of the more difficult aspects of quantum software engineering that are distinct from traditional software engineering (e.g., explaining quantum code).
Our work	**Combination of mining software repositories and a survey with practitioners.**	**Taxonomies of current usages and challenges, perceived by quantum programmers, of quantum programming.**

discussing challenges and potential direction in QSE research under various perspectives. Details are reported in Table 1.

Piattini et al. (2021) defined a set of topics on which researchers should pay attention. In particular, after digression on the "Golden Eras of Software Engineering", some priority areas for quantum software engineering were explored. To be more precise, they pointed out 4 areas of quantum software engineering which need high attention from developers: software design of quantum hybrid systems, testing techniques for quantum programming, quantum programs quality, and re-engineering and modernization toward classical–quantum information systems. Finally, they provide some useful insights for researchers, practitioners and universities, described in the following. According to them, researchers should consider that many quantum computer scientist do not have a sufficient knowledge of software engineering techniques, so many errors might be done again, and some expensive "rediscoveries" could happen. Moreover, they think that researchers should not wait until quantum programming languages would be "stable" or "refined" in order to propose, adapt or develop software engineering quantum techniques, but develop them in parallel with the evolution of the quantum languages. Finally, they claim that researchers should take advantage from the mistakes of the past and must rely on empirical validation when proposing new software engineering quantum techniques. This work, however, does not rely on a documented and reproducible research methodology to find out the challenges, but leverages on the direct consequences of the statements present in the "Talavera Manifesto".

Zhao (2020) provided a complete overview of quantum software engineering, including all aspects of the quantum software life cycle (requirements analysis, design, implementation, testing) as well as the critical topic of quantum software reuse. The report also explored some of the challenges and opportunities of the field, and recognizes as critical that a comprehensive software engineering discipline emerges for the development of quantum software. Moreover, this work defines the life cycle of quantum software and leverages it as a basis for a comprehensive survey of current research efforts in quantum software engineering. It also overviews quantum software testing and maintenance and discusses some fundamental issues in quantum software engineering. Similarly to Piattini et al.'s work (Piattini et al., 2021), this study does not exploit an empirical study to discover the challenges of the field, but relies on the open questions posed by the surveyed studies.

The first attempt to understand the challenges of quantum programming from a developer perspective was made by El aoun et al. (2021). The paper reports on an empirical investigation conducted analyzing the Stack Exchange forum, where developers ask and answer QSE-related topics, and the GitHub issue reports, where developers raise QSE-related concerns in real-world quantum computing projects. First, the authors reviewed the categories of QSE-related questions raised on Stack Exchange, based on an existing taxonomy of question types on Stack Overflow. Afterward, the subjects of QSE-related Stack Exchange posts and GitHub bug reports were discovered using automated topic modeling. Their main findings showed that the most asked and answered issues in online forums are related to the theory behind quantum programming, the usage of specific data structures and algorithms, the implementation of quantum-related tasks, and the lack of learning resources. This work differs from ours from a methodological perspective: El Aoun et al. derive their set of challenges in QSE from an automated approach (i.e., topic modeling), whilst in our work, the taxonomy of quantum challenges is derived from what developers pointed out as a challenge, having explicitly asked them. Under this perspective, these two approaches can be seen as complementary.

Finally, some other studies in the field of quantum software engineering have tackled a specific challenge, and proposed preliminary solutions. For instance, some have faced the challenge of artifact modeling (Barbosa, 2020; Exman and Shmilovich, 2021; Pérez-Castillo et al., 2021; Gemeinhardt et al., 2021), and others have discussed quality issues, ranging from the definition of specific metrics, to debugging (Zhao et al., 2021; Zhao, 2021; Campos and Souto, 2021).

Our work puts its foundations on the idea that quantum programming poses some challenges that research in quantum software engineering should face. However, differently from previous studies, we conducted a preliminary software repository mining investigation, to better understand the current usage of quantum programming technologies, and then we surveyed quantum developers, asking directly to them what they perceive as a challenge in their field.

4. Research methodology

The *goal* of our study was to investigate the current usage of quantum programming technologies and to explore the challenges that quantum developers face nowadays, with the *purpose* of assessing where software engineering methods and practices can be applied and how. The *perspective* was of researchers, practitioners, and tool vendors: the former are interested in understanding how software engineering could steer the research in the quantum programming field, letting the challenges emerge; practitioners are interested in gathering insights on how to engineer software products that include quantum computing components; tool vendors are instead interested in assessing the current support provided to quantum developers, to understand possible addressable limitations.

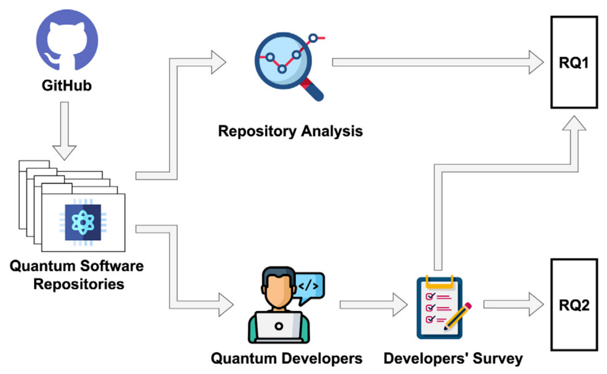

Fig. 1. Graphical summary of the applied methodology. By mining GitHub, we have obtained a set of repositories that we manually classified. Afterward, we extracted the list of public emails of their developers, whom we surveyed. We addressed RQ$_1$ using the classification results, while we addressed RQ$_2$ with the survey responses.

More specifically, our work was structured around two main research questions. We started our investigation by assessing the current usage of quantum programming, namely, the developer's purpose to employ quantum computing frameworks in their products. An improved understanding of these aspects may provide insights into the activities developers typically perform with the support of quantum computing and where software engineering might be more needed. Specifically, we asked:

RQ$_1$. *To what extent and for what purposes are quantum programming frameworks being used?*

Once we had investigated the current adoption of quantum programming in practice, we deepen our analysis to identify the major challenges that quantum developers face when dealing with the frameworks that make quantum computing accessible and usable. Knowing the challenges that developers typically encounter would help tool vendors better assist developers and researchers in devising novel techniques and approaches to deal with those issues. Hence, we formulated the second research question:

RQ$_2$. *What are the main challenges that quantum developers are experiencing when interacting with quantum frameworks?*

We aimed at elaborating on the key challenges and opportunities for software engineering when it turns to quantum programming. In particular, we sought to elicit a comprehensive set of challenges that would make quantum developers' daily programming tasks easier if addressed.

As an ultimate result of our empirical investigation, we provided an improved view of how quantum programming currently is and how it might potentially be, should a software engineering process be successfully adopted. We approached such an objective using multiple research instruments, as depicted in Fig. 1 and detailed in the next sections. In a nutshell, we employed software repository mining and grounded theory to classify the current adoption of quantum computing frameworks; then, we set up a survey study to elicit the software engineering challenges that quantum developers face. By design, our empirical study can be classified as a mixed-method investigation where both quantitative and qualitative methodologies are employed (Johnson and Onwuegbuzie, 2004). In terms of reporting, we followed the recent ACM/SIGSOFT Empirical Research Standards (Ralph et al., 2021).[5]

4.1. Context of the study

As explained in Section 2, universal gate quantum programming is based on APIs provided by third-party libraries. For this reason, the scope of the work is mainly delimited by the quantum technologies considered. To understand the current adoption and challenges of quantum programming, we focused on understanding how third-party libraries are used and their limitations from both a technological and a socio-technical perspective. Hence, the *context* of the study was focused on the three state-of-the-practice universal gate quantum programming technologies, i.e., Qiskit (Aleksandrowicz et al., 2019), Cirq (Developers, 2021), and Q♯ (Quantum Development Kit, 2021). These three frameworks represent the main instruments that quantum developers can currently use. As shown later in the paper, they are indeed used by over 95% of the open-source projects on GitHub that include quantum components.

The frameworks are developed and maintained by three big corporations that are widely involved in quantum computing technologies, namely IBM, Google, and Microsoft. These technologies are widely recognized as more mature and stable than others (What to Look, 2021; Open-Source, 2021), having their own peculiar functionalities, and allowing to write and run quantum programs on both local simulators and real quantum devices provided by their vendors.

4.2. Data collection

In our empirical study, we exploited two main sources of information, i.e., the *quantum software repositories* and the *quantum developers*. In the following, we describe the steps we performed in the data collection phase, whilst the data analysis process is described in the next section.

4.2.1. Quantum software repositories

To understand the extent to which and how quantum frameworks are currently used in practice, we first adopted a software repository mining approach aiming at finding those projects which are hosted on GitHub that use at least one of the considered technologies.

By means of the Python Client Library of the GitHub REST APIs,[6] we searched for code snippets (using the `search_code` function) that indicated the use of the technologies we were interested in. More specifically, for Qiskit and Cirq we looked for patterns like '`from qiskit import`' and '`from cirq import`', respectively. Both frameworks are written in Python and, therefore, developers must necessarily use those patterns to include the libraries in their programs. In other words, the use of these patterns ensured the identification of all the repositories that currently employ Qiskit and Cirq in their code. When it turns to Q♯, we had to use a different strategy. Unlike the others, Q♯ is recognized by GitHub as a programming language. For this reason, we could directly look for the repositories written in such a language: we used the `search_repositories` function, passing Q♯ as language parameter. In our online appendix (De Stefano et al., 2022), we released the source code developed for mining quantum repositories. Overall, this process found a total of 731 unique repositories (442 using Qiskit, 217 using Cirq, 72 using Q♯).

[5] Given our study and currently available standards, we followed the general guidelines when reporting the study design and results.

[6] `PyGitHub`: https://github.com/PyGithub/PyGithub

Table 2
Questions asked in the survey.

Question text	Answer type	Possible answers
Part 1 — — — Background		
What is your current employment status?	Multiple choice	B.Sc. Student; M.Sc. Student; Ph.D. Student; Researcher; Open Source Developer; Industrial Developer; Other
What is your educational background?	Single choice	Computer Science; Chemistry; Physics; Other
What is your age range?	Single choice	18–24; 25–34; 35–44; 45–54; 55+
What is your gender?	Free text	–
Please, indicate your expertise (in years) in Software Development.	Single choice	None; 0–3; 3–5; 5–10; 10+
Please, indicate your expertise (in years) in Industrial Development.	Single choice	None; 0–3; 3–5; 5–10; 10+
Please, indicate your expertise (in years) in Quantum Programming.	Single choice	None; 0–3; 3–5; 5–10; 10+
What is your Country?	Free text	–
Part 2 — — — Current adoption		
Which quantum technology are you most confident with?	Single choice	Qiskit; Cirq; Q♯; Other
Which other quantum technology do you use?	Multiple choice	Qiskit; Cirq; Q♯; Other
In which context are you using quantum computing?	Multiple choice	Academic Study; Hackaton; Industry; OSS; Personal Study; Research; Other
Could you please tell more about the tasks you are performing with quantum computing?	Long free text	–
Part 3 — — — Potential adoption and challenges		
Consider the technology you are most confident with. What were the top 3 challenges that you have faced?	Multiple free text	–
Based on your experience, have you ever solved (or tried to solve) a problem using quantum programming which has no "traditional" solution (or the solution is intractable)?	Single choice	Yes; No
If yes, could you please elaborate on the problem and why you have to use quantum computing?	Long free text	–
Based on your experience, have you ever solved (or tried to solve) a problem that has a "traditional" solution using quantum programming?	Single choice	Yes; No
If yes, could you please elaborate on what it was and explain why you chose to use quantum computing?	Long free text	–

4.3. *Quantum software developers*

In the context of our empirical investigation, we needed to collect opinions from developers of quantum-based applications. This step was harder than expected since quantum programmers are a new category of developers that are not necessarily computer scientists but also physicians, chemists, and others. Therefore, we had to customize the recruitment of the participants, the design of the survey, and its dissemination.

Survey recruitment. We took advantage of the software repositories mined from GitHub to obtain a list of eligible candidates for our survey. In this way, we ensured to involve developers having some real experience with quantum programming. Starting from the initial set of 2399 unique developers, we selected only developers having a public email to be contacted with. The email address collection was done by exploiting the GitHub APIs. By applying this selection criterion, we obtained a set of 984 developers. Of these, 79 had less than ten commits: we excluded these developers as they might have only tangentially contributed to the projects and, as such, they might not have the adequate expertise and/or experience to address our questions (Sugar, 2014). This filter led to a total of 905 contributors which could participate to our survey.

We employed an *opt-in* strategy (Hunt et al., 2013) when involving developers. We sent a first email asking whether they would have liked to participate in our survey study and, only in case of positive feedback, we sent them additional instructions. With this strategy we recruited 56 *volunteers* and mitigated possible legal concerns (ICT, 2016). Nonetheless, it might have led to self-selection or voluntary response bias (Heckman, 1990; Sakshaug et al., 2016). To mitigate this, we introduced a prize of four Amazon gift cards with a total value of $100.

Survey design. The survey was composed of three main sections, as reported in Table 2. The first one aimed at gathering the background of the participants. Besides asking about their current employment and their experience with software development, industrial development, and quantum programming, we asked for information about gender and country (in a free text form, as recommended by recent research (Fink, 2003; Broussard et al., 2018)) and educational background. This latter question was intended to understand more closely the expertise of the current population of quantum programmers; such information might be interesting to reveal cultural barriers to quantum software engineering (Noll et al., 2011).

The second section of the survey aimed at gathering developer's opinions about the current use of quantum technologies. We first asked which of the available frameworks they use and, more in general, the technologies they are typically comfortable with. Afterward, we asked the reason behind the use of quantum programming. This question was essential to understand better why quantum technologies are currently employed, e.g., for an academic interest or for a personal study of how to program using quantum computing. In addition, this question enabled us to complement the grounded-theory exercise conducted to address **RQ**$_1$ (more details are reported in Section 4.4).

The last section targeted the longer-term adoption of quantum technologies. More particularly, it aimed at assessing the main challenges currently faced by developers when adopting the quantum framework they are most comfortable with. This question sought to elicit the most relevant limitations of the existing instruments, potentially highlighting the most pressing issues and challenges that researchers and tool vendors might need to pay attention to. Afterward, we focused on the interplay between quantum and traditional computing. Finally, it aimed at analyzing whether developers addressed or are trying to address problems with and without traditional or tractable solutions through the adoption of quantum technologies. In both cases, we asked to further elaborate on the specific problems treated and the rationale behind the use of quantum programming. Answers to these questions helped us to understand how far quantum technologies are from solving real-case problems and the software engineering methodologies and tools required to support developers.

Survey dissemination. The survey was developed and disseminated to participants using LimeSurvey,[7] an open-source survey editor released under GNU-GPL license. The questionnaire was available from June 1 to June 30, 2021. The survey link was sent to every recruited developer via e-mail. We estimated a completion time of 15 min.

Ethical and privacy considerations. In our country, it is not mandatory yet to seek approval from an Ethical Review Board when releasing surveys and experiments with human subjects. Nonetheless, we took into account several possible ethical and privacy concerns. In the first place, we guaranteed the participants' privacy by not asking their names or email addresses, hence gathering anonymous answers. When recruiting developers, we clearly stated the goal of the survey study, as well as explicitly report that the given answers would have been used in the scope of a research activity that would not have any intention of publishing sensitive data. We also clarified that completed surveys would eventually become public, although guaranteeing their privacy. Finally, opting for an open-source survey tool avoided ethical concerns that might have potentially led participants to feel uncomfortable answering questions using commercial editors (e.g., Google Forms). Indeed, previous work (Buchanan and Hvizdak, 2009) showed that this aspect impacts the potential response rate in survey studies.

4.4. Data analysis

Once we had collected all the required data, we addressed our research questions. To answer the first research question (**RQ**$_1$), we employed the information coming from both the repository mining and the first two parts of the developers' survey. To answer the second research question (**RQ**$_2$) we only relied on the responses provided to the third part of the survey. The investigation into the purposes of the repositories collected from GitHub and the challenges from the developer's survey share the same methodology. In particular, in both cases, we applied a systematic approach that constructs theories applying a methodical gathering and analysis of the data, known as Straussian Grounded Theory (Corbin and Strauss, 1990). This methodology does not assume previous theoretical formulation but rather adopts an approach wherefore a theory is directly and solely generated from data. As for the other data collected from the survey, i.e., the background and current adoption of quantum programming, we mainly relied on statistics. More details are reported in the following.

Application of Straussian Grounded Theory. The first author (i.e., hereafter, *the main inspector*) extracted the excerpts from the textual data we had. In the context of the repository analysis, the excerpts were README files and repository descriptions (translated in English if written in another language) of each analyzed repository. If both README file and the repository description were missing, we labeled the purpose of the repository as "Unknown". As for the survey, the excerpts were the individual answers provided by developers. The main inspector applied open coding (Corbin and Strauss, 1990), which is the process of taking the excerpts and continuously comparing and contrasting them with other excerpts, with the final aim of grouping them, and thus giving them a code. These excerpts were considered based on semantic similarity (Harispe et al., 2015). This process was also carried out individually by the other two inspectors (the second and third author of the paper) to have coding as unbiased as possible. To validate this step, we also computed their agreement in terms of Fleiss' k (Fleiss, 1971).

Once the inspectors assigned the codes to the excerpts, they proceeded with the axial coding step (Corbin and Strauss, 1990), which consists of grouping into categories the codes found in the previous step. Alongside the open coding, it is a cyclical process since new excerpts and examined categories might contradict, support, or expand the existing codes and categories. When additional excepts do not expand upon the found code and categories, theoretical saturation (Corbin and Strauss, 1990; Walker, 2012) is reached, i.e., the point at which codes and categories are stabilized and fixed.

The resulting taxonomy collects all the codes and categories, resulting from the open and axial coding steps, linked together by a common aspect (or core category), found by selective coding (Corbin and Strauss, 1990). This core category represents the foundation on which the answers to our research questions rely.

Analysis of Other Survey Information. We analyzed the first two sections of the survey to understand the background of the developers in our sample and their take on the current adoption of quantum programming. The demographics of the surveyed developers and their current employment status were summarized using statistics. Furthermore, we analyzed the distribution of contributors per repository category and the distribution of repositories per developer. These distributions were compared against the answers obtained from the survey to add qualitative insights to the repository analysis previously conducted.

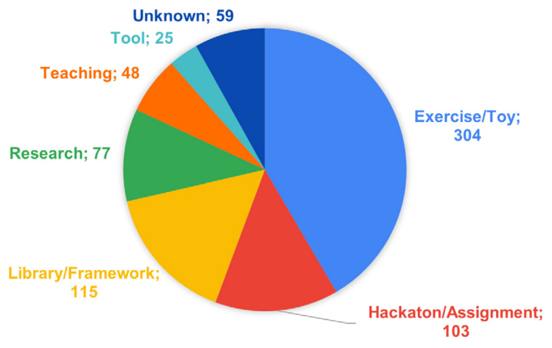

Fig. 2. Partition of the repository per class of usage.

5. RQ$_1$. On the current usage of quantum programming

The analysis of 731 repositories led to a taxonomy of the current usage of quantum programming technologies, reported in Table 3. The taxonomy is composed of six categories, representing the high-level purpose for which the repository was created, e.g., study purposes. Fig. 2 summarizes the repositories partitioned using the taxonomy that we have built, Fig. 3 plots the distribution of developers per kind of repository, whilst Fig. 4 depicts the distribution of developers by type of repository. It must be pointed out that the total number of unique contributors found is 2,399, which is not obtainable by summing up all the values reported in Fig. 3. The reason is that a developer might be contributing to more than a single repository. To this extent, we also analyzed the number of repositories for single developers,

7 LimeSurvey website: https://www.limesurvey.org.

Table 3
Summary of the labels employed in the classification of the mined repositories.

Label name	Purpose	Example
Exercise/Toy	Repository containing toy projects or collection of sample code.	**ryuNagai/QML.** Repository containing experimental code on Quantum Machine Learning.
Hackaton/Assignment	Repository containing code developed for a hackaton or a school assignment.	**oliverfunk/quantum-natural-gradient** Quantum Natural Gradient implemention in Qiskit for the 2019 Qiskit Africa Camp.
Library/Framework	Repository containing code composing a library or a framework.	Davidelanz/quantum-robot. Package for quantum-like perception modeling for robotics.
Research	Repository containing code belonging to a paper or research appendix.	BramDo/custom-cx-gate-on-Casablanca. Code of Qiskit Pulse - Programming Quantum Computers Through the Cloud with Pulses.
Teaching	Repository containing code that complements a lecture or a textbook.	**kongju/QML.** Lecture notes and the code for the course on Quantum Machine Learning offered by University of Toronto on edX.
Tool	Repository containing code for a supporting tool, e.g., a quantum compiler.	**mtreinish/bqskit-qiskit-synthesis-plugin.** This repository contains a PoC unitary synthesis plugin for Qiskit.
Unknown	Repository containing code not classifiable by reading the README or the Description or not having any of them.	**eggerdj/backends.** Repository missing both the README and the Description.

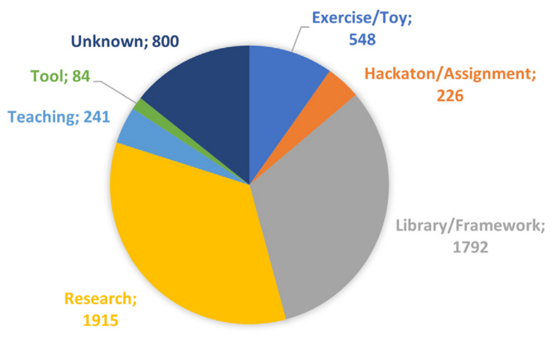

Fig. 3. Number of contributors per type of repository.

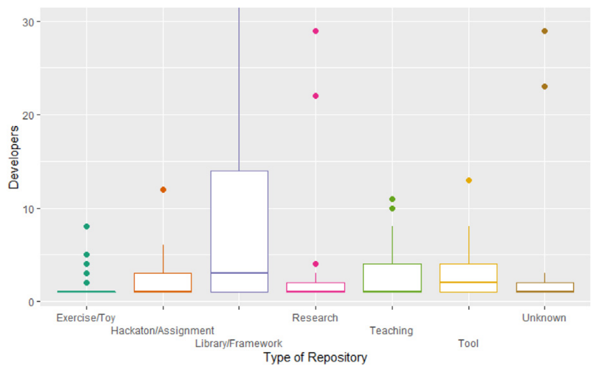

Fig. 4. Distribution of developers by kind of repository. Toy projects have a distribution skewed on the minimum (1), whilst Framework ones have a much wider distribution of contributors.

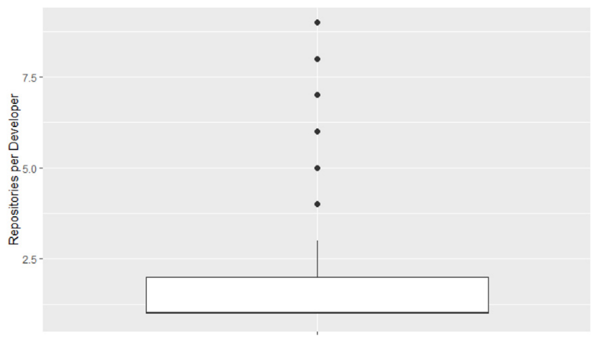

Fig. 5. Distribution of repositories per single developer. The median value insists on 1, indicating that most quantum developers have contributed only to a single repository.

whose distribution is reported in Fig. 5. It is possible to note that, although some developers might have been counted more than once in Fig. 3, the majority we took into consideration has contributed just to a single repository. Thus the number of developers per class of repositories is much more significant.

Since quantum programming is still in its infancy, the main purpose of use is for exercise or personal study. The hosted code aims to explore the features of quantum programming and, in general, is not intended to become a real-world software. This is proved by the fact that 41% of the analyzed repositories belong to this category. However, as shown in Fig. 3 only 548 developers contribute to this kind of repositories, since most of these repositories have only one single contributor, as shown in Fig. 4. Some example of projects developed for such purpose are quantum games, i.e., small and generally text-based games, which leverage the power of *real* randomness given by the qubits to generate random worlds, or scenarios. These are directly correlated with the third-largest slice of repositories, which is composed of repositories developed during hackathons or class assignments.

The other main purpose for which quantum programming is used is to develop quantum libraries or frameworks, which represent the 16% of the total repositories. This outcome is indeed reasonably expected since quantum technologies currently under development are mostly open-source (e.g., Qiskit, Cirq). Moreover, domain- and task-specific libraries are also emerging (e.g., quantum machine learning or chemistry ones). This is the second top class of repositories for number of contributors (1792), and the class of repositories having the most variable distribution of contributors, with a median value of 3.

Online appendices of research papers or related research projects represent 11% of the considered repositories. This result is in line with the fact that quantum programming is still a neat field in the vast plethora of computer science and physics research. Although being the top class for number of developers (1915), the distribution of developers ranges from 1 to 375, with 1 as median value.

The remaining cases of use of quantum programming represent only a small percentage over the total. Evidence has been observed of material used for teaching purposes: book appendices, blog posts, etc. represent 7% of the analyzed repositories. There are also some tools among the repositories that we have inspected (3%), but these, along with teaching materials, were never mentioned by the surveyed developers.

Insights from the developers. Fig. 6 depicts the educational background of the survey participants. Most of the respondents to the survey had a computer science background (55% of the total), while 20% had a physics one. The remaining 25% of the

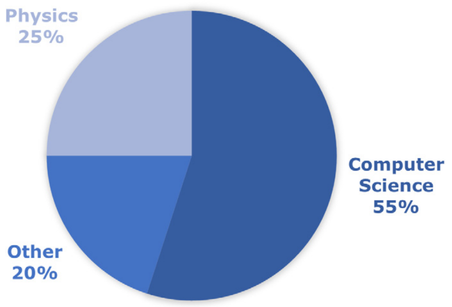

Fig. 6. Educational background of the survey participants.

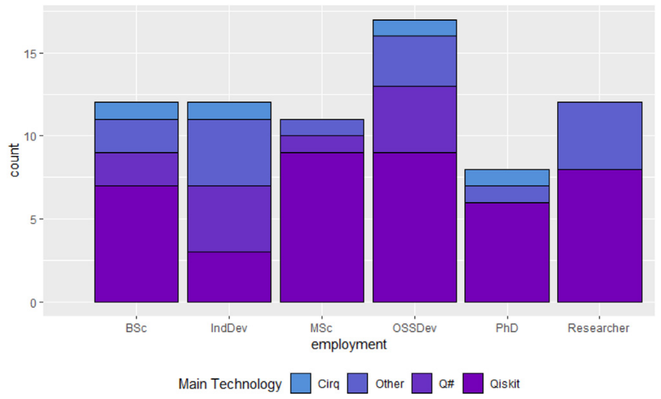

Fig. 7. Employment status of the survey participants, divided by main technology used.

respondents have other backgrounds, e.g., Mathematics, electrical engineering, civil engineering. 71% of the participants were younger than 35 years, with most of them (39%) belonging to the 25–34 age range. Another 21% have an age between 35 and 44 years. Finally, 4% of them belong to the 45–54 age range; the last 4% are older than 54 years.

Fig. 7 shows the employment status of the participants at the time of the survey, divided by used technologies. Most of the participants are Open Source Developers (17), although the majority of them (31) are involved in academia (12 BSc Students, 11 MSc Students, 8 Ph.D. Students). Researchers and Industrial developers include 12 people each. It is also possible to notice that the most used technology is indeed QISKIT in almost all the employments categories, except for industrial developers, who mostly use Q♯ and other technologies. CIRQ is the least used technology by all the categories. It is worth pointing out that researchers only employ QISKIT and Q♯. Most of the respondents (27%) come from the USA, while the other countries with the highest number of respondents are India (12%) and Italy (9%). The other respondents mainly come from European countries (i.e., France, Germany, Greece, Poland, Portugal, Spain, Sweden, Switzerland, United Kingdom), Eastern countries (i.e., China, Japan, Israel, Turkey), and American countries (i.e., Canada, Colombia, Mexico).

The results of the survey show that most of the interviewees (29%) indeed use quantum programming for personal learning. Another large group of users of quantum programming belongs to those who carry out research, who are divided into two categories, namely those who use quantum programming *for* research, and those who actually do research *on* quantum programming. The first group is represented by 14% of the interviewees, while the second by 23%, for a total of 37% developers using quantum programming for research activities. Another 14% of survey respondents use quantum programming to develop technologies for quantum programming itself.

Another interesting result that emerges by analyzing of the answers given to the last questions of the third part of the survey, i.e., whether the respondent has ever applied quantum computing for an intractable classical computation problem (and what was it), and whether the interviewee has ever applied quantum computing to a problem solvable with classical computation (and what was it). The analysis of these answers show that few respondents have tried it, and the problems solved are actually those which are used as an example to show the advantages of quantum computation over the classical one (e.g., applying the Shor algorithm for the factorization). This reinforces the idea that the current use of quantum programming is mainly didactic.

Main findings for RQ1

Quantum programming technologies are used mostly for personal study purposes: 41% of the analyzed repositories were created for this. On the other hand, framework and research repositories have the largest number of contributors, meaning that a major effort is put in these activities. The conclusion is supported by the survey that showcased that most respondents were involved in research or frameworks development activities.

6. RQ$_2$. On the current challenges of quantum programming

Fig. 8 depicts a hierarchical taxonomy of the challenges in quantum programming that we have constructed as outcome of the Straussian Grounded Theory exercise described in 4. Some categories have just one layer, while others have been partitioned. In particular, some categories presented a set of challenges that were much more cohesive than others, and thus they do not need a specialized category to be described. In the following, we describe each category (alongside their sub-categories) of challenges.

☞ **Environment (A)** The problems described in this macro-category are all related to the *quantum* environment, both in terms of hardware and software.

Software Infrastructure (A.1) The challenges in this area range from those relating to the framework being utilized to those relating to the execution environment.

Framework (A.1.1). Challenges regarding the frameworks are mainly related to API design, missing features and standardization. In particular, 15 participants mostly complained about the "*constant changes in API*" of quantum technologies, which often are not correlated with an adequate "*deprecation policy*". Other 2 developers complained of lacking features. For instance, "*some kind of operations are not directly supported by* QISKIT", or "*Are not supported quantum operations among different circuits, which could be interesting to develop operations in different nodes, for Quantum Networks*". A set of 12 participants also pointed out challenges concerning "*gate definitions*" and "*endianess of the qubits*", which differ from framework to framework. i.e., standardization problems. Some technologies implement concepts (e.g., quantum gates) differently from how they are defined in theory and thus hindering the actual development process. "*Conventions and standards haven't fully solidified across the industry yet*". Standardization challenges are revealed also in the form of lack of abstraction. At the time of writing, quantum programs are generally written in

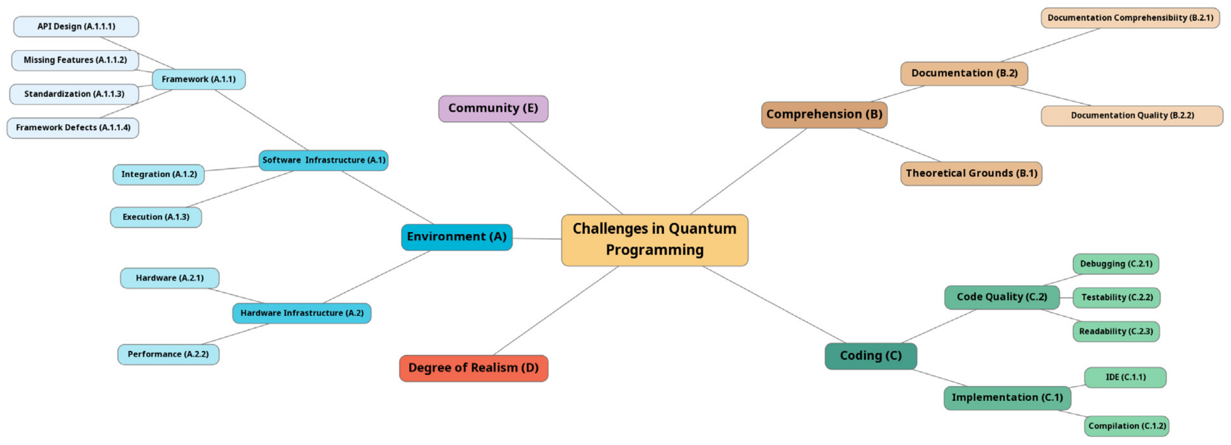

Fig. 8. Graphical representation of the obtained taxonomy of quantum computing challenges.

quantum circuits interacting with classical code. To some extent, a quantum computer is programmed at a very low level of abstraction compared to traditional computers. Having a low level of programming inevitably raises some issues, such as code portability and "interfacing with a wide variety of hardware platforms". In general, as other developers pointed out, quantum code written with a specific technology is meant to be run on the machines that the technology vendor offers, thus leading to *vendor lock-in*. Finally, although some technologies offer ready-to-use implementations, developers must implement quantum algorithms from scratch, relying on what the platform offers. Some developers, finally, (4) complained about defects within the used framework (e.g., "*Incorrect noise modeling of CX gates in parallel*").

Integration (A.1.2) . All challenges related to integrating quantum systems with traditional ones were reported by 6 participants. Developers, for instance, found it challenging to "*integrating a classical algorithm into its quantum analog*" or " *Connecting quantum computers to blockchain networks*".

Execution (A.1.3). The reported challenges falling in this category are related to the execution environment, and were reported by 11 survey participants. They struggle with setting up execution environments, simulators, or classical systems with which quantum programs interact, hindering their ability to execute their programs.

Hardware Infrastructure (A.2) This category groups together challenges that somehow relate to hardware aspects, ranging from the "bare metal" to the performances.

Hardware (A.2.1) Reported hardware challenges fall into this category. Quantum programming requires specialized hardware, which is under constant development. At the time of writing, we have access to quantum computers with a limited small number of qubits. Therefore, large-scale applications cannot be effectively developed. "*No real quantum computer to work on*", "*lack of hardware support*", "*access that is limited to small quantum devices*" are examples of this category of challenges developers face. Finally, quantum computers suffer from noise because of the relative *immaturity* of quantum technologies and some physical limitations (e.g., "IBM's hardware has an error rate that is too high for many purposes", "*qubits propagate error to others while entangling*").

Performance (A.2.2). Quantum hardware is limited, so vendors offer emulators to run quantum programs on classical computers. This leads to another set of challenges related to performance. Emulators are resource greedy, and limit the actual execution of quantum programs ("*Performance of emulators - even with few qubits*", "QISKIT *objects use memory inefficiently, which made them too slow and memory hungry.*"). Similarly, running programs on real quantum devices "*takes time to complete execution*", since vendors queue jobs to guarantee free access to everyone. Finally, a lack of optimization given by the vendors causes troubles to developers, who complain "*Poor optimization of operations*" and compile-time issues ("QISKIT *does not have support in its pipeline for optimizing synthesis for more than 2 qubits*").

☞ **Comprehension (B)** This macro-category refers to all the challenges that developers face in comprehending quantum programs, in all facets. Thus, this category ranges from challenges which are inherent to the documentations of the frameworks, to challenges that are inherent to (lack of) theoretical grounds.

Theoretical Grounds (B.1) This category includes all those challenges that are related to learning quantum technologies. A set of 20 respondents reported that many concepts are needed to get acquainted with quantum programming, particularly linear algebra. Moreover, it was pointed out that programming a quantum computer is completely different, and many found it challenging to theoretically understand new concepts related to quantum circuits and quantum gates. For instance, as one developer reported, it is fairly challenging "*designing a circuit in terms of the basis gates of the QPU to reduce the computational cost to perform a desired task*".

Documentation (B.2) The category represents challenges related to documentation that developers face when approaching quantum technologies. Most of the reported challenges complain about poor consistency in the documentation, and lack of proper getting started material.

Documentation Comprehensibility (B.2.1). Challenges falling into this category include inconsistent tutorials that make difficult to completely understand the technology, and, as a consequence, hinder the actual learning process of the quantum technology. This was pointed out by 3 participants

Documentation Quality (B.2.1). Among all the participants, 16 of them pointed out problems related to the quality of the provided documentation. "*Documentation out-of-date or not comprehensive enough*", "*Outdated and erroneous documentation*", or "*Missing Documentation*" are some examples of the reported complaints.

☞ **Coding (C)** In this category fall all the reported challenges related to coding quantum programs.

Implementation (C.1) Challenges in this category relate to the implementation activities.

IDE (C.1.1). Although not necessary, it is known that a good IDE makes the difference when programming: quantum programming makes no exception. However, some developers (3) have pointed out some IDE-related challenges that hinder, rather than helping, their development tasks. For instance, many found it difficult to work with the Q♯ environment, particularly with the IDE. Others pointed out that working with Qiskit IDEs and Language extensions resulted in an unstable experience on particular hardware architectures. Finally, when it comes to plotting the defined circuit, the IDE does not allow a clear visualization, making coding even harder.

Compilation (C.1.2). Quantum circuits are defined based on quantum technologies that must be compiled to be executed on real quantum machines. The current models for quantum computing require quantum algorithms to be specified as quantum circuits on ideal hardware, ignoring hardware-specific details. Although such modeling should make programs more portable and allow developers not to concentrate on hardware-specific issues (Understanding, 2021), such programs need to be translated into code that quantum computers can execute, i.e., need to be compiled (Understanding, 2021). However, as 4 participants said, "*adapting ideal quantum circuits to available device architectures*" is a tough challenge for developers, particularly, "*hard to write compiler passes*" since "*non standard benchmarking on compilation depths*" are available.

Code Quality (C.2) The challenges that fall into this category are related to *traditional* code quality problems, e.g., testability, debugging, and readability.

Debugging (C.2.1) Some participants (11) have pointed out that they often found it challenging to understand the error messages given by the execution of quantum programs or even the code itself. Debugging these errors is even more complicated, if we consider the uniqueness of this new programming paradigm.

Testability (C.2.2) Understanding if a program performs as intended is a fundamental concept in programming. However, it is not so simple for quantum programming: 4 respondents have pointed out that "*being able to check that the circuit does what you want it to do*" is particularly challenging. In particular, they pointed out that a major problem "*the little understanding of what should the result look like resulting in a lack of common sense regarding the correctness of the result*".

Readability (C.2.3) Another quality-related set of challenges is readability, as pointed out by 1 respondent. Since the quantum code consists of defining a register of qubits and applying gates on them, one of the challenges pointed out by developers is "*creating a readable code*".

☞ **Degree of Realism (D)** This category of challenges involves the applicability of quantum programming to solve real-world problems. In theory, quantum computers should allow developers to solve problems that classical computers cannot solve. Nevertheless, in practice, there are still hardware and performance limitations, which were pointed out by 10 respondents. "*Finding interesting use cases*", "*Using quantum computers on real products*", or "*Formulate a problem*" are only a few examples of the kind of challenges reported by developers. Many people also find it challenging to design quantum programs able to solve real problems or design their quantum algorithms since "*There aren't yet many problems that quantum can solve that traditional tech can't*".

☞ **Community (E)** Challenges in this category concern the lack of a community to interact with. Indeed, many developers wish to have other people they can compare their work with and find support: 12 respondents reported that they suffer from "*lack of professional connections; lack of peer guidance*". These kind of challenges also concern the difficulty of performing code reviews because of missing peers. In particular, developers pointed out that "*Code reviews are slow*", and the issue affects the entire review process. Furthermore, the additional effort necessary to comprehend the programs does not promote code reviews for developers who want to "*learn to review the source code to understand how to use some features*".

Main findings for RQ2

Our results identify challenges related to the comprehension of quantum programs, the hardness of setting up hardware and software infrastructures, the implementation and code quality issues, the difficulty of building a quantum developer's community, and the lack of realism of the current quantum applications.

7. Discussion and implications

Our study's key findings clearly show that quantum programming is still an emerging subject, even though the software engineering approaches available to developers are still limited, and, perhaps more crucially, there are few real-world applications for quantum technology. Both the research questions of the study converge toward a clear implication for the research community:

Take Away Message. There is the need for a joint research effort toward the definition of follow-up empirical studies aiming at delving into the quantum programmers' needs, alongside to the development of tools that these programmers can use to comprehend, set up the required infrastructure, implement, and verify quantum programs. Our study defined a set of research directions to pursue, represented by the challenges defined in our taxonomy, along with concrete and specific problems that quantum programmers currently experience and that the research community is called to address.

While the key take away message is similar to what has been somehow reported in early research on the matter (Piattini et al., 2021, 2020b; Zhao, 2020; El aoun et al., 2021), we aimed at elaborating more on this point.

By nature, survey studies like the one we conducted in this paper might not necessarily provide insights at a granularity level that would allow a proper understanding and analysis of the developer's opinions (Coughlan et al., 2009). For instance, some of the open answers might not be clear enough or might even be contrasting. For this reason, we decided to complement the insights from the survey with additional, finer-grained observations into the potential adoption of quantum technologies and the role of software engineering for quantum programming. To this aim, we created several discussion groups on developer's forums. Being these discussions defined *after* the analysis of the survey responses, our goal was to clarify or explore more closely some of the aspects mentioned in the survey rather than creating generic discussions about quantum software engineering.

We opted for Reddit,[8] a popular social news aggregation, web content rating, and discussion website that developers often

[8] Reddit website: https://www.reddit.com.

use to discuss open issues and challenges on various computer science-related subjects (Medvedev et al., 2017). In particular, REDDIT allows the creation of the so-called *sub-reddits*, i.e., discussion channels dedicated to specific matters. In our case, there exist three sub-reddits dedicated to quantum computing and programming, e.g., '`r/QuantumComputing`', '`r/Qiskit`', and '`r/cirq`'.[9] Hence, also in this case we could target a population of developers that are currently working on quantum programming.

As already mentioned, the goal of such an additional analysis was to let developers discuss specific aspects that were found to be particularly interesting, contrasting, or unclear from the survey responses. These pieces of information were extracted from the first author of the paper after the survey analysis. For each of the identified discussion points, we created a post where we presented ourselves, the goals of the study, and the topic we were interested in. We asked developers to comment and provide their take on it. As an example, let consider the case of theoretical ground—this is one of the key challenges discovered and detailed in Section 6. We created the following post (the introductory part is omitted for the sake of brevity):

Discussion - Example. It seems that this documentation, although simplifying most of the concepts related to quantum mechanics, is still really hard to understand, and most of the parts assume a deep knowledge of linear algebra. Is there a way for a beginner who has mainly a computer science background to start programming with quantum technologies?

As shown, the post aimed to engage developers in a discussion on how to use the documentation of quantum frameworks to start programming with quantum technologies. Similar posts were created for the other discussion points: the complete list of posts is available in our online appendix (De Stefano et al., 2022). Overall, we obtained around 30 answers.

Among these discussions, we found of particular interest the ones about the real applicability of quantum programming and the difficulty for a practitioner with only a computer science background to cope with these technologies, along with other questions regarding code quality issues. An interesting point of discussion was concerned the lack of abstraction which affect the quantum technologies taken into consideration. We found out that the idea of programming is completely different, which *"is more like designing circuits rather than programming the high level stuff. Much like probably 60 to 70 years ago when the concept of computers wasn't well defined and those which existed back then were specific purpose computers and making them would require designing circuits (similar to what we are doing in Quantum programming)"*. This comment suggests that the software engineering researchers might take advantage of the methodologies defined in the past, which resulted in the software and abstraction layers that we currently have, to apply them to quantum computers.

As for the background challenge, we found something interesting. While it is generally recognized that to be able to program a quantum computer a knowledge of quantum physics and advanced linear algebra is required, the discussions we had let emerge it as a *misconception* or *false myth* of quantum programming. This was made clear by one of the developers who reported that: *"Much like how a deep understanding of the band structure of the Silicon in a CPU is not required to do computer science, the end goal of quantum programming is to be able to program on a quantum computer without needing knowledge of what is physically happening to the qubits during the computation"*. Of course, being familiar with some notion of math can help in having a better understanding of the underlying technology, but is not mandatory to be a quantum physics expert—indeed, many books have been published which propose an "hands on" approach, such as the one written by Johnston et al. (2019).

Speaking about *misconceptions*, what emerges from the discussions about the use of quantum programming in real world applications is that many developers struggle to find an actual application of these algorithms. These discussions lead to the idea that quantum computer might sometime replace digital computers, and developers are adapting their programming abilities to achieve that. However, what we know now is that a complete replacement of digital computer is not a matter of near future, given several technical limitations. One of the developers that discussed with us about this topic (using Grover's Algorithm as an example), said that *"real world use cases of Grover's algorithm would definitely take at least thousands of fault tolerant qubits, which at the moment would require millions of noisy qubis"*, which the current state of the art technology cannot give. What we are really able to do with quantum computer is solving optimization problems exploiting variational quantum algorithms on near terms devices (i.e., the quantum computers available so far) (Wecker et al., 2015; Moll et al., 2018; Cerezo et al., 2021). In this scenario, a quantum computer should be seen as an external computational unit (Quantum Computational Unit, QPU), and programmed with that idea in mind, just as Graphical Processing Units (GPUs) are programmed (Johnston et al., 2019). Thus, we can claim that all the challenges which we have shown should be solved by software engineering research with this idea in mind, trying to apply the acquired knowledge with GPU programming and all its issues and challenges to this new emerging field. Remembering that as Booch remarks (Booch, 2018): "No matter the medium or the technology or the domain, the fundamentals of sound software engineering will always apply: craft sound abstractions". Without this in mind, *misconceptions* are behind the corner, making us far from software engineering for quantum programming. The message for practitioners aiming at joining the QSE challenge derives from this: quantum programming should not be considered the panacea of all evils; its current applications are still minimal, especially at the industry level. Practitioners should focus on leveraging the benefits of quantum programming in particular situations. In doing so, they could develop new software engineering methods and tools to boost the research in the field.

8. Threats to validity

A number of threats might have possibly influenced the results of our study. In the following, we report and discuss how we mitigated them.

8.1. Threats to construct validity

Threats to construct validity concern the relationship between hypotheses and observations. When studying the current adoption of quantum programming (**RQ**$_1$), we mined repositories using search strategies aiming at identifying all the projects using the most widely used quantum frameworks available to date. These strategies pertain to source code patterns that developers must necessarily use to include the frameworks in their code or, in the case of Q♯, through the features provided by GITHUB. In any case, our mining strategies represent the only available, hence ensuring the maximum coverage possible. In this respect, we can claim that the analysis done is extensive and not threatened by false positive or false negative projects.

9 At the submission date, there was no sub-reddits specific for Q#.

As for **RQ**$_2$, whenever needed we defined free text answers to let practitioners express their opinions freely, without any restriction. As for the participants involved, we invited developers who contributed to the GitHub projects relying on quantum frameworks. Based on the considerations done for the projects selection, we can ensure that the developers involved in the survey are the ones that are actually working on open-source quantum projects. In addition, we excluded the developers who contributed to those projects with less than ten commits. This was done to make sure to rely on the opinions of developers who had done significant contributions in terms of quantum programming. Nonetheless, replications of the survey study would be beneficial to discover additional points of view and perspectives on the state of quantum programming.

8.2. Threats to conclusion validity

A threat of this kind which might affect the validity of our study is related with the subjectivity of the constructed taxonomies, either the one regarding the current adoption of quantum technologies or the one related to the quantum programming challenges, which might result in a biased classification. In both cases we iteratively built them by splitting and aggregating categories, following a rigorous schemed procedure. Moreover, different authors have taken part of the taxonomy building phase, which provides more confidence of the result achieved. Anyway, also in this case replications would be desirable. To ease the work of other researchers, we released all the material produced in the context of this study in our online appendix (De Stefano et al., 2022).

8.3. Threats to external validity

These threats concern with the generalization of our results. First, the scope of this study is limited to the quantum logic gate model of quantum computing. On the one hand, we did focus on a single model given the exploratory nature of the study. On the other hand, follow-up studies on other quantum models, such as the very promising quantum annealing one, are already part of our research agenda. The study considers the three state-of-the-practice and most mature quantum technologies available so far (What to Look, 2021; Open-Source, 2021), namely Qiskit (Aleksandrowicz et al., 2019), Cirq (Developers, 2021), and Q♯ (Quantum Development Kit, 2021). While other quantum frameworks are emerging, we leave to further research the willingness to compare the results achieved in our empirical study. It must also be considered that our study focuses solely on open-source quantum projects, which can be freely mined and analyzed. Thus the selected repositories only come from GitHub which offers a plethora of mining tools. We are aware that other hosting platforms are available (e.g., Quantum Programming Studio), although not automatically minable; therefore, we acknowledge a threat to validity to the generalizability of our results. It is worth mentioning that the latter platform is technology-agnostic: its inclusion in our work would have been out of scope since the circuits were not designed for the languages we took into consideration but rather translated. They can only host quantum circuits (i.e., algorithms) and not entire quantum applications (which require a classical part). Finally, our results observations might not be valid in industrial contexts. We acknowledge a limitation of our study, limited by the proprietary and closed-source nature of industrial code. The majority of the mined repositories are developed for didactic purposes, which might represent a potential bias to the final results. This characteristic might have biased the results, especially in the case of the faced challenges, which might pertain mainly to developers who are learning the quantum technologies under consideration. Therefore, we plan to replicate this study once quantum programming is more mature and pervasive to improve the generalizability of the already achieved results, targeting a different population of developers. Another potential threat might be represented by the fact that we selected only developers coming from the contributors of the mined GitHub repositories. On the one hand, this might introduce a bias in the generalizability of the study. On the other hand, we had to select a very niche set of developers, so we chose stricter selection criteria. Another potential threat might be represented by the response rate achieved when involving developers in the survey study. In particular, we got an answer from 56 out of 905 contacted developers, which corresponds to a response rate of 6,1%. Such a response rate is in line with respect to other papers that conducted survey studies in software engineering (Blackburn et al., 1996): as such, we deemed the response rate to be good enough. Of course, further experimentation might reveal additional insights that were not discussed by the participants of our study. Finally, the selection criteria we adopted (as recommended by Sugar (2014)) might have excluded a more diverse and potentially more experienced population of developers. We are aware of this risk, and we will consider different selection criteria in the follow-up studies.

9. Conclusions

We have conducted an exploratory study into the current state of quantum programming. First, we applied a manual coding exercise to understand for which purposes the currently available software repositories that use quantum technologies are created. Secondly, we surveyed 56 quantum programmers, inquiring them on the current usage and challenges they face when interacting with the state of the practice quantum frameworks, i.e., Qiskit (Aleksandrowicz et al., 2019), Cirq (Developers, 2021), and Q♯ (Quantum Development Kit, 2021). The results of our empirical study revealed that quantum programming is mainly used for didactic purposes or for curiosity to experiment with quantum technologies. In addition, the survey study shed light on several challenges which are not only related to technical aspects of quantum development but also socio-technical matters, such as the comprehension challenges, the degree of realism, and the community ones.

The output of the study represents the input of our future research agenda. We indeed aim at addressing the challenges identified, by providing (semi-)automated support for developers in terms of quantum program comprehension, analysis, manipulation, and testing.

CRediT authorship contribution statement

Manuel De Stefano: Conceptualization, Formal analysis, Methodology, Writing, Data curation, Writing – original draft. **Fabiano Pecorelli:** Validation, Formal analysis, Writing – review & editing. **Dario Di Nucci:** Validation, Writing – review & editing, Data curation. **Fabio Palomba:** Supervision, Validation, Methodology, Writing – review & editing. **Andrea De Lucia:** Supervision, Validation, Methodology, Writing – review & editing.

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.

Acknowledgment

Fabio gratefully acknowledges the support of the Swiss National Science Foundation through the SNF Projects No. PZ00P2_186090.

References

Aaronson, S., 2005. Guest column: NP-complete problems and physical reality. ACM Sigact News 36 (1), 30–52.
Aleksandrowicz, G., Alexander, T., Barkoutsos, P., Bello, L., Ben-Haim, Y., Bucher, D., Cabrera-Hernández, F.J., Carballo-Franquis, J., Chen, A., Chen, C.-F., et al., 2019. Qiskit: An open-source framework for quantum computing. Accessed on: Mar 16 (2019).
Altenkirch, T., Grattage, J., 2005. Qml: quantum data and control.
Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J.C., Barends, R., Biswas, R., Boixo, S., Brandao, F.G., Buell, D.A., et al., 2019. Quantum supremacy using a programmable superconducting processor. Nature 574 (7779), 505–510.
Barbosa, L.S., 2020. Software engineering for'quantum advantage'. In: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops. pp. 427–429.
Barenco, A., Bennett, C.H., Cleve, R., DiVincenzo, D.P., Margolus, N., Shor, P., Sleator, T., Smolin, J.A., Weinfurter, H., 1995. Elementary gates for quantum computation. Phys. Rev. A 52 (5), 3457.
Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., Lloyd, S., 2017. Quantum machine learning. Nature 549 (7671), 195–202.
Blackburn, J.D., Scudder, G.D., Van Wassenhove, L.N., 1996. Improving speed and productivity of software development: a global survey of software developers. IEEE Trans. Softw. Eng. 22 (12), 875–885.
Booch, G., 2018. The history of software engineering. IEEE Softw. 35 (5), 108–114.
Broughton, M., Verdon, G., McCourt, T., Martinez, A.J., Yoo, J.H., Isakov, S.V., Massey, P., Niu, M.Y., Halavati, R., Peters, E., et al., 2020. Tensorflow quantum: A software framework for quantum machine learning. arXiv preprint arXiv:2003.02989.
Broussard, K.A., Warner, R.H., Pope, A.R., 2018. Too many boxes, or not enough? Preferences for how we ask about gender in cisgender, LGB, and gender-diverse samples. Sex Roles 78 (9), 606–624.
Buchanan, E.A., Hvizdak, E.E., 2009. Online survey tools: Ethical and methodological concerns of human research ethics committees. J. Empir. Res. Hum. Res. Ethics 4 (2), 37–48.
Campos, J., Souto, A., 2021. Qbugs: A collection of reproducible bugs in quantum algorithms and a supporting infrastructure to enable controlled quantum software testing and debugging experiments. arXiv preprint arXiv:2103.16968.
Cerezo, M., Arrasmith, A., Babbush, R., Benjamin, S.C., Endo, S., Fujii, K., McClean, J.R., Mitarai, K., Yuan, X., Cincio, L., et al., 2021. Variational quantum algorithms. Nat. Rev. Phys. 1–20.
Corbin, J.M., Strauss, A., 1990. Grounded theory research: Procedures, canons, and evaluative criteria. Qual. Sociol. 13 (1), 3–21.
Coughlan, M., Cronin, P., Ryan, F., 2009. Survey research: Process and limitations. Int. J. Ther. Rehabil. 16 (1), 9–15.
De Stefano, M., Pecorelli, F., Di Nucci, D., Palomba, F., De Lucia, A., 2022. Software Engineering for Quantum Programming: How Far Are We? — Online Appendix. shorturl.at/sBOP0.
Developers, C., 2021. Cirq. See full list of authors on Github: https://github.com/quantumlib/Cirq/graphs/contributors.
El aoun, M.R., Li, H., Khomh, F., Openja, M., 2021. Understanding quantum software engineering challenges: An empirical study on stack exchange forums and GitHub issues. In: 37th International Conference on Software Maintenance and Evolution (ICSME).
Exman, I., Shmilovich, A.T., 2021. Quantum software models: The density matrix for classical and quantum software systems design. arXiv preprint arXiv:2103.13755.
Farhi, E., Goldstone, J., Gutmann, S., Sipser, M., 2000. Quantum computation by adiabatic evolution. arXiv preprint Quant-Ph/0001106.
Feynman, R.P., 2017. Quantum mechanical computers. Between Quantum Cosm. 523–548.
Fink, A., 2003. The Survey Handbook. sage.
Finnila, A.B., Gomez, M., Sebenik, C., Stenson, C., Doll, J.D., 1994. Quantum annealing: A new method for minimizing multidimensional functions. Chem. Phys. Lett. 219 (5–6), 343–348.
Fleiss, J.L., 1971. Measuring nominal scale agreement among many raters.. Psychol. Bull. 76 (5), 378.
Gemeinhardt, F., Garmendia, A., Wimmer, M., 2021. Towards model-driven quantum software engineering. In: Second International Workshop on Quantum Software Engineering (Q-SE 2021) Co-Located with ICSE 2021.
Guerreschi, G.G., Smelyanskiy, M., 2017. Practical optimization for hybrid quantum-classical algorithms. arXiv preprint arXiv:1701.01450.
Harispe, S., Ranwez, S., Janaqi, S., Montmain, J., 2015. Semantic similarity from natural language and ontology analysis. Synth. Lect. Hum. Lang. Technol. 8 (1), 1–254.
Heckman, J.J., 1990. Selection bias and self-selection. In: Econometrics. Springer, pp. 201–224.
Hoare, T., Milner, R., 2005. Grand challenges for computing research. Comput. J. 48 (1), 49–52.
Hunt, K.J., Shlomo, N., Addington-Hall, J., 2013. Participant recruitment in sensitive surveys: a comparative trial of 'opt in'versus 'opt out'approaches. BMC Med. Res. Methodol. 13 (1), 1–8.
ICT, L., 2016. Is it legal for ghtorrent to aggregate github user data?. https://www.ictrecht.nl/en/blog/is-it-legal-for-ghtorrent-to-aggregate-github-user-data.
Johnson, R.B., Onwuegbuzie, A.J., 2004. Mixed methods research: A research paradigm whose time has come. Educ. Res. 33 (7), 14–26.
Johnston, E.R., Harrigan, N., Gimeno-Segovia, M., 2019. Programming Quantum Computers: Essential Algorithms and Code Samples. O'Reilly Media.
Jozsa, R., Linden, N., 2003. On the role of entanglement in quantum-computational speed-up. Proc. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci. 459 (2036), 2011–2032.
Kaye, P., Laflamme, R., Mosca, M., et al., 2007. An Introduction to Quantum Computing. Oxford University Press on Demand.
Knight, W., 2018. Serious quantum computers are finally here. What are we going to do with them. MIT Technol. Rev. Retrieved on October 30, 2018.
Li, G., Zhou, L., Yu, N., Ding, Y., Ying, M., Xie, Y., 2020. Projection-based runtime assertions for testing and debugging quantum programs. Proc. ACM Program. Lang. 4 (OOPSLA), 1–29.
Lidwell, W., Holden, K., Butler, J., 2010. Universal Principles of Design, Revised and Updated: 125 Ways to Enhance Usability, Influence Perception, Increase Appeal, Make Better Design Decisions, and Teach Through Design. Rockport Pub.
Mailloux, L.O., Lewis II, C.D., Riggs, C., Grimaila, M.R., 2016. Post-quantum cryptography: what advancements in quantum computing mean for it professionals. IT Prof. 18 (5), 42–47.
Medvedev, A.N., Lambiotte, R., Delvenne, J.-C., 2017. The anatomy of reddit: An overview of academic research. In: Dynamics on and of Complex Networks. Springer, pp. 183–204.
Miszczak, J.A., 2012. High-level structures for quantum computing. Synth. Lect. Quantum Comput. 4 (1), 1–129.
Moguel, E., Berrocal, J., García-Alonso, J., Murillo, J.M., 2020. A roadmap for quantum software engineering: Applying the lessons learned from the classics. In: Q-SET@ QCE. pp. 5–13.
Moll, N., Barkoutsos, P., Bishop, L.S., Chow, J.M., Cross, A., Egger, D.J., Filipp, S., Fuhrer, A., Gambetta, J.M., Ganzhorn, M., et al., 2018. Quantum optimization using variational algorithms on near-term quantum devices. Quantum Sci. Technol. 3 (3), 030503.
Mueck, L., 2017. Quantum software. Nature 549 (7671), 171.
Noll, J., Beecham, S., Richardson, I., 2011. Global software development and collaboration: barriers and solutions. ACM Inroads 1 (3), 66–78.
Ohya, M., Volovich, I.V., 2008. New quantum algorithm for studying NP-complete problems. In: Selected Papers of M Ohya. World Scientific, pp. 83–90.
Ömer, B., 2003. QCL-A programming language for quantum computers. Software Available on-line at http://tph.tuwien.ac.at/~{}oemer/qcl.html.
2021. Open-source quantum software projects. https://quantumcomputingreport.com/tools/ (Accessed: 2021-06-05).
Pérez-Castillo, R., Jiménez-Navajas, L., Piattini, M., 2021. Modelling quantum circuits with UML. arXiv preprint arXiv:2103.16169.
Piattini, M., Peterssen, G., Pérez-Castillo, R., 2020a. Quantum computing: A new software engineering golden age. ACM SIGSOFT Softw. Eng. Notes 45 (3), 12–14.
Piattini, M., Peterssen, G., Pérez-Castillo, R., Hevia, J.L., Serrano, M.A., Hernández, G., de Guzmán, I.G.R., Paradela, C.A., Polo, M., Murina, E., et al., 2020b. The talavera manifesto for quantum software engineering and programming.. In: QANSWER. pp. 1–5.
Piattini, M., Serrano, M., Perez-Castillo, R., Petersen, G., Hevia, J.L., 2021. Toward a quantum software engineering. IT Prof. 23 (1), 62–66.
2021. Q#: A quantum programming language. https://qsharp.community (Accessed: 2021-09-21).
2021. Quantum development kit. https://azure.microsoft.com/it-it/resources/development-kit/quantum-computing/ (Accessed: 2021-06-05).
Ralph, P., bin Ali, N., Baltes, S., Bianculli, D., Diaz, J., Dittrich, Y., Ernst, N., Felderer, M., Feldt, R., Filieri, A., de França, B.B.N., Furia, C.A., Gay, G., Gold, N., Graziotin, D., He, P., Hoda, R., Juristo, N., Kitchenham, B., Lenarduzzi, V., Martínez, J., Melegati, J., Mendez, D., Menzies, T., Molleri, J., Pfahl, D., Robbes, R., Russo, D., Saarimäki, N., Sarro, F., Taibi, D., Siegmund, J., Spinellis, D., Staron, M., Stol, K., Storey, M.-A., Taibi, D., Tamburri, D., Torchiano, M., Treude, C., Turhan, B., Wang, X., Vegas, S., 2021. Empirical standards for software engineering research. arXiv:2010.03525.
Reiher, M., Wiebe, N., Svore, K.M., Wecker, D., Troyer, M., 2017. Elucidating reaction mechanisms on quantum computers. Proc. Natl. Acad. Sci. 114 (29), 7555–7560.

Sakshaug, J.W., Schmucker, A., Kreuter, F., Couper, M.P., Singer, E., 2016. Evaluating active (opt-in) and passive (opt-out) consent bias in the transfer of federal contact data to a third-party survey agency. J. Surv. Statist. Methodol. 4 (3), 382–416.

Steiger, D.S., Häner, T., Troyer, M., 2018. Projectq: an open source software framework for quantum computing. Quantum 2, 49.

Sugar, W., 2014. Studies of ID Practices: A Review and Synthesis of Research on ID Current Practices. Springer.

2021. Understanding the complexity of quantum circuit compilation. https://www.ibm.com/blogs/research/2018/08/understanding-complexity-quantum-circuit-compilation/ (Accessed: 2021-06-05).

Walker, J.L., 2012. Research column. The use of saturation in qualitative research.. Canad. J. Cardiovasc. Nurs. 22 (2).

Wecker, D., Hastings, M.B., Troyer, M., 2015. Progress towards practical quantum variational algorithms. Phys. Rev. A 92 (4), 042303.

2021. What to look for in a quantum machine learning framework. https://bit.ly/2ZkC0jr (Accessed: 2021-06-05).

Ying, M., 2016. Foundations of Quantum Programming. Morgan Kaufmann.

Zhao, J., 2020. Quantum software engineering: Landscapes and horizons. arXiv preprint arXiv:2007.07047.

Zhao, J., 2021. Some size and structure metrics for quantum software. arXiv preprint arXiv:2103.08815.

Zhao, P., Zhao, J., Ma, L., 2021. Identifying bug patterns in quantum programs. arXiv preprint arXiv:2103.09069.

Manuel De Stefano is a Ph.D. Student at the Software Engineering (SeSa) Lab of the University of Salerno, Italy, under the supervision of Professor Andrea De Lucia. He received a bachelor's and master's degree in computer science from the University of Salerno, Italy. His research interests include software code quality, predictive analytics, mining software repositories, software maintenance and evolution, and empirical software engineering.

Fabiano Pecorelli is a Researcher at Tampere University, Finland. He received a bachelor's and master's degree in computer science from the University of Salerno, Italy. In 2018, he started a Ph.D. at the University of Salerno, under the supervision of Professor Andrea De Lucia. He has already submitted a Ph.D. Thesis about technical debt to be defended in March 2022. His research interests include software code and test code quality, predictive analytics, mining software repositories, software maintenance and evolution, and empirical software engineering.

Dario Di Nucci is an Assistant Professor at the Software Engineering (SeSa) Lab of the University of Salerno, Italy. His research is on empirical software engineering, particularly software maintenance and evolution and software testing, where he applies machine learning, search-based algorithms, and mining of software repositories. He is a member at large of the IEEE Conference Activities Committee. He serves as editor for Elsevier Journal of Systems and Software (JSS) and Elsevier Science of Computer Programming (SCICO), he is an editorial board member of IEEE Software, program committee member for international conferences, and referee for international journals in software engineering and artificial intelligence, for which he received several Distinguished/Outstanding Reviewer Awards.

Fabio Palomba is an Assistant Professor at the Software Engineering (SeSa) Lab of the University of Salerno, Italy. He received the European Ph.D. degree in Management & Information Technology in 2017. His Ph.D. Thesis was the recipient of the 2017 IEEE Computer Society Best Ph.D. Thesis Award. His research interests include software maintenance and evolution, empirical software engineering, source code quality, and mining software repositories. He was the recipient of two ACM/SIGSOFT and one IEEE/TCSE Distinguished Paper Awards at the IEEE/ACM International Conference on Automated Software Engineering (ASE'13), the International Conference on Software Engineering (ICSE'15), and the IEEE International Conference on Software Maintenance and Evolution (ICSME'17), respectively, and Best Paper Awards at the ACM Computer Supported Cooperative Work (CSCW'18) and the IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER'18). In 2019 he was the recipient of an SNSF Ambizione grant, one of the most prestigious individual research grants in Europe. He is a member of the Steering Committee of ICPC (elected in 2021). He has been program co-chair of ICPC 2021, industrial track co-chair of SANER 2022, NIER/ERA track co-chair of ASE 2022, SCAM 2022, and MobileSoft 2022, FOSS Award co-chair of MSR 2022, other than program co-chair of MaLTeSQuE 2018 and 2019. In addition, he has been a member of the organizing committee of ICPC 2015 and SANER 2018. Since 2021, he is Editorial Board Member of the Springer's Empirical Software Engineering Journal (EMSE), where he already was Review Board Member since 2016, and the e-Informatica Software Engineering Journal (EISEJ). Since 2020 he is Review Board Member of the IEEE Transactions on Software Engineering. Since 2019, he is Editorial Board Member of ACM Transactions on Software Engineering and Methodology (TOSEM), Elsevier's Journal of Systems and Software (JSS), and Elsevier's Science of Computer Programming (SCICO). For his reviewing activities, he was the recipient of 12 Distinguished/Outstanding Reviewer Awards.

Andrea De Lucia is a Full Professor of Software Engineering at the University of Salerno, Italy, Head of the Software Engineering (SeSa), and Director of the Ph.D. program in Computer Science. He received a Ph.D. degree in electronic engineering and computer science from the University of Naples Federico II, Italy, in 1996. His research interests include software maintenance and testing, reverse engineering and reengineering, source code analysis, code smell detection and refactoring, defect prediction, traceability management, visual modeling languages, and collaborative software development. He has published more than 250 papers on these topics in international journals, books, and conference proceedings and has edited books and journal special issues. Prof. De Lucia is co-editor in chief of Science of Computer Programming (Elsevier) and serves on the editorial board of Empirical Software Engineering (Springer) and Journal of Software Evolution and Process (Wiley). He also serves on the organizing and program committees of several international conferences in the field of software engineering. Prof. De Lucia is a senior member of the IEEE and was a member-at-large of the executive committee of the IEEE Technical Council on Software Engineering.