POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Empirical Studies of Quantum Programming Issues

MOHAMED RAED EL AOUN
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Août 2022

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Empirical Studies of Quantum Programming Issues

présenté par **Mohamed Raed EL AOUN**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Ettore MERLO, président
Foutse KHOMH, membre et directeur de recherche
Heng LI, membre et codirecteur de recherche
Mohammad HAMDAQA, membre

DEDICATION

To my beloved mother, my beloved father (deceased),
for all their sacrifices, their love,
their tenderness, support, and prayers
throughout my studies,
Let this work be the fulfillment
of your so much alleged wishes,
and flees from your unfailing support.
To all my lab friends,
I will miss you...

ACKNOWLEDGEMENTS

This thesis would not have been possible without the help of several people whom I would like to thank.

First of all, I would like to express my gratitude to my supervisors Prof. Foutse Khomh and Prof. Heng Li for all they have done for me throughout my master's studies.

Then, I would also like to thank Mr. Moses Openja and Dr. Lionel Tidjon for their collaboration in writing the articles for this thesis. My special thanks to Mr. Houssem Ben Braiek for his help and suggestions during my master's studies.

I would also like to thank all my friends and member of the Computer Engineering and Software Engineering (GIGL) department.

I express my gratitude to my family members and loved ones for their support and inspiration.

Finally, I would like to thank the members of my committee, Prof. Merlo Ettore, Prof. Mohammad Hamdaqa, Prof. Heng Li, and Prof. Foutse Khomh for evaluating my master's thesis.

RÉSUMÉ

Avec les progrès de l'informatique quantique ces dernières années, les logiciels quantiques deviennent essentiels pour explorer le plein potentiel des systèmes informatiques quantiques. Récemment, le génie logiciel quantique (QSE) est devenu un domaine émergent qui attire de plus en plus l'attention. Cependant, on ne sait pas quels sont les défis et les opportunités de l'informatique quantique auxquels est confrontée la communauté du génie logiciel. En tant que nouvelle approche pour effectuer des calculs afin de résoudre plus rapidement des problèmes spécifiques (par exemple, des problèmes d'optimisation combinatoire), la programmation quantique diffère de la programmation classique de plusieurs manières. Par exemple, l'état d'un programme quantique est de nature probabiliste et un ordinateur quantique est sujet aux erreurs en raison de l'instabilité des mécanismes quantiques. Par conséquent, les caractéristiques des problèmes de programmation quantique peuvent être très différentes de celles de la programmation classique.

Cette thèse rapporte deux études empiriques menées dans le but de comprendre les problèmes de programmation quantique auxquels sont confrontés les développeurs de logiciels quantiques. Premièrement, nous effectuons une étude des défis perçus par les développeurs de logiciels quantiques et identifions des opportunités pour la recherche et la pratique du QSE. En particulier, nous examinons les forums de questions-réponses techniques où les développeurs posent des questions liées au QSE, et les rapports de problèmes GitHub où les développeurs soulèvent des problèmes liés au QSE. Deuxièmement, nous effectuons une étude des caractéristiques des bogues dans l'écosystème logiciel quantique et de leur distribution dans les composants du programme quantique.

Pour comprendre les enjeux QSE perçus par les développeurs. Nous réalisons une étude empirique sur les forums Stack Exchange où les développeurs postent des questions et donnent des réponses liées à QSE et des rapports de problèmes Github où les développeurs soulèvent des problèmes liés au QSE dans des projets pratiques d'informatique quantique. Sur la base d'une taxonomie existante des types de questions sur Stack Overflow, nous effectuons d'abord une analyse qualitative des types de questions liées au QSE posées sur les forums Stack Exchange. Nous utilisons ensuite la modélisation automatisée des sujets pour découvrir les sujets dans les publications Stack Exchange liées au QSE et les rapports de problèmes GitHub. Notre étude met en évidence certains domaines particulièrement difficiles du QSE qui sont différents de ceux du génie logiciel traditionnel, tels que l'explication de la théorie derrière le code informatique quantique, l'interprétation des résultats des programmes quantiques et la réduction du fossé des connaissances entre l'informatique quantique et l'informatique classique, ainsi que leur opportunités associées.

Pour ce qui concerne les caractéristiques des bogues survenant dans les projets de logiciels quantiques. Nous avons realisé une étude empirique des rapports de bogues (documentés sous forme de pull requests et de rapports de problèmes) de 125 projets de logiciels quantiques hébergés sur GitHub. Ceci afin de découvrir des informations qui peuvent aider à concevoir des mécanismes de test et de débogage efficaces pour les projets de logiciels quantiques. Nous observons que les projets de logiciels quantiques sont plus bogués que les projets de logiciels classiques comparables et que les bogues des projets quantiques sont plus coûteux à corriger (en termes de lignes de code modifiées) que les bogues des projets classiques. Nous identifions également les types de ces bogues et les composants de programmation quantique (par exemple, la préparation d'état) où ils se sont produits. Notre étude montre que les bogues sont répartis sur différents composants, mais des bogues spécifiques au quantique apparaissent particulièrement dans les composants du compilateur, d'opération de porte et de préparation d'état. Les trois types de bogues les plus fréquents sont les bogues d'anomalie de programme, les bogues de configuration, et les bogues de types et de structure de données. Notre étude met en évidence certaines lacunes actuelles de l'écosystème de développement de logiciels quantiques, tels que le manque de bibliothèques scientifiques de calcul quantique qui implémentent des fonctions mathématiques complètes pour les algorithmes de calcul quantique et les définitions d'opération de porte quantique. Les développeurs quantiques recherchent également des bibliothèques spécialisées dans la manipulation de données (par exemple, la manipulation de tableaux) dédiées à l'ingénierie logicielle quantique telles que `Numpy` pour l'informatique quantique. Nos découvertes fournissent également des informations pour les travaux futurs visant à faire progresser le développement, le test, et le débogage des programmes et logiciels quantiques. Nous espérons qu'elles permettront par exemple de fournir un support d'outillage pour le débogage des circuits de bas niveau.

ABSTRACT

With the advance of quantum computing in recent years, quantum software becomes critical for exploring the full potential of quantum computing systems. Recently, quantum software engineering (QSE) becomes an emerging area attracting more and more attention. However, it is not clear what are the challenges and opportunities of quantum computing facing the software engineering community. As a new approach of performing computation to solve specific problems (e.g., combinatorial optimization problems) faster, quantum programming is different from classical programming in several different ways. For example, the state of a quantum program is probabilistic in nature, and a quantum computer is error-prone due to the instability of quantum mechanisms. Therefore, the characteristics of quantum programming issues may be very different from that of classical programming.

This thesis reports two empirical studies to understand quantum programming issues facing quantum software developers. Firstly, we perform a study of the challenges perceived by quantum software developers and seek opportunities for future QSE research and practice. In particular, we examine technical Q&A forums where developers ask QSE-related questions, and GitHub issue reports where developers raise QSE-related issues. Secondly, we perform a study of the bug characteristics in the quantum software ecosystem and their distribution throughout the quantum program components.

To understand the QSE-related challenges perceived by developers. We perform an empirical study on Stack Exchange forums where developers post QSE-related questions & answers and Github issue reports where developers raise QSE-related issues in practical quantum computing projects. Based on an existing taxonomy of question types on Stack Overflow, we first perform a qualitative analysis of the types of QSE-related questions asked on Stack Exchange forums. We then use automated topic modeling to uncover the topics in QSE-related Stack Exchange posts and GitHub issue reports. Our study highlights some particularly challenging areas of QSE that are different from that of traditional software engineering, such as explaining the theory behind quantum computing code, interpreting quantum program outputs, and bridging the knowledge gap between quantum computing and classical computing, as well as their associated opportunities.

Regarding the characteristics of bugs occurring in quantum software projects. We conduct an empirical study on the bug reports (in the forms of pull requests and issue reports) of 125 quantum software projects hosted on GitHub, in order to provide insights that can help devise effective testing and debugging mechanisms for quantum software projects. We observe that quantum software projects are more buggy than comparable classical software projects and that quantum project bugs are more costly to fix (in terms of the code changed) than classical project bugs. We also identify the types of these bugs and the quantum programming components (e.g., state preparation) where they occurred. Our study shows that the bugs are spread across different components, but quantum-specific bugs particularly appear in the compiler, gate operation, and state preparation components. The three most occurring types of bugs are Program anomaly bugs, Configuration bugs, and Data type and structure bugs. Our study highlights some particularly challenging areas in quantum software development that are different from traditional software development, such as the lack of scientific quantum computation libraries that implement comprehensive mathematical functions for quantum computing algorithms and quantum gate operation definitions. Quantum developers also seek specialized data manipulation (e.g, array manipulation) libraries dedicated to quantum software engineering such as `Numpy` for quantum computing. Our findings also provide insights for future work to advance quantum software development, testing and debugging. We hope that our findings will, for example, help deliver tooling support for debugging low-level circuits.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

QSE	Quantum software engineering
Q&A	question and answers

CHAPTER 1 INTRODUCTION

In recent years, the development of quantum software engineering (QSE) has shown significant progress. This breakthrough is primarily driven by the achievement of a series of important milestones in recent years. For example, D-Wave claimed the first commercial quantum computer in 2011 [5]. Tech giants such as IBM, Amazon, Google, and Microsoft are racing to build their quantum computers. Quantum computers are expected to make revolutionary computation improvements over modern classical computers in certain areas, such as optimization, simulation, and machine learning [6, 7]. The rapid development of quantum computers has driven the development of quantum programming languages and quantum software [3], with many of them released as open source [8]. A variety of quantum programming frameworks and languages have been introduced, such as Qiskit [9], Cirq [10], and Q# [11]. For example, IBM's Qiskit is a Python-based software development toolkit for developing quantum applications that can run on quantum simulators or real quantum computers (e.g., IBM Quantum Cloud). Quantum software, by its nature, is drastically different from classical software. For instance, the classical software system is executed sequentially and the status of the system is typically deterministic. However, a quantum software system is intrinsically parallel and can have multiple possible states at the same time [12]. In addition, quantum computers are error-prone due to the instability of quantum mechanisms, hence the output of a quantum software system is often noisy [13]. Thus, the challenges and issues of quantum programming may possess unique characteristics that are very different from those in classical programming.

This thesis examines the technical Q&A forums and GitHub issue reports in depth. In fact, we look to understand the intention behind developers' questions on technical forums and the types of information that they are seeking, and their faced challenges. We also examine the quantum software ecosystem bugs to build a taxonomy for the quantum bugs and understand the unique characteristics of bugs occurring in quantum programs.

1.1 Quantum Software Engineering Challenges

Quantum computing is expected to help solve the computational problems that are difficult for today's classical computers, including issues in cryptography, chemistry, financial services, medicine, and national security [14]. In the classical computing world, a modern CPU is nearly useless without an operating system and software tools for developing applications, and we can assume that this will also be the case for quantum computers. Without powerful software, quantum computing will fail to deliver on its promise. Quantum programming languages like Q# from Microsoft, Qiskit from IBM, or Cirq from Google primarily operate at the gate or building-block level. If a required building block is not yet implemented, the user needs to specify the exact sequence of interconnections between qubits and quantum gates. The complexity in writing quantum software has another unfortunate side effect: because quantum programming is unlike classical programming, quantum software engineers are a rare breed. They need a level of knowledge in quantum information theory and have a working understanding of quantum physics as well as a mastery of linear algebra. Furthermore, quantum software engineers need domain expertise in option pricing, molecular biology, supply-chain optimization, or whatever problem the teams set out to solve. The need to define new algorithms at the gate level makes it very difficult to integrate domain-specific experts into quantum teams. It is therefore expected to observe an increase in the prevalence of discussion about quantum software engineering in the technical Q&A forums.

In this chapter, we perform a large-scale empirical study of quantum computing posts on technical Q&A forums and GitHub issue reported by leveraging topic modeling.

1.2 Quantum Software Ecosystem and Quantum Bugs

In modern software development, debugging and testing play a critical part to ensure good quality and performance. A software bug is an abnormal behavior that deviates from the specification of a program [15]. Software bugs impact massively the performance, security, and quality of a program. Even though quantum programs, by their nature, are drastically different from classical software programs, they do not escape the risks that bugs cause. Quantum computing is fancy, and various efforts are currently competing to build the quantum software ecosystem (e,g. Qiskit [9], Cirq [10], and Q# [11]). With the continuous growth of this field, ensuring correctness is becoming more and more a high priority. Testing and debugging, which are two approaches used to prevent and find bugs in classical software can be applied to quantum software. However, an important pillar to preventing and detecting bugs is understanding what bugs exist in the quantum software ecosystem. Quantum software is probabilistic by nature which makes it drastically different from classical software. For example, a classical software system is executed sequentially and the status of the system is typically deterministic. However, a quantum software system is intrinsically parallel and uses the qubits which can have multiple possible states at the same time [12]. Also, because quantum computers are error-prone due to the instability of quantum mechanisms, the measurement of the qubit state of a quantum software system is often noisy [13]. Therefore, the bugs of quantum software can possess unique characteristics that are very different from those in classical software.

In this chapter, we perform an empirical study on 125 open-source quantum software projects hosted on GitHub to understand the characteristics of bugs occurring in quantum software projects.

1.3 Thesis Statement

Although software engineering challenges, as well as software bug characteristics, have been widely investigated for classic software systems, no study to date has examined the technical Q&A forums and GitHub issue reports to identify the quantum programming issues and challenges. Besides, there are very few empirical studies on quantum computing bugs characteristics and no prior in-depth study of quantum software bugs. To fill this gap, in this thesis we present two empirical study on technical Q&A forums and 125 open-source quantum software projects hosted on GitHub. These quantum software projects cover a variety of categories, such as quantum programming frameworks, quantum circuit simulators, or quantum algorithms, and present a taxonomy of bug characteristics in the quantum software ecosystem. Our goal is to help researchers and practitioners better understand the challenges and issues of quantum programming for which the demand is increasing rapidly. We hope that our work will encourage software engineering researchers to tackle the most important challenging tasks in quantum software engineering, such as quantum software debugging and testing.

It is critical to understand the extent to which quantum software practitioners are struggling and identify the bug characteristics in the quantum software ecosystem to ensure the correctness of the quantum programs. To the best of our knowledge, this thesis presents the first in-depth empirical study on quantum computing programming.

1.4 Thesis Overview

In this thesis we report two empirical studies. First, we perform a large-scale empirical study of quantum computing posts on technical Q&A forums and GitHub issue reports by applying topic modeling to assimilate the discussed challenges and highlight the most relevant topics that quantum software engineers are facing. Second, using the GitHub projects selected in the first empirical study, we examine the bugs reports. For instance we study the distribution of bugs based on the quantum software components and project types, the duration to fix the bugs, and propose a taxonomy for bugs in quantum computing programs.

In the following we elaborate on each of the studies.

1. `Understanding quantum software engineering challenges:` We introduce the first empirical study that aims to understand the challenges perceived by quantum software developers and seek opportunities for future QSE research and practice. In particular, we examine technical Q&A forums where developers ask QSE-related questions, and GitHub issue reports where developers raise QSE-related issues. We apply a series of heuristics to search and filter Q&A posts that are related to QSE and to search and filter GitHub projects that are related to quantum software. In total, we extract and analyze 3,117 Q&A posts and 43,979 Github issues that are related to QSE. We combine manual analysis and automated topic modeling to examine these Q&A posts and Github issues, to understand the QSE challenges developers are facing. Firstly, we manually examined a statistically representative sample of questions. We extended a previous taxonomy from prior work [16] and found nine categories of questions. Secondly. we use topic models to extract the semantic topics in the technical Q&A forums posts. We derived nine topics including traditional software engineering topics. Finally, we analyze the topics in the GitHub issue reports to understand the challenges developers are facing in practical quantum computing projects.

2. `Bug Characteristics in Quantum Software Ecosystem:` To understand the characteristics of bugs occurring in quantum software projects, we conduct a empirical study on 125 open-source quantum software projects hosted on GitHub. The quantum software projects cover a variety of categories, such as quantum programming frameworks, quantum circuit simulators, or quantum algorithms. An analysis of the development activity of these selected projects show a level of development activities similar to that of classical projects hosted on GitHub. First, we compare the distribution of bugs in quantum software projects and classical software projects, as well as developers' efforts in addressing these bugs. Second, we qualitatively studied a statistically representative sample of quantum software bugs to understand their characteristics. In fact, we analyzed the quantum software components (e.g., quantum measurement) where these bugs occurred, examined the nature of these bugs (e.g., performance bugs) and reported code examples of each bug type. We build a quantum software ecosystem bug type taxonomy in which we identify each detected bug type and map it to the quantum components.

1.5 Thesis Contribution

In this thesis we conduct an empirical study on the quantum software ecosystem challenges, with the focus to understand the difficulties that quantum software practitioner are facing. Our study results in the following observations:

- We identified nine categories of QSE-related questions in technical Q&A forums. The categories `Theoretical`, `Errors`, `Learning`, and `Tooling` are new or become more frequent in QSE-related questions.

- From Q&A forums, we derived nine topics discussed in QSE-related posts, including traditional software engineering topics (e.g., `environment management` and `dependency management`) and QSE-specific ones (e.g., `quantum execution results` and `quantum circuits`).

- Our results identify some particularly challenging areas for QSE, such as interpreting quantum program outputs and bridging the knowledge gap between quantum computing and classical computing.

- QSE-related challenges impact practical quantum program development in GitHub projects. For instance we report that the challenges are generally among the quantum computing projects like quantum programming frameworks, tools, algorithms, and applications.

- Quantum software projects are more buggy than classical software projects. While developers of quantum software projects are actively addressing their bugs, fixing quantum software bugs is more costly than fixing classical software bugs. For instance, projects in the `quantum machine learning`, `quantum programming framework`, and `quantum-based simulation` categories are the most buggy and have the most difficult bugs to fix.

- We propose a taxonomy of quantum bugs containing 13 different types of bugs. Our result shows that the quantum bugs are distributed across different quantum component. For instance program anomaly bugs is the most spread bug type (it occurs in all components). Data type and structure bugs are mostly located in the components `State preparation` (10 occurrences) and `Compiler` (12 occurrences).

- Our study reports that researchers and tool builders should consider contributing specialized `data manipulation (e.g., array manipulation)` libraries and provide `mathematical algorithms for quantum computing and convenience functions` tools to support quantum software development, and reduce the occurrence of program anomaly bugs and data type and structure bugs. `Circuit visualization and analysis` tools are needed to help developers debug and fix bugs in quantum circuits.

1.6 Organization of the Thesis

The remainder of the thesis is organized as follows:

- Chapter 2 introduces the fundamental concepts and requirements related to quantum computing and quantum programming that help understand the research work.
- Chapter 3 presents a comprehensive review of quantum software engineering, quantum programming, and quality assurance of quantum programs.
- Chapter 4 presents our first empirical study, in which we examine technical Q&A forums and GitHub issue reports to identify the most important challenges.
- Chapter 5 presents our second empirical study on quantum software ecosystem bugs.
- Chapter 6 summarizes the thesis and discusses future works.

CHAPTER 2 BACKGROUND

This theses aims to understand the quantum software engineering challenges by examining technical forum posts and GitHub issue reports. In this chapter, we introduce the background knowledge on quantum software engineering, quantum programming, and quality assurance in quantum and classical software. This will be helpful to follow the rest of this thesis.

2.1 Quantum Computing

Quantum computers aim to leverage the principles of quantum mechanics such as `superposition` and `entanglement` to provide computing speed faster than today's classical computers. While classical computers use bits in the form of electrical pulses to represent 1s and 0s, quantum computers use quantum bits or **Qubits** in the form of subatomic particles such as electrons or photons to represent 1s and 0s. A Qubit, unlike a classical bit, can be 0 or 1 with a certain probability, which is known as the **superposition** principle [17].

In other words, a quantum computer consisting of Qubits is in many different states at the same time. When a Qubit is **measured**, it collapses into a deterministic classical state. The status of two or more Qubits can be correlated (or entangled) in the sense that changing the status of one Qubit will change the status of the others in a predictable way, which is known as the **entanglement** phenomenon [17].

The `superposition` and `entanglement` phenomenons give quantum computers advantages over classical computers in performing large-scale parallel computation [17].

Similar to classical logic gates (e.g., `AND`, `OR`, `NOT`), **quantum logic gates** (or **quantum gates**) alter the states (the probability of being 0 or 1) of the input Qubits. Like classical digit circuits, **quantum circuits** are collections of quantum logic gates interconnected by quantum wires. Figure 2.1 illustrates the architecture of a quantum computer [2, 3]. The architecture contains two layers: a quantum computing layer where the quantum physics and circuits reside, and a classical computing layer where the quantum programming environment and software applications reside.

- *Physical building blocks*: physical realization of Qubits and their coupling/interconnect circuitry.
- *Quantum logic gates*: physical circuitry for quantum logic gates.

- *Quantum-classical computer interface*: the hardware and software that provides the boundary between classical computers and the quantum computing layer.
- *Quantum programming environment*: quantum programming languages and development environment.
- *Business applications*: quantum software applications (based on quantum programming languages) that meet specific business requirements.

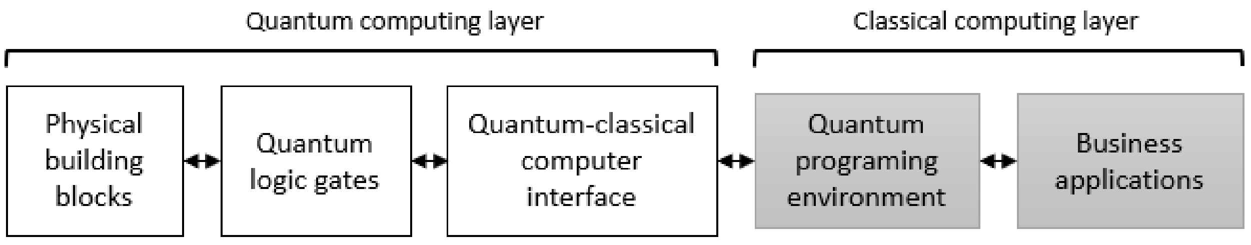
Quantum computing layer
Classical computing layer
Physical building blocks
Quantum logic gates
Quantum-classical computer interface
Quantum programing environment
Business applications

Figure 2.1 The architecture of a quantum computer [2, 3]

Beside, Figure 2.2 illustrates an example quantum circuit with two qubits (*q0* and *q1*). Below, we describe the three parts of the circuit:

1. *Reset and initialization*: The states of the two qubits are initialized as 0s.
2. *Quantum gate*: A Hadamard (***H***) gate denoted by a blue square is applied on the qubit *q0*. The gate ***H*** generates a superposition state with equal probabilities for the states of 1 and 0. Then, a controlled-NOT ***CX*** gate represented by a blue circle is applied on qubits *q0* and *q1*: the state of qubit *q1* is flipped if and only if the state of qubit *q0* is 1. Thus, the *CX* gate creates entanglement between the pair of qubits *q0* and *q1* (i.e., the state of one qubit is predictable from the state of the other).
3. *Measurement*: The measurement collapses the state of a qubit from a superposition state into a deterministic single state. The output of this step is a qubit with the most probable state.

2.2 Quantum Software Development

Quantum software development typically follows the `quantum gate-model` to accomplish a certain task [4]. In this model, the problem is expressed in terms of quantum gates (described in Section 2). In Figure 2.3, we illustrate an example of a quantum algorithm workflow using

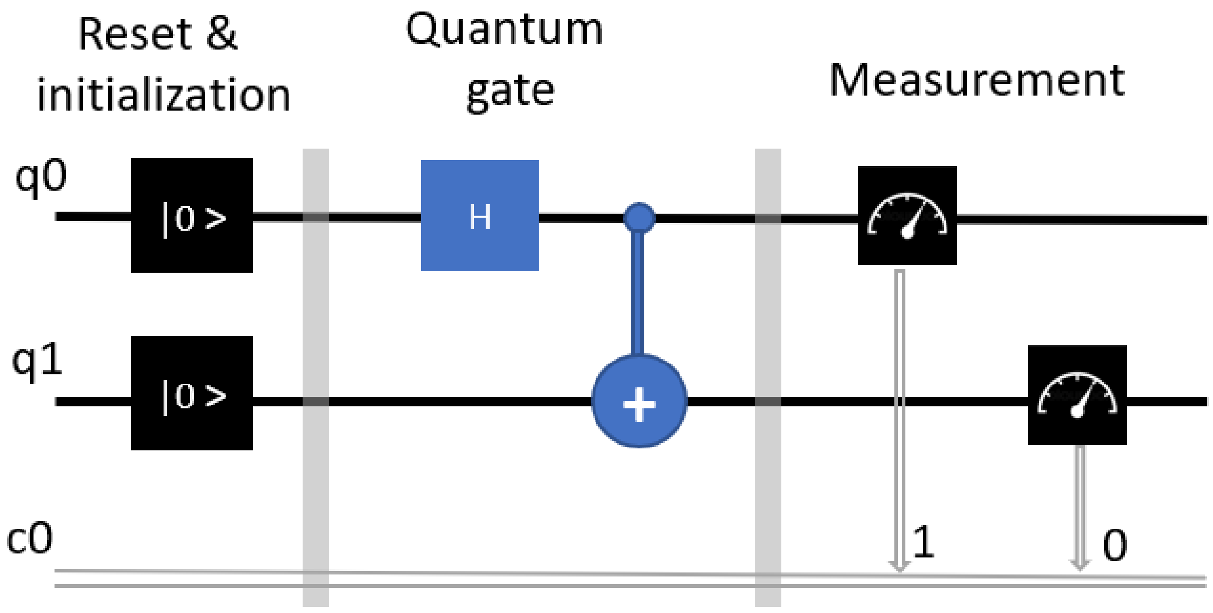

Figure 2.2 An example quantum circuit

the gate model. The first step is to define the problem. In this example, we define the Travelling Salesman Problem (TSP). Secondly, based on the nature of the problem we need to choose the most suitable algorithm to find a solution. In our example, we can consider the quantum approximation algorithm [18]. This algorithm was proposed to find the optimal solutions using the gate model. Next, the quantum algorithm has to be implemented in quantum code which is then compiled into a quantum circuit consisting of quantum gates. Finally, the quantum circuit will be executed on a quantum computer or a simulator running on a classical computer.

```python
import qiskit as q
# create register to store bits
qr = q.QuantumRegister(2)
cr = q.ClassicalRegister(2)
#create the circuit
circuit = q.QuantumCircuit(qr, cr)
#0th index on the quantum register
circuit.h(qr[0])
#apply CX gate (control_bit, target_bit)
circuit.cx(qr[0], qr[1])
# measure quantum bit into cassical bit
circuit.measure(qr, cr)
```

Listing 2.1 Quantum code example based on Qiskit that produces the quantum circuit shown

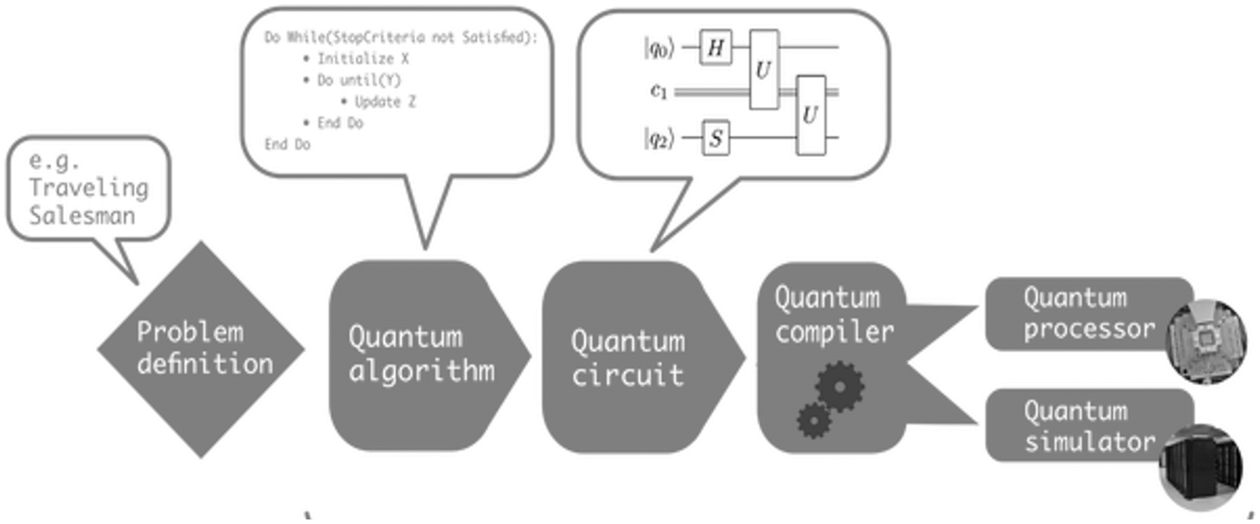

Figure 2.3 Quantum software workflow on a gate-model quantum computer [4]

in Figure 2.2

Listing 2.1 shows a code snippet based on the Qiskit python library that produces the quantum circuit shown in Figure 2.2. First, we import the *Qiskit* library (line 1). Then, lines 3 and 4 create 2 qubits and 2 classical bits which form a quantum circuit in line 6. Line 8 applies the Hadamard (H) gate on the first qubit, which generates a superposition of the qubit with equal probabilities of being 1 and 0. Then, line 10 applies a controlled-NOT (CX) gate on the output of the Hadamard gate and the second qubit, which generates entanglement between the two qubits. Finally, line 12 measures the final states of the two qubits and maps the measurement results to the two classical bits.

2.3 Quantum Computing Ecosystem

Quantum computing aims to solve problems that are challenging for a classical computer using principles of quantum mechanics. The Quantum computing ecosystem is starting to take shape, coalescing around hardware (quantum computers), software such as quantum programming frameworks, quantum programming languages, utility tools, and libraries (i.e., testing, error mitigation). In Figure 2.4, we present an overview of the state of the quantum ecosystem nowadays. This overview was inspired by Qiskit [9], IBM-Q [19], and an online article [20]. From Figure 2.4, we can see that the quantum ecosystem has two different layers. A physical layer that includes both quantum and classical hardware, as well as a logical layer that covers the quantum computing software. The physical layer contains:

- **Physical quantum processor:** A quantum circuit on a chip with a size of hundreds

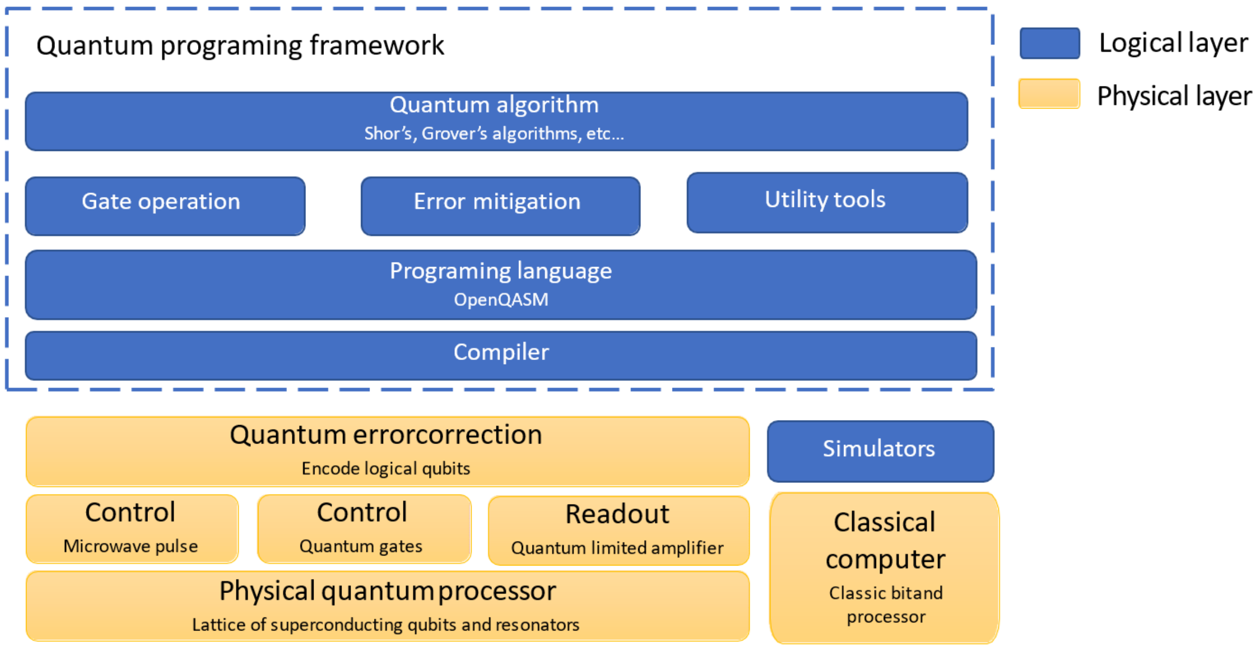

Figure 2.4 Quantum computing ecosystem overview

of nanometers of quantum elements such as atoms and molecules known as qubit [21].

- **Microwave pulse:** Device used to generate pulses to control and measure qubits fabricated on superconducting circuits [22].
- **Quantum gates:** Physical quantum gates and building blocks of quantum circuits [23]
- **Quantum limited amplifier:** Amplification of the quantum signals (pulse) while adding the minimum amount of noise tolerated by quantum mechanics [24].
- **Quantum error correction:** Encode the logical qubits into multiple physical qubits while protecting the quantum states and actively correcting the errors [25].
- **Classical computer:** Traditional computer stores information in classical bits that are represented logically by either a 0 (off) or a 1 (on) [26].

The logical layer contains:

- **Simulator:** Libraries and software to simulate the quantum computer behavior on a classical computer.
- **Compiler:** Tools and software used to compile and optimize the quantum circuits.
- **Programming language:** Quantum programming languages (e.g., Q# [11]) are implemented in development kits to support the development of quantum algorithms.

- **Gate operation:** A set of unitary operations that are used to control the state of qubits.
- **Error mitigation:** Libraries and algorithms for software-based quantum error correction.
- **Utility tools:** Libraries that are used to support quantum software development activities, such as testing and debugging.
- **Quantum algorithms:** A collection of quantum algorithms that runs on top of a quantum computer or a simulator.

CHAPTER 3 LITERATURE REVIEW

In this chapter, we review the related literature on quantum software engineering, software quality assurance, and topic modeling in software engineering studies.

3.1 Quantum Software Engineering

Quantum software engineering (QSE) is still in its infancy. As the result of the first International Workshop on Quantum Software Engineering & Programming (QANSWER), researchers and practitioners proposed the "Talavera Manifesto" for quantum software engineering and programming, which defines a set of principles about QSE [14], including:

(1) QSE is agnostic regarding quantum programming languages and technologies; (2) QSE embraces the coexistence of classical and quantum computing; (3) QSE supports the management of quantum software development projects; (4) QSE considers the evolution of quantum software; (5) QSE aims at delivering quantum programs with desirable zero defects; (6) QSE assures the quality of quantum software; (7) QSE promotes quantum software reuse; (8) QSE addresses security and privacy by design; and (9) QSE covers the governance and management of software.

Zhao [3] performed a comprehensive survey of the existing technology in various phases of quantum software life cycle, including requirement analysis, design, implementation, testing, and maintenance. Prior work [7,12,27,28] also discussed challenges and potential directions in QSE research, such as modeling [28] and quantum software processes & methodologies [12], and design of quantum hybrid systems [7]. In addition, prior work conducted extensive exploration along the lines of quantum software programming [29] and quantum software development environments [30]. The survey [3] provides an comprehensive overview of the work along these lines. Different from prior work, firstly, this thesis makes the first attempt to understand the challenges of QSE perceived by practitioners. Secondly, this thesis examines the characteristics of bugs occurring in the quantum software ecosystem.

3.2 Quantum Programming

Quantum computing as a new general paradigm can massively influence how software is developed [3, 14, 29]. Quantum programming is the process to design an executable quantum program to accomplish a specific task [29]. Every block of code is composed of classical and quantum operations [3]. Classical operations act on classical bits in order to register the states and measurements of qubits, while quantum operations operate on the quantum computers using registers of qubits. Quantum programming uses syntax-based notations to represent and operate quantum circuits and gates. Early efforts of quantum programming language development focused on the quantum Turing machine [31] but did not produce practical quantum programming languages. Later efforts have turned to the quantum circuits model where the quantum system is controlled by a classical computer [32]. This concept has given birth to many new quantum programming languages such as qGCL [33], LanQ [34], Q# [35] and Qiskit [9]. Prior work conducted extensive exploration along the lines of quantum programming [29] and quantum software development environments [30]. The survey [3] also provides a comprehensive overview of research works along these lines.

Like classical computing, QSE is not limited to quantum programming and quantum software development methods are thriving. To overcome the challenges in the software development process and ensure the high quality of quantum software, a series of steps are followed in the shape of a life cycle, known as `Quantum Software Life Cycle` (QSDLC) [36]: Software requirement analysis, software design, software implementation, software testing, and software maintenance. The model begins with the requirement analysis step were developers discuss the requirement to be developed in order to achieve their goal. In fact, the scope of the work is defined along with the requirements that are going to be satisfied. The models follow with the design step, this is where the architectural and detailed design is made [36]. At the implementation step, the developer starts coding following the requirement and design agreed on in the previous steps. To detect defects in the software and verify the behavior of the software, the testing step comes to action before releasing the system. Finally, as the last step, maintenance represents the changes and the modifications after the release of the quantum software [36], [3], [14].

3.3 Quality assurance in quantum and classical software

In this section, we briefly discuss the existing work related to quantum bugs. Next, we give an overview of quantum testing and discuss its challenges and the proposed solutions.

3.3.1 Quantum Bugs Characteristics

In their 2021 position paper, Campos and Souto [37] argued for the creation of a benchmark dataset of quantum bugs. In the same year, Zhao et al. [38] provided a data set of 36 bugs identified in the quantum computing framework Qiskit. In 2022, Matteo and Michael [39] examined 283 bugs from 18 open-source quantum computing platforms to identify bug patterns. In this thesis, we study a larger set of bugs from a larger number of quantum projects. We perform quantitative and qualitative analysis on the characteristics and types of the quantum bugs occurring in different quantum components and formulate recommendation for researchers, tool builders, and practitioners.

3.3.2 Quantum Programs Testing

Identifying bugs in the programs that run on quantum computers (i.e, quantum programs) can prove to be helpful for researcher to improve the quality and understand quantum software challenges. Quantum programs are more difficult to test and debug than a classical program because of the impossibility to copy the quantum information in the qubits [40], and the probabilistic nature of the measurement. To face these challenges, Huang and Martonosi [41] introduced statistical assertions that can be used to validate patterns and detect bugs in quantum programs. Li et al. [42] proposed `Proq`, a project-based run-time analysis tool for testing and debugging quantum programs. The evaluation of the tool shows that it can effectively help locate bugs in quantum programs. Yu and Palsberg [43] proposed an abstract interpretation of quantum programs and use it to automatically verify assertions in polynomial time. Similar to our work, these previous works on quantum program testing contribute to improving our understanding of the nature of quantum bugs.

3.3.3 Studying bugs characteristic

Prior works studied different kinds of bugs related to our study. Most of the studies are widely related to bugs in frameworks and platforms. Chou et al. [44] present bugs in operation systems, Sun et al [45] recent work studied compiler bugs, and Islam et al. [46] discuss the bugs in deep learning libraries. In our work, we cover domain-specific bugs which are similar to prior works that studied bugs in various platform applications.

3.4 Topic Modeling in Software Engineering Studies

Topic modeling has been extensively used recently in software engineering studies. In the following, we review some of the recent works.

3.4.1 Topic Analysis of Issue Reports

Issue reports have been widely explored in prior work. Here we focus on studies that apply topic analysis on issue report data. Prior work leverages topic models to automatically assign issue reports to developers (*a.k.a.* bug triage) [47–50]. These studies first uses topic models to categorize the textual information in the issue reports, then learn mappings between the categorized textual information and developers. Prior work also leveraged topic models to automatically detect duplicate issue reports based on the similarity of their topics [51–53]. Nguyen et al. [54] use topic models to associate issue reports and source code based on their similarities, in order to help developers narrow down the searched source code space when resolving an issue. Finally, prior work also studied the trends of topics in issue reports [55,56]. Latent Dirichlet Allocation (LDA) and its variants are the most popular topic modeling approaches used in these studies. Therefore, we also leverage LDA to extract topics from GitHub issue reports related to QSE.

3.4.2 Topic Analysis of Technical Q&As

Prior work performed rich studies on technical Q&A data, especially on Stack Exchange data [57]. Here we focus on prior work that performs topic analysis on technical Q&A data. Topic models are used extensively in prior work to understand the topics of general Stack Overflow posts and the topic trends [28, 58–60]. Prior work also leveraged topic models to understand the topics of Stack Overflow posts related to specific application development domains, such as mobile application development [61,62], client application development [63], machine learning application development [64], as well as concurrency [65], and security [66] related development. In addition, prior work leveraged topic models to understand non-functional requirements communicated in Stack Overflow posts [67, 68]. Zhang et al. [69] use topic models to detect duplicate questions in Stack Overflow. Finally, Treude et al. [70] proposes an automated approach to suggest configurations of topic models for Stack Overflow data. Most of these studies use LDA or its variants to extract topics from the technical Q&A data. In this work, we also leverage the widely used LDA algorithm to extract topics from the technical Q&A data related to quantum software engineering.

CHAPTER 4 UNDERSTANDING QUANTUM SOFTWARE ENGINEERING CHALLENGES: AN EMPIRICAL STUDY ON STACK EXCHANGE FORUMS AND GITHUB ISSUES

4.1 Introduction

Over the past decades, quantum computing has made steady and remarkable progress [3, 71, 72]. For example, IBM Quantum [73] now supports developers to develop quantum applications using its programming framework and execute them on its cloud-based quantum computers. Based on the quantum mechanics principles of `superposition` (quantum objects can be in different states at the same time) [74] and `entanglement` (quantum objects can be deeply connected without direct physical interaction) [75], quantum computers are expected to make revolutionary computation improvement over today's classical computers [6]. In particular, quantum computing is expected to help solve the computational problems that are difficult for today's classical computers, including problems in cryptography, chemistry, financial services, medicine, and national security [14]. The success of quantum computing will not be accomplished without quantum software. Several quantum programming languages (e.g., QCL [76]) and development tools (e.g., Qiskit [9] have been developed since the first quantum computers. Large software companies like Google [77], IBM [73], and Microsoft [78] have developed their technologies for quantum software development. Quantum software developers have also achieved some preliminary success in applying quantum software to certain computational areas (e.g, machine learning [79], optimization [80], cryptography [81], and chemistry [82]). However, there still lacks large-scale quantum software. Much like Software Engineering is needed for developing large-scale traditional software, the concept of Quantum Software Engineering (QSE) has been proposed to support and guide the development of large-scale, industrial-level quantum software applications. This concept has been gaining more and more attention recently [3, 14, 27]. QSE aims to apply or adapt existing software engineering processes, methods, techniques, practices, and principles to the development of quantum software applications, or create new ones [14]. Pioneering work sheds light on new directions for QSE, such as quantum software processes & methodologies [12], quantum software modeling [28], and design of quantum hybrid systems [7]. In the meanwhile, we observe an exponential increase of discussions related to quantum software development on technical Q&A forums such as Stack Overflow(e.g. from 8 in 2010 to 1434 in 2020). We also notice an increasing number of quantum software projects hosted on GitHub, where developers use issue reports to track their development and issue fixing processes.

Such technical Q&As and issue reports may communicate developers' faced challenges when developing quantum software applications.

In this chapter, we aim to understand the challenges perceived by quantum software developers and seek opportunities for future QSE research and practice. In particular, we examine technical Q&A forums where developers ask QSE-related questions, and GitHub issue reports where developers raise QSE-related issues. We apply a series of heuristics to search and filter Q&A posts that are related to QSE and to search and filter GitHub projects that are related to quantum software. In total, we extract and analyze 3,117 Q&A posts and 43,979 Github issues that are related to QSE. We combine manual analysis and automated topic modeling to examine these Q&A posts and Github issues, to understand the QSE challenges developers are facing.

In particular, this chapter aims to answer the three following research questions (RQs):

RQ1: *What types of QSE questions are asked on technical forums?*

To understand the intention behind developers' questions on technical forums and the types of information that they are seeking, we manually examined a statistically representative sample of questions. We extended a previous taxonomy from prior work [16] and found nine categories of questions. Our results highlight the need for future efforts to support developers' quantum program development, in particular, to develop learning resources, to help developers fix errors, and to explain the theory behind quantum computing code.

RQ2: *What QSE topics are raised in technical forums?*

The QSE-related posts may reflect developers' challenges when learning or developing quantum programs. To understand their faced challenges, we use topic models to extract the semantic topics in their posts. We derived nine topics including traditional software engineering topics (e.g., `environment management` and `dependency management`) and QSE-specific topics (e.g., `quantum execution results` and `quantum vs. classical computing`). We highlighted some particularly challenging areas for QSE, such as interpreting quantum program outputs, understanding quantum algorithm complexity, and bridging the knowledge gap between quantum computing and classical computing.

RQ3: *What QSE topics are raised in the issue reports of quantum-computing projects?*

Issue reports of quantum computing projects record developers' concerns and discussions when developing these projects. Thus, we analyze the topics in the issue reports to understand the challenges are developers facing in practical quantum computing projects. We observe that the QSE-related challenges that we derived from forum posts indeed impact practical quantum program development in these GitHub projects, while GitHub issues bring new perspectives on developers' faced challenges (e.g., on specific quantum computing applications such as machine learning). We also observe that such challenges are general among quantum computing projects.

Chapter organization. The rest of the chapter is organized as follows. In Section 4.2 we describe the design of our study. In Section 4.3 we present our results. Section 4.4 we discuss the implication of our findings. Section 4.5 review threats to the validity of our findings. Finally, Section 4.6 summarize the chapter.

4.2 Experiment Setup

This section describes the design of our empirical study.

4.2.1 Overview

Figure 4.1 provides an overview of Chapter 4 empirical study. We study QSE-related posts on Stack Exchange (SE) forums and the issue reports of quantum computing GitHub projects. From Stack Exchange forums, we first use tags to filter QSE-related posts. In RQ1, we manually analyze a statistically representative sample of these posts to understand the type of information sought by developers. In RQ2, we use automated topic models to analyze the topics of these posts and their characteristics. From GitHub repositories, we first apply a set of heuristic rules to filter the quantum computing projects. Then we extract the issue reports of these quantum computing projects. Finally, we perform topics modeling on these issue reports to analyze the topics in the textual information of the issue reports (RQ3). We describe the details of our data collection and analysis approaches in the rest of this section.

4.2.2 Stack Exchange forums data collection

We follow three steps to collect QSE related data from Stack Exchange forums. First, we collect Q&A data from four Stack Exchange forums. Second, we identify a set of tags that are related to QSE. Finally, we use the identified tags to select the posts that are related to QSE. We explain the steps below.

Step 1: Collecting technical Q&A data. We extract technical Q&A data from four Stack Exchange forums: Stack Overflow [83], Quantum Computing Stack Exchange [84], Computer

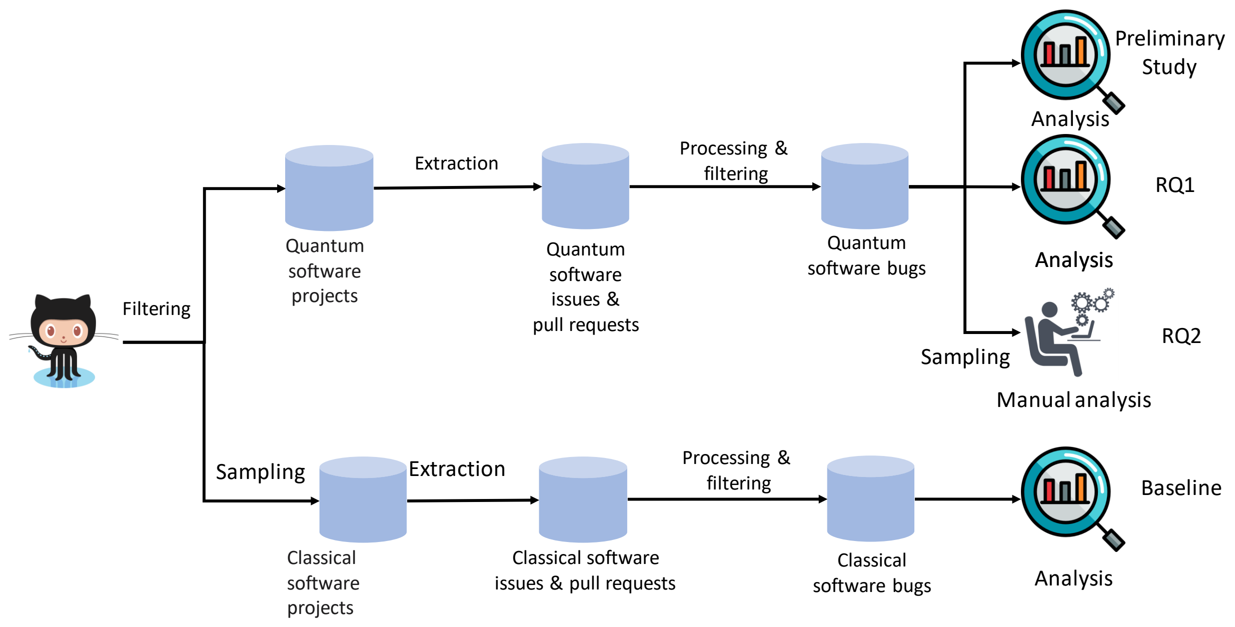

Figure 4.1 Schematic diagram of chapter 4 empirical study

Science Stack Exchange [85], and Artificial Intelligence Stack Exchange [86]. We consider the Stack Overflow forum as it contains posts related to quantum programming and it is widely used for studying various software engineering topics (e.g., mobile app development [61, 62], machine learning application development [64], etc.). We consider the other three forums because they contain posts that discuss topics related to quantum computing and quantum programming.

We extracted the post data from these forums with the help of the Stack Exchange Data Explorer [87]. Stack Exchange data explorer holds an up to date data for these forum between 08-2008 and 03-2021.

Step 2: Identifying tags related to QSE.

The studied Stack Exchange forums use user-defined tags to categorize questions. We follow two sub-steps to select the tags that are related to QSE. We started by searching for questions with the tag "quantum-computing" in the entire Stack Exchange dataset D through the data exchange explorer. We obtained 254 questions tagged with "quantum-computing" from the studied forums. After manually inspecting the 30 most voted questions, we selected an initial tag set T_{init} consisting of ten tags including "quantum-computing", "qiskit", "qsharp", "q#", "quantum-development", "quantum-circuit", "ibmq", "quantum-ai", "qubit" and "qutip". Then we extracted the questions related to T_{init} from the initial dataset D and obtained a new set of questions P. In order to expand the initial tag set, we extracted the frequently co-occurring tags with T_{init} from P and build a new tag set T_2.

Not all the tags in T_2 are related to quantum computing. To determine the final tag set T_{final}, following previous work [88] [89], we filter the tags in T_2 based on their relationships when the initial tag set T_{init}. For each tag t in T_2, we calculate:

$$\text{(Significance)}\,\alpha(t) = \frac{\#\text{ of questions with tag } t \text{ in } P}{\#\text{ of questions with tag } t \text{ in } D} \tag{4.1}$$

$$\text{(Relevance)}\,\beta(t) = \frac{\#\text{ of questions with tag } t \text{ in } P}{\#\text{ of questions in } P} \tag{4.2}$$

To select a tag t, the value of significance-relevance $\alpha(t)$, $\beta(t)$ need to be higher than a threshold we set. To select the optimal threshold values for α and β, we experimented with a set of values respectively between 0.05, 0.35 and 0.001, 0.03. For each α and β and for each tag above the threshold, we inspected the top 10 most voted posts and verified if the tag is related to QSE, we ended up with the optimal threshold

respectively equal to 0.005 and 0.2 which are consistent with previous work [90] [89]. The final tag set T_{final} is formed of 37 tags in total. Since quantum computing is a wide topic and our focus is QSE, we further manually inspected the description of each tag t in T_{final} and the top 10 questions of each tag in each studied forum to remove tags that are not related to QSE. Finally our tag set T_{final} was reduced from 37 to 18 tags (14 unique tags as different forums have tags with the same names). Table 4.1 lists our final set of tags.

Step 3: Selecting questions and answers. We extract the final sets of questions and answers using the final tag sets shown in Table 4.1. We select all the posts that are tagged with at least one of the tags. We ended up with a total of 3,117 questions and answers from the four considered forums in our data set D_{final}. 35% of the final data are answers where 65% are questions. The number of posts (questions and answers) extracted from each forum is shown in Table 4.1.

4.2.3 GitHub issues data collection

In this work, we study the issue reports of quantum computing projects on GitHub. We downloaded the GitHub selected quantum computing projects issues in March 2021.

We follow three steps described below to extract the issue reports of quantum computing projects from GitHub.

Step 1: Searching candidate projects.

Table 4.1 Our selected tags and the number of questions and answers

Stack Ex. forum	**Tag set**	**#Q**	**#A**
Stack overflow	post-quantum-cryptography, q#, quantum-computing, qiskit, qcl, qutip, qubit, tensorflow-quantum	250	183
Quantum computing	programming, classicalcomputing, q#, qiskit, cirq, ibm-q-experience, machine-learning, qutip	1534	778
Computer science	quantum-computing	238	117
Artificial intelligence	quantum-computing	13	4

We search for quantum computing related projects using three criteria: 1) The description of the project must be in English (i.e., for us to better understand the content). 2) The project name or description must contain the word "quantum" (the word quantum is case sensitive in the project name or description). 3) The project is in a mainline repository (i.e., not a fork of another repository). We end up with a total of 1,364 repositories.

Step 2: Filtering quantum computing projects. We filter the searching result and identify quantum computing related projects with three criteria:

1) To avoid selecting student assignments, following previous work [91] [92], we select repositories that were forked at least two times. 2) The projects must have a sufficient history of development for us to analyze the issue reports. Therefore, we select the projects that were created at least 10 months earlier than the data extraction date. Moreover, only the projects that have at least 100 commits and 10 issues are selected.

3) To ensure the quality of the project selected, we manually inspect the projects' descriptions and remove projects that are not related to quantum computing, projects that are created for hosting quantum computing related documentation, as well as lecture notes related to quantum computing. Finally, we obtain a total of 122 projects directly related to quantum computing applications.

Step 3: Extracting issue reports. We use the GitHub Rest API [93] to extract all the issue reports of the final 122 projects on GitHub. In total, we obtain 43,979 issue reports.

4.2.4 Data pre-processing for topic modeling

We build one topic model on the Stack Exchange forum data and another topic model on the GitHub issue data. Below we describe how we pre-process these two types of data before feeding them into topic models.

Pre-processing Q&A post data. We treat each post (i.e., a question or an answer) as an individual document in the topic model. For each question, we join the title and the body of the question to create a single document. As Q&A posts contain code snippets between <code> and </code> which may bring noise to our topic models, we remove all text between <code> and </code>. We also remove HTML tags (e.g., <p></p>), URLs and images from each post. In addition, we remove stop words (e.g., “like”, “this”, “the”), punctuation, and non-alphabetical characters using the Mallet and NLTK stop words set. Finally, we apply the Porter stemming [94] to normalize the words into their base forms (e.g., “computing” is transformed to “comput”), which can reduce the dimensionality of the word space and improve the performance of topic models [95]

Pre-processing issue report data. We treat each issue report as an individual document in the topic model. We join the title and the body of each issue as a single document. Similarly, we remove code snippets, URLs and images from the issue body. Since there are no tags in GitHub issues that identify code snippets, we look for backquote ” ” or triple backticks “‘ in the content of the issues and remove the code enclosed between this punctuation. We also remove stop words, non-alphabetical characters, and punctuation. Finally, we apply Porter stemming to normalize the words into their base forms.

4.2.5 Topic modeling

We use automated topic modeling to analyze the topics in the Q&A posts and issue reports. Specifically, we use the Latent Dirichlet Allocation (LDA) algorithm [96] to extract the topics from both of our datasets. LDA is a probabilistic topic modeling technique that derives the probability distribution of frequently co-occurred word sets (i.e., topics) in a text corpus. A topic is represented by a probability distribution of a set of words, while a document is represented as a probability distribution of a set of topics. LDA is widely used for modeling topics in software repositories [97], including technical Q&A posts (e.g, [98]) and issue reports (e.g., [51]).

We use two separate topic models to extract the topics from the Q&A post data and the issue report data. For a better performance of the topic modeling and a good classification quality, following previous work [89] [99], we consider both uni-gram and bi-gram of words in our topic models.

LDA Implementation. We use the Python implementation of the Mallet topic modeling package [100] to perform our topic modeling. The Mallet package implements the Gibbs sampling LDA algorithm and uses efficient hyper-parameter optimization to improve the quality of the derived topics [100].

Determining topic modeling parameters.

The number of topics (K) is usually manually set by the user as it controls the granularity of the topics [89]. The α parameter controls the topic distribution in the documents (i.e., Q&A posts or issue reports), while the β parameter controls the word distribution in the topics. In this work, we use the topic coherence score [101] to evaluate the quality of the resulting topics and determine the appropriate parameters (K, α, and β), similar to prior work [35, 89]. The coherence score measures the quality of a topic by measuring the semantic similarity between the top words in the topic. Thus, this score distinguishes between topics that are semantically interpretable and topics that are coincidences of statistical inference [101]. Specifically, we use the Gensim Python package's `CoherenceModel` [102] module to calculate the coherence scores of the resulting topics. To capture a wide range of parameters and keep the topics distinct from each other, we experiment with different combination of the parameters, by varying the values of K from 5 to 30 incremented by 1 each time, the values of document-topic distribution α from 0.01 to 1 incremented by 0.01 [103], and the values of word-topic distribution β from 0.01 to 1 incremented by 0.01 [103]. We retain the resulting topics with the highest average coherence score.

After getting the automatically derived topics, we manually analyze the resulting topics and assign meaningful labels to the topics. We elaborate more on this process in RQ2 and RQ3 for the Q&A post topics and the issue report topics, respectively.

4.3 Experiment Results

In this section we report and discuss the results of our three research questions. For each research question, we first present the motivation and approach, then discuss the results for answering the research question.

RQ1: What types of QSE questions are asked on technical forums?

Motivation

In order to understand QSE challenges developers are facing, we first want to understand what types of questions they are asking (e.g., whether they are asking questions about using APIs or fixing errors). This is important to identify the areas in which QSE developers should be supported and the type of resources that they need. Similar to prior work [1], we focus on the intent behind the questions asked by QSE developers instead of the topics of the questions.

Approach

To identify the type of questions that users are asking in technical forums, we performed a manual analysis of a statistically representative sample from our studied QSE questions. We sampled 323 questions with a confidence level of 95% and a confidence interval of 5%. For each question, we examined its title and body, to understand the intent of the user who posted the question. We used a hybrid card sorting approach to perform the manual analysis and assign labels (i.e., types of questions) to each sampled question. Specifically, we based our manual analysis on an existing taxonomy of the types of questions asked on Stack Overflow [1] and added new types when needed. For each question we assigned one label; in case a question is associated with two or more labels, which we found only in a few cases, we chose the most relevant one.

Hybrid card sorting process. Two researchers (i.e., coders) jointly performed the hybrid card sorting. We split the sampled data into two equal subsets and performed the sorting in two rounds, similar to prior work [104]. Our process guaranteed that each question is labelled by both coders.

1. **First-round labeling.** Each coder labels a different half of the questions independently.

2. **First-round discussion.** In order to have a consistent labeling strategy, we had a meeting to discuss the labeling results in the first round and reached an agreed-upon set of labels. A third researcher of is involved in the discussion.

3. **Revising first-round labels.** Each coder updated the first round labeling results based on the discussion.

4. **Second-round labeling.** Each coder labeled the other half of the questions independently based on the agreed-upon labels in the first round. New labels are allowed in this round.

5. **Second-round discussion.** We had a meeting to discuss the second-round labeling results, validate newly added labels and verify the consistency of our labels. A third researcher is also involved in the discussion.

6. **Revising second-round labels.** Based on the second-round discussion, each coder revised the labels and finalized its individual labeling of the questions. We calculate

the inter-coder agreement after this step.

7. **Resolving disagreement.** We had a final meeting to resolve the disagreement in our labeling results and reached the final label for each question. For each difference in our labels, the two coders and a third researcher discussed the conflict and reached a consensus.

Inter-coder agreement. We measured the inter-coder agreement between the coders and obtained a Cohen's kappa k value of 0.73 which indicates a substantial agreement [105]. Therefore our manual labeling results are reliable.

Results

Table 4.2 shows the result of our qualitative analysis for identifying the categories of questions in technical forums. Among the 323 questions we analyzed, we could not assign a label to only one question. In the table, we provide the description of each category and how frequent it appears in our qualitative analysis.

All seven categories of Stack Overflow questions identified in prior work appear in QSE-related posts. Prior work [1] identified seven categories of questions on Stack Overflow by studying Android-related questions, including `API usage`, `Conceptual`, `Discrepancy`, `Errors`, `Review`, `API change`, and `Learning`, ordered by their occurrence frequency. Although quantum computing is still a new area, people start to ask all these different categories of questions, indicating that quantum computing face similar software engineering challenges (e.g., `API usage` and `API change`) as other software engineering domains. Similar to prior work, we find that `API usage` is the most frequent category with 26.3% instances. The questions of this category are usually identified by "how to"; e.g., "*How to return measurement probabilities from the QDK Full-state simulator?*"

The categories of `Errors` and `Learning` are relatively more frequent in QSE-related questions than in the prior taxonomy of question categories [1]. Compare to prior work [1] on classifying Android-related questions, we find that `Errors` and `Learning` questions are relatively more frequent. As quantum computing is still an emerging domain, people practicing it face many errors when developing quantum computing applications and they find it challenging to find learning resource for quantum computing. An example of the `Errors` category is "*I have Qiskit installed via Anaconda and a virtual environment set up in Python 3.8. ... I get an error. I'm not sure what the problem is. How do I fix it?*". Another example for the `Learning` category is "*How do I learn Q#? What languages should I know prior to learning Q#? How do I get started with quantum computing?*". These find-

Table 4.2 A taxonomy of Question Categories which bases on and extends [1]

Category	Description	Freq
API usage	Questions of this category are usually identified by "how to", i.e., how to use an API or how to implement a functionality.	85
Theoretical*	This category of questioners ask about theoretical explanations of quantum programs, algorithms, and concepts.	54
Errors	This category of questions search for explanations and solutions of errors and exceptions when developing or executing quantum programs.	49
Conceptual	Questions in this category are related to the limitation, background and the underlying concept of an API.	45
Discrepancy	Question of this category usually ask for explanations or solutions for unexpected results (e.g., "what is the problem", "why not work".	31
Learning	Questions in this category are searching for learning resources such as documentation, research papers, tutorials, or websites.	22
Review	This category describes questions like: "How/Why this is working?" or "Is there a better solution?". Generally, the questions in this category look for a better solution to a problem or for help reviewing the current solution.	17
Tooling*	This category describes questions like "I am looking for ...", "Is there a tool for ...". These questions search for tools to solve a specific problem or check the features of a tool.	16
API change	This category of questions concern about changes of an API and the associated compatibility issues and other implications.	2

*Categories newly identified in QSE-related questions.

ings suggest the need to develop tools or resources to help developers avoid or address such errors, as well as developing tutorials, books, and other learning resources to help beginners get acquainted with quantum computing.

Two new categories of questions (i.e., `Theoretical` and `Tooling`) emerge in QSE-related posts. In fact, the category of `Theoretical` is the second most frequent among all categories. This category is usually associated with keywords such as "can someone explain", "what is", and "does quantum". An example question of this category is "What is the analysis of the Bell Inequality protocol in Cirq's 'examples'?" where Cirq [10] is a Python library for developing quantum computing applications. This category of questions indicates that people have challenges understanding the theoretical concepts behind quantum computing code. Future efforts are needed to explain such theoretical concepts for developers. The category of `tooling` represents questions that are looking for tools, frameworks, or libraries that can help solve a QSE-related problem or verifying whether a tool, framework, or library can help solve a problem. For example, "*I want to use Blender and Blender Python Scripts working with Qiskit. How can I do this? How to make communication between Blender and Qiskit installed with Anaconda Python?*". This category indicates the lack of established tools for supporting quantum program development.

We identified nine categories of QSE-related questions in Stack Exchange forums. The categories `Theoretical`, `Errors`, `Learning`, and `Tooling` are new or become more frequent in QSE-related questions. Our results highlight the need for future efforts to support developers' quantum program development, in particular, to develop learning resources, to help developers fix errors, and to explain theory behind quantum computing code.

RQ2: What QSE topics are raised in technical forums?

Motivation

Developers post QSE-related questions and answers on technical forums. Their posts may reflect their faced challenges when learning or developing quantum programs. To understand their faced challenges, we use topic models to extract the semantic topics in their posts and analyze the characteristics of these topics.

Approach

Topic assignment and frequency.

The automated topic modeling generated nine topics and distribution of co-occurring words in each topic. We then manually assigned a meaningful label to each topic. Following prior work [89, 106, 107], to assign a meaningful label to a topic, the first researcher first proposed labels using two pieces of information: (1) the topic's top 20 keywords, and (2) the top 10-15 most relevant questions associated with the topic. Then, three researchers of the study reviewed the labels in meetings and reassigned the labels when needed. We obtained a meaningful label for each of the nine topics at the end. For each topic, we measure the percentage of the posts (i.e., frequency) that have it as the dominant topic (i.e., with the highest probability).

Topic popularity. To understand developers' attention towards each topic, following previous work [89, 106, 107], we measured three metrics for each topic: (1) the median number of views of the associated posts, (2) the median number of associated posts marked as `favorite`, and (3) the median score of the associated posts. For each topic, the associated posts refer to the posts that have it as the dominant topic.

Topic difficulty. In order to better understand the most challenging aspects for developers, we measure the difficulty of each topic in terms of how difficult it is for the associated posts to get accepted answers. Following prior work [89, 106, 107], for each topic, we measure two metrics: (1) the percentage of the associated questions with no accepted answer, and (2) the median time required by the associated questions to get an accepted answer (only considering the ones with an accepted answer). For each topic, the associated questions refer to the questions that have the topic as the dominant topic.

Results

Table 4.3 Topics extracted from QSE related posts on Stack Exchange forums

Topic (manual label)	Keywords	Description	% Freq
Environment management	quot, error, python, build, code	Development environment and build problems	15.03
Dependency management	qiskit, import, ibmq, operator, provider	Library installation, use, and versioning issues	14.82
Algorithm complexity	time, problem, algorithm, number, function	Quantum algorithm complexity and optimization	14.06
Quantum execution results	circuit, result, back-end, simulator, measure	Quantum program execution results on quantum backends (e.g., simulators)	13.22
Learning resources	question, paper, work, understand, answer	Searching for learning resources such as research papers and tutorials	9.05
Data structures and operations	matrix, return, array, datum, list	Data structures (e.g., matrix, arrays and list) and their operations in quantum programs	8.81
Quantum circuits	qubit, gate, control, operation, cirq	Elements of quantum circuits (e.g., Qubits, gates) and their operations	8.66
Quantum vs. classical computing	quantum, computer, classical, computing, algorithm	Comparisons between quantum and classical computing or migrating classic algorithms to quantum computing	8.30
Quantum algorithms understanding	state, rangle, frac, theta, sqrt	Quantum algorithm explanation and interpretation	7.51

We derived nine topics that are discussed in QSE-related posts, including traditional software engineering topics (e.g., `environment management` and `dependency management`) and QSE-specific topics (e.g., `quantum execution results` and `Quantum circuits`). Table 4.3 describes the nine topics and their frequency in the analyzed posts. As one can observe in QSE there is no wide range of topics discussed with a total of 9. Since the number of the detected topics in not big, We present a low level of granularity. Also for each topics we illustrate the percentage of the question asked order by their occurrence.The percentage indicate the dominance of a topic compared to others.

Table 4.4 shows the median views, scores, and favorites of the posts associated with these topics. The three most dominant topics are `environment management`, `dependency management`, and `algorithm complexity`.

`Environment management` is the most dominant topic representing 15.03% of posts. For example, the most viewed question of this topic is "*I downloaded the Quipper package but I have not been able to get haskell to recognize where all of the modules and files are and how to properly link everything*" which gained 2772 views. Other examples include "How can I run QCL (quantum programming language) on Windows?" and "Visualization of Quantum

Circuits when using IBM QISKit". We can observe that users are new to quantum computing and facing problem while setting up their environment and installing their tools. This topic is also linked to the question category `tooling` that we derived in RQ1.

`Dependency management` is the second most discussed topic representing 14.82% of the posts. For example, the most viewed question (with 2,239 views) of this topic is "*When trying the above code, I am receiving the following error: ModuleNotFoundError: No module named qiskit*" where `qiskit` is an open source framework for quantum program development [9]. We noticed that a large number of questions are directly related to `qiskit`. This can be explained by the lack of documentation or tutorials in using this framework.

`Algorithm complexity` is the third most dominant topic. This topic is about understanding the complexity of quantum algorithms and how to optimize quantum algorithms. For example, the most viewed question of this topic is "*For the other algorithms, I was able to find specific equations to calculate the number of instructions of the algorithm for a given input size (from which I could calculate the time required to calculate on a machine with a given speed). However, for Shor's algorithm, the most I can find is its complexity: O((log N)^3)*", which receives 4,718 views. This topic is linked to the questions category `theoretical` derived from RQ1. This topic indicates developers' challenge in understanding the complexity of quantum algorithms.

As quantum programming is oriented to searching solutions in a probabilistic space, which is counter-intuitive from the classical computing perspective, understanding `quantum execution results` is particularly challenging for developers. As a Qubit can be 0 or 1 with a certain probability, a quantum program that has Qubits as its basic units can have many different states at the same time. The results of a quantum program are certain only when the results are observed (or "measured"). Therefore, it is more challenging for developers to understand the results of quantum programs than that of classical programs. For example "How to plot histogram or Bloch sphere for multiple circuits? I have tried to plot a histogram for the multiple circuits for the code given below. I think I have done it correctly but don't know why it's not working. Please help me out. If possible please solve it for the Bloch sphere" Future efforts are needed to interpret quantum program outputs.

QSE has a distinct characteristic for each topic. For example, algorithm complexity is more focused on studying adopting and optimizing classical algorithms to quantum computing. The user's challenges evolve along with the topics associated. Therefore, in order to understand the topics evaluations, in figure 4.2 we show the trends of QSE topics between 2012 and 2020. For most topics, the trend starts gradually to increase. Quantum simulators started

Table 4.4 Popularity of QSE-related topics on Stack Exchange forums

Topic name	$\tilde{View}$	$\tilde{Score}$	$\tilde{Favorite}$
Quantum vs. classical computing	147.5	3	1.5
Quantum circuits	107.0	2	1.0
Environment management	106.0	1	1.0
Learning resources	102.0	2	1.0
Quantum execution results	98.0	1	1.0
Quantum algorithms understanding	97.5	2	1.0
Dependency management	93.0	2	1.0
Algorithm compolexity	87.5	1	1.0
Data structures and operations	82.0	1	1.0

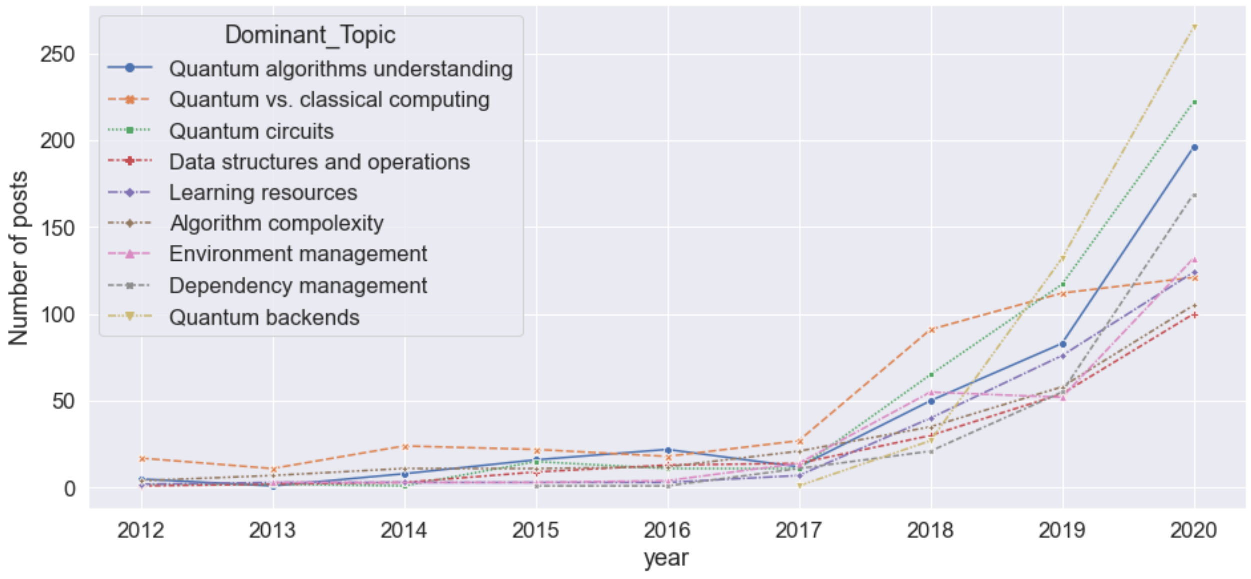

Figure 4.2 Q&A forums topics evolution overtime

to get interested in 2012 compared to the rest of the topic. In 2014 the all QSE topic's the number of question and answers increased while in 2016 we observe the exponential jump of QSE posts in the Q&A forums. The massive increase in the number of posts related to QSE indicates the growing interest in quantum computing by the software engineering community. Moreover as shown in figure 4.2 the trend is not showing any signs of decreasing in the future.

We further study the popularity in QSE topics in the technical forum posts. **The number of QSE-related posts started to grow exponentially after 2017. In particular, the topics of Quantum algorithm understanding, quantum execution results and Quantum circuits which emerged in 2017 becomes the most frequent topic recently.**

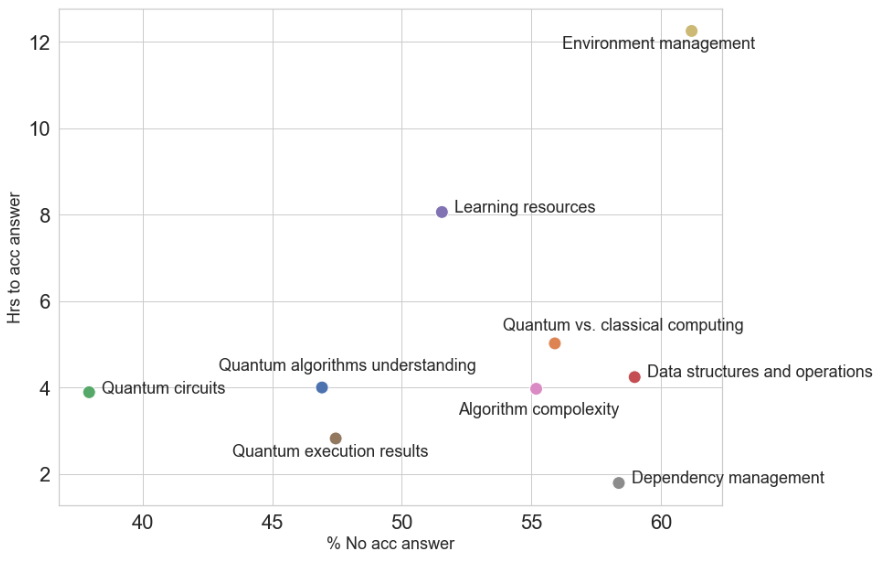

Figure 4.3 The difficulty aspect of QSE-related topics on Stack Exchange forums

Posts related to `quantum vs. classical computing` are gaining the most attention from developers. Since quantum computing is based on a new philosophy different from classical computing, developers often ask questions about the differences and look to understand the new doors quantum computing is opening. According to Table 4.4, posts with this topic receive the highest median number of views. This may show that software engineers are eager to contribute to QSE by starting from the differences between the two paradigms. However, as shown in Figure 4.3, posts on this topic are least likely to receive accepted answers. Our results indicate the need for resources and tools for bridging the knowledge gap between quantum computing and classical computing.

Questions of some topics (e.g., `environment management`) are much more difficult than others to receive accepted answers. According to Figure 4.3, the topic `environment management` is the most difficult topic to answer, with 61% of posts not receiving an accepted answer and a median time of 12 hours to receive an accepted answer. `Learning resources`, `quantum vs. classical computing` and `data structures and operations` are also among the most difficult topics in terms of the ratio of posts getting accepted answers and the time to get one. The results indicate the lack of community support in aspects such as setting up a development environment, searching for learning resources, and understanding differences between quantum computing and classical computing, which could impair the advancement of quantum software development practices.

From Q&A forums, we derived nine topics discussed in QSE-related posts, including traditional software engineering topics (e.g., `environment management`) and QSE-specific ones (e.g., `quantum execution results`). We highlighted some particularly challenging areas for QSE, such as interpreting quantum program outputs and bridging the knowledge gap between quantum computing and classical computing.

RQ3: What QSE topics are raised in the issue reports of quantum-computing projects?

Motivation

Issue reports of quantum computing projects record developers' concerns and discussions when adding features or resolving issues in these projects. The textual information in the issue reports may communicate developers' challenges when developing quantum computing applications. Therefore, we analyze the topics in the issue reports to understand the challenges developers are facing as well as the prevalence of these challenges. While the questions on technical forums can provide information about developers' general challenges, the issue reports may communicate developers' challenges for specific problems (i.e., issues).

Approach

Topic assignment and frequency. Our topic model on the issue report data generated 17 topics. We follow the same process as described in RQ2 to manually assign meaningful labels to the automated topics, based on the top words in the topics and the content of the associated issue reports. During the manual assignment process, we found that some topics are similar to each other even though such similarity is not detected by the probabilistic topic model. Therefore, we follow prior work [62, 108] and merged similar topics. We also discarded one topic as we could not derive a meaningful label from the top words and the associated issue reports. In the end, we obtained 13 meaningful topics. For each topic, we measure the percentage of the issue reports (i.e., frequency) that have it as the dominant topic (i.e., with the highest probability). We also measure the number and percentage of the studied projects that have at least one issue report of each topic.

Topic difficulty. To further understand developers' challenges in developing quantum computing applications, we measure the "difficulty" of the issue reports associated with each topic. As we cannot directly measure the "difficulty" of issue reports, we measure three indirect metrics for each topic: (1) the percentage of issue reports associated with the topic that

Table 4.5 QSE-related topics derived from issue reports of quantum computing projects on GitHub

Manual label	Keywords	Description	% Freq	# Projects
Learning resources	summary, remove, tutorial, link, documentation	Search for documentation, tutorials, websites, etc.	14.94	109 (90.08%)
Environment management	build, include, library, release, variable	Development environment and build problems	13.72	107 (88.43%)
API change	version, qiskit, code, issue, update	API update or deprecation issues	11.65	98 (80.99%)
Quantum circuits	gate, circuit, qubit, operation, control	Elements of quantum circuits (e.g., Qubits, gates) and their operations	8.30	70 (57.85%)
Quantum chemistry	input, calculation, basis, energy, pyscf	Issues with quantum chemistry libraries (e.g., PySCF)	6.72	76 (62.80%)
Quantum execution errors	error, artiq, follow, experiment, device	Errors in the execution of quantum programs	6.38	80 (66.12%)
Unit testing	test, check, fail, unit, script	Unit testing failures	6.32	87 (71.90%)
API usage	function, method, class, parameter, call	How to use an API	5.81	80 (66.11%)
Quantum execution results	state, number, result, time, measurement	Quantum program execution results (i.e., measured state)	5.76	89 (73.55%)
Data structures and operations	implement, operator, matrix, problem, array	Data structures (e.g., matrix, arrays and list) and their operations	5.73	88 (72.73%)
Machine learning	model, datum, dataset, layer, benchmark	Quantum computing application in machine learning	5.26	75 (61.9%)
Dependency management	file, python, import, package, install	Library installation, use and versioning issues	5.23	93 (76.86%)
Algorithm optimization	case, time, optimization, long, performance	Program performance and algorithm optimization	4.19	83 (68.6%)

Table 4.6 The difficulty aspect of QSE-related topics on GitHub issues

Topic name	$\widetilde{Hr\ to\ close}$	$\widetilde{\#\ comments}$
Data structures and operations	151.40	1
Quantum circuits	114.98	1
Quantum execution results	98.02	1
Machine learning	94.68	2
API usage	80.70	1
Quantum chemistry	62.89	2
Quantum execution errors	59.44	2
API change	47.34	1
Algorithm optimization	39.83	1
Dependency management	32.40	2
Unit testing	28.26	1
Learning resources	27.02	1
Environment management	21.57	1

All the issues are closed at the time we analyzed their status.

is `closed`, (2) the median time required to close an issue (since its creation) associated with the topic, and (3) the median number of comments in an associated topic (intuitively, an issue report with more comments may be more difficult [109]). For each topic, the associated issue reports refer to the issue reports that have it as the dominant topic.

Results

We derived 13 topics from GitHub issue reports, bringing new perspectives to the challenges faced by QSE developers. Table 4.5 shows the list of our derived topics, their descriptions, their percentage frequency in the studied issue reports, and the number of projects that have at least one issue report of the topic. Among the 13 topics, 6 of them (`learning resources`, `environment management`, `quantum circuits`, `quantum execution results`, `data structures and operations`, and `dependency management`) are overlapping with the topics derived from Stack Exchange posts (RQ2), and another 2 of them (`API change` and `API usage`) are overlapping with the categories of Stack Exchange questions derived in RQ1. This result indicates that the QSE-related challenges that we derived from forum posts indeed impact practical quantum program development in GitHub projects.

Among the other five topics, two of them (i.e., `machine learning` and `quantum chemistry`) are related to the most popular and promising quantum computing application areas: machine learning and chemistry. For example, an issue report associated with `quantum chemistry` raises an issue when using a molecular optimizer Python library: "*it leads to PyBerny optimizing an unconverged ground state energy, which generally leads to the geometry optimization never converging*". The other three topics (i.e., `quantum execution errors`, `unit testing`, `algorithm optimization`) are related to applications of traditional software engineering processes in quantum program development.

All derived topics are general among the quantum computing projects, as each topic is present in the issue reports of 58% to 90% of the projects. We observe that `learning resources` and `environment management` are the two most frequent topics and appear in 90% and 88% of all the studied projects, respectively, which once again highlights the need of efforts for developing learning resources and supporting developers in setting up their quantum program development environment.

Some topics are particularly challenging for developers, such as `data structures and operations`, `quantum circuits`, and `quantum execution results`. Table 4.6 shows the median time it takes to close an issue report and the number of comments in an issue report associated with each topic. All the issues are closed at the time when we analyzed their status. The issues associated with each topic only have a median of one to two comments, indicating that developers' interactions on these issue reports are not intense. `Data structures and operations` is also among the most difficult topics on forum posts (as discussed in RQ2). However, the `Quantum circuits` and `quantum execution results` topics are not among the most difficult topics on forum posts, while they are two of the most difficult ones on GitHub issues, which indicates that quantum circuit issues and the interpretation of quantum program execution results are more difficult in specific problem contexts.

QSE-related challenges that we derived from forum posts indeed impact practical quantum program development in GitHub projects, while GitHub issues bring new perspectives on developers' faced challenges (e.g., on specific quantum computing applications such as machine learning). In particular, we observe that the challenges are generally among the quantum computing projects.

4.4 Discussion and implication

In this chapter, we have presented the diverse issues reported not only on stack exchange forums (i.e., StackOverflow, Quantum Computing Stack Exchange) but also on GitHub, calling for attention to the most difficult and discussed topics. Finally, we highlighted the types of questions that quantum software engineers discuss in stack exchange forums. This section discusses the identified challenges and their implications.

Impactful topics: All the topics identified in this work are fundamental in their specific context. Quantum computing is a new computation approach explored by software engineers. This work summarizes the trendy and challenging quantum software engineering topics in GitHub and Stack exchange forums. We believe the pinpointed topics should be given further awareness by researchers and practitioners to assist the development of the quantum computing area. Figure 4.3 presents the relation between the percentage of not accepted answers (X-axis) and the hours to get an accepted answer (Y-axis). As can be observed, the topic `Environment management` appears to be the most difficult topic to answer with the least accepted answers and the longest time to receive one. This is an important path for the researcher to highlight the different aspects of the quantum environment and the challenges practitioners are facing. `Learning resources`, `quantum vs. classical computing` and `data structures and operations` need more attention as shown in Figure 4.3. GitHub issue report identifies three forward quantum computing topics `Data structure and operations`, `quantum circuits` and `quantum execution results` as the most difficult because of the long time they take to get a fix. This suggests the need for a general effort from researchers to help practitioners to investigate further the quantum computing challenges. We also pointed out that the categories `Theoretical`, `Errors`, `Learning`, and `Tooling` are new or become more frequent in QSE-related questions. This indicates the need for future efforts to support the community, in particular, to **develop learning resources**, to help developers **fix errors**, and to explain **theory behind quantum computing code**.

4.5 Threats to Validity

External validity. In this chapter, we analyze four Stack Exchange forums and 122 GitHub repositories to understand the challenges of QSE. Our studied forum posts and GitHub issues may not cover all the ones that are related to QSE. Developers may also communicate their discussions in other media (e.g., mailing lists). Future work considering other data sources may complement our study. In addition, we identify and collect the posts from Q&A forums using a selected set of tags. Our analysis may miss some QSE tags. However, to alleviate this threat, we follow prior work [88, 89] and use an iterative method to identify the relevant tags.

Internal validity. In this chapter, we use topic models to cluster the forum posts and GitHub issue reports, based on the intuition that the same clusters would have similar textual information. However, different clusters of posts and issue reports may exist when a different approach is used. To ensure the quality of the clusters, we manually reviewed the resulting topics, merged similar topics when needed, and assigned meaningful labels to them.

Construct validity. In RQ1, we manually analyze the categories of QSE-related questions on technical Q&A forums. Our results may be subjective and depend on the judgment of the researchers who conducted the manual analysis. To mitigate the bias, two researchers collectively conducted an manual analysis and reached a substantial agreement, indicating the reliability of the analysis results. A third researcher also participated in the discussions during the manual analysis, to ensure the quality of the results. In RQ2 and RQ3, the parameters of the topic models (e.g., the number of topics K) may impact our findings. To mitigate this threat, following previous work [88] [89], we did multiple experiments and use the topic coherence score to select the most suitable parameters. The manual labeling of topics can be subjective. To reduce this threat, the researchers read each topic's top 20 keywords and the top 15 highest contributed posts to the topic. We followed a clear-cut approach adapted in previous works [88] [89]. In addition, in our analysis of the QSE posts, we did not filter posts using the number of comments, votes or answers (as done in prior work [110]), which may lead to noise in the analyzed posts (e.g., low-quality posts). We made this decision since QSE is a new topic and the number of posts in the Q&A forums is relatively small.

4.6 Chapter summary

This chapter presented the results of a large-scale study performed using Stack exchange forums and GitHub issues reports to understand what quantum software engineers ask about and highlighted the topics that are the most challenging. We identified the popular and complex issues and inspected the correlations between the latter in GitHub and Stack exchange independently. Besides we have identified the nature of questions that quantum software engineers asked; grouping them into nine types (i.e., "API usage", "Conceptual", "Discrepancy", "Errors", "Review", "API change", "Learning", "Theoretical" and "Tooling"). Finally, we discussed our findings and presented a set of recommendations for researchers, practitioners, and tool providers.

CHAPTER 5 BUG CHARACTERISTICS IN QUANTUM SOFTWARE ECOSYSTEM

5.1 Introduction

Quantum computing has achieved a series of important milestones in recent years. For example, D-Wave claimed the first commercial quantum computer in 2011 [5]. Tech giants such as IBM, Amazon, Google, and Microsoft are racing to build their quantum computers. Quantum computers are expected to make revolutionary computation improvements over modern classical computers in certain areas, such as optimization, simulation, and machine learning [6, 7].

The rapid development of quantum computers has driven the development of quantum programming languages and quantum software [3], with many of them released as open source [8]. A variety of quantum programming frameworks and languages have been introduced, such as Qiskit [9], Cirq [10], and Q# [11]. IBM's Qiskit is a Python-based software development toolkit for developing quantum applications that can run on quantum simulators or real quantum computers (e.g., IBM Quantum Cloud).

Quantum software, by its nature, is drastically different from classical software. For example, a classical software system is executed sequentially and the status of the system is typically deterministic. However, a quantum software system is intrinsically parallel and can have multiple possible states at the same time [12]. In addition, as quantum computers are error-prone due to the instability of quantum mechanisms, the output of a quantum software system is often noisy [13]. Thus, the bugs of quantum software may possess characteristics that are very different from those in classical software.

In this chapter, we perform an empirical study on 125 open-source quantum software projects hosted on GitHub. These quantum software projects cover a variety of categories, such as quantum programming frameworks, quantum circuit simulators, or quantum algorithms. An analysis of the development activity of these selected projects show a level of development activities similar to that of classical projects hosted on GitHub. To understand the characteristics of bugs occurring in quantum software projects, we examined the following two research questions.

RQ1: *How buggy are quantum software projects and how do developers address them?* In this research question, we compare the distribution of bugs in quantum software projects and classical software projects, as well as developers' efforts in addressing these bugs.

We observe that quantum software projects are more buggy than comparable classical software projects. In addition, fixing quantum software bugs is more costly than fixing their classical counterparts. Our results indicate the need for efforts to help developers identify quantum bugs in the early development phase (e.g., through static analysis), to reduce bug reports occurrence and bug fixing efforts.

RQ2: *What are the characteristics of quantum software bugs?* We qualitatively studied a statistically representative sample of quantum software bugs to understand their characteristics. In particular, we analyzed the quantum software components (e.g., quantum measurement) where these bugs occurred and examined the nature of these bugs (e.g., performance bugs). We observed that both quantum computing-related bugs and classical bugs occur in quantum computing components. The gate operation component is the most buggy. We identified a total of 13 different types of bugs occurring in quantum components. The three most occurring types of bugs are `Program anomaly bugs`, `Configuration bugs`, and `Data type and structure bugs`. These bugs are often caused by the wrong logical organization of the quantum circuit, state preparation, gate operation, measurement, and state probability expectation computation.

Our work is important to guide future works that aim to develop methodology and tooling to support the identification and diagnosis of quantum software bugs. Some of our bug results emphasize a quantum-specific approach to identifying bugs not detected by traditional techniques. Our findings can be beneficial to quantum computing developers. They can learn from the most occurring bug types and avoid them in future work. Our results on which component is more buggy can help guide developers' testing efforts. As quantum computing frameworks are still in their early days and not yet widely used, our study of bug types can have a beneficial impact on the future releases of the quantum frameworks.

Because quantum computing is still in its early stages, there have been very few empirical studies on quantum computing bugs. In 2021, Campos and Souto [37] highlighted the need to study quantum computing bugs. In the same year, Zhao et al [38] studied 36 bugs from a single quantum computing framework (Qiskit). In 2022, Matteo and Michael [39] identified the patterns of 223 real-world bugs in 18 quantum software projects. In this chapter, we study a larger set of bugs from a larger number of projects and perform a deeper analysis of the characteristics and types of quantum bugs.

In essence, this work makes the following contributions:

- A preliminary study of the status and a categorization of quantum software projects.

- A quantitative study of the bugs of 125 quantum software projects with comparison to bugs of classical software projects.
- An in-depth qualitative study of 333 real-world bugs in 125 quantum software projects.
- An investigation of the types of quantum bugs, and their distribution across the quantum components.
- Insights about the challenges and the complexity to fix the bugs.

Chapter organization. The rest of the chapter is organized as follows. In Section 5.2, we introduce the experimental setup of our study. A preliminary study on the characteristics of quantum software projects is presented in Section 5.3. In Section 5.4, we present the answers to our research questions. Section 5.5 discuss the finding of the study, while Section 5.6 review threats to the validity of our findings. Finally, Section 5.7 concludes and summarize the chapter.

5.2 Study Definition and Data Extraction Methodology

The *goal* of this study is to understand the characteristics of bugs occurring in quantum software projects. The *perspective* is that of researchers and tool builders interested in developing methodologies and tools to support the identification and diagnosis of quantum software bugs. The *context* of the study is 125 open-source quantum software projects hosted on GitHub. To achieve the study's goal, we proceed in three steps. First, in a preliminary study, we examine the nature of quantum projects, comparing them to classical projects. This preliminary study is important to identify the specific characteristics of quantum projects that could affect the bugs. Next, we conduct a quantitative study, comparing the distribution of bugs in quantum software projects and classical software projects, as well as developers' efforts in addressing these bugs. Finally, we qualitatively studied a statistically representative sample of quantum software bugs and build a taxonomy of quantum software bugs. Figure 5.1 depicts the methodology of our study. First, we collect quantum and classical projects from GitHub. Then, we collect the pull requests and issues of these projects using GitHub issues API. Next, we apply a set of heuristics to identify quantum software project bugs and classical project bugs. Finally, we perform open coding and apply statistical analysis to answer our research questions. The following sections elaborate in detail on each of these steps.

5.2.1 Quantum software projects collection

We followed two steps to collect quantum software projects.

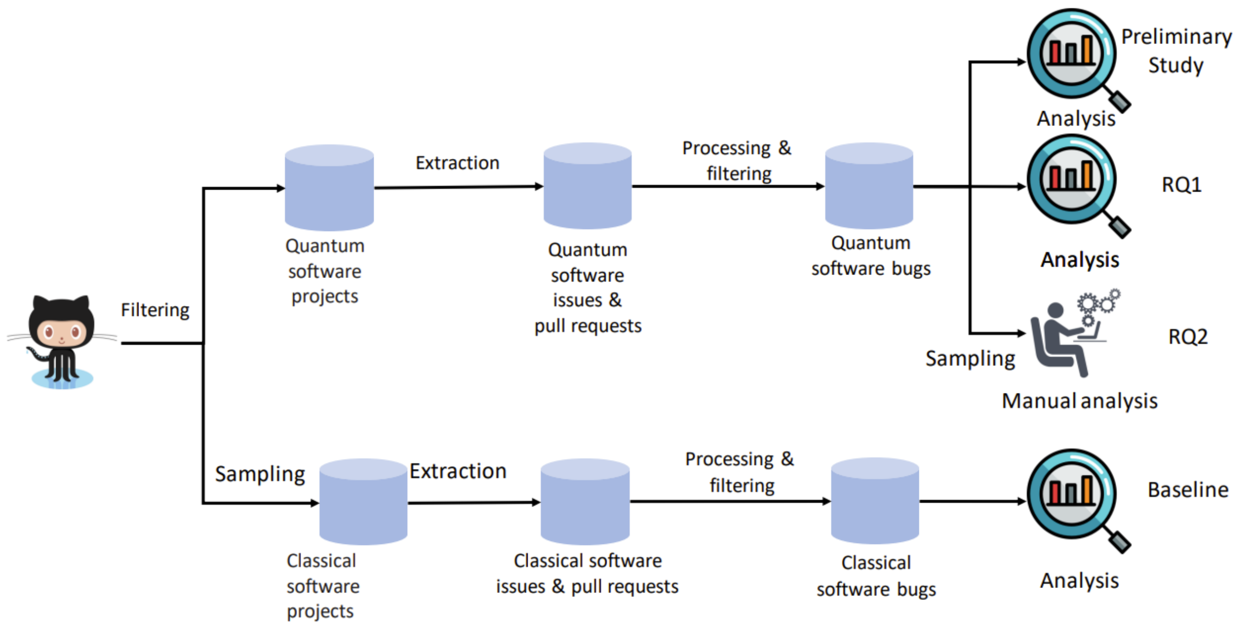

Figure 5.1 Schematic diagram of chapter 5 empirical study

Step 1: Searching candidate projects. To select representative projects from the quantum ecosystem, we searched through the GitHub `search API`. We used two ways to search for quantum software projects.

- First, we conduct a `keyword search` using the GitHub search API. To identify the quantum software projects in GitHub, we reduce our search scope and select the projects with a description or a topic that contains the word "quantum computing" (the word quantum computing is case sensitive to not miss relevant projects).
- Next, we `search for code` that imported quantum programming libraries. Our code search items Q_{code} contain import statements from popular quantum computing libraries such as `Qiskit` and `Cirq` developed by pioneers like Google and IBM. Using the `search code` feature, we search for Q_{code} in the quantum projects source code to identify the quantum software projects. Our searched items Q_{code} include: `import qiskit`, `import cirq`, `from cirq import`, `from qiskit import`, `import tensor flow_quantum`, `from tensorflow_quantum import`, `from braket.circuits import`, `import braket.circuits`.

In addition, we limit our search results based on the following criteria:

- To allow us to better understand the issues and content of the projects, the description of the project must be in English.
- Since we are looking to characterize the bugs in the quantum software projects, the projects must have issues.

At the end of this step, we obtained $Q_{P-initial}$ containing a total of 2,105 unique quantum project.

Step 2: Filtering collected projects. We filter $Q_{P-initial}$, the collected set of quantum software projects following guidelines proposed by Kalliamvakou et al [111]. Specifically, we selected projects based on their `number of commits in 2021`, their `total number of commits`, and their `number of contributors`. These 3 criteria ensure that we identify the most active projects, remove the abandoned projects, and filter out the quantum computing projects related to documentation, lecture notes, and student assignments [111].

1. We keep projects with at least 51 commits to ensure that they have sufficient development activity and to avoid student assignment [91] [92]. Figure 5.3 show the distribution of the `total number of commits` in $Q_{P-initial}$. Since the third quantile is 51, we used it as our threshold to filter the set of projects $Q_{P-initial}$.
2. We select the projects with at least 17 commits in the past year (i.e., 2021) to remove the abandoned and inactive projects. Figure 5.2 shows the distribution of the number of commits in the past years in $Q_{P-initial}$. We observe that 75% of the projects have less than 17 commits in the past year, therefore we use this value as a threshold to filter the set of projects $Q_{P-initial}$.

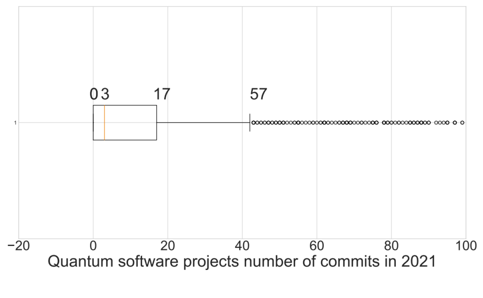

Figure 5.2 Distribution of the number of commits in quantum software projects in the past year

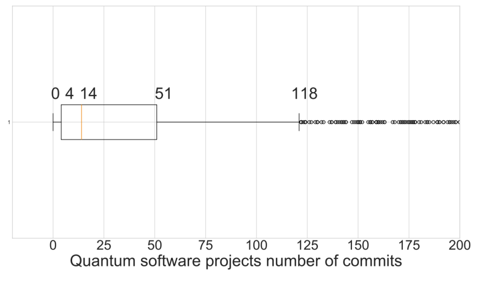

Figure 5.3 Distribution of the number of commits in quantum software projects

3. We retain projects with more than one contributor to avoid selecting toy projects.

After applying these filtering criteria, we manually inspected the remaining projects' descriptions and removed non-quantum projects (e.g,, projects related to quantum physics, documentation, or lecture notes).

Finally, we ended up with a total of 125 projects ($Q_{P-final}$) that are directly related to quantum computing.

5.2.2 Classical projects collection

To understand the quantum software projects' characteristics, we need a baseline for our analysis. Hence, we collected a set of classical software projects comparable to the quantum software projects. From GitHub, we selected classical projects that have the range of `number of stars`, `programming language`, `number of watcher`, similar to our quantum repositories and the repository must have `issues`. As the number of projects in the extracted list of classical software projects is very large, to have a representative baseline, we collected a random sample of 324 projects using a 95% confidence interval level and a 5% margin of error.

5.2.3 GitHub issues and pull requests collection

To identify the bugs and to uncover the cost to fix a bug in the quantum software projects, we collected the pull requests and issue reports for each of the studied projects, using GitHub issues API which considers every pull request as an issue. We collected the following fields: `issue number, state, title, repository url, issue url, pull request url, body, creation date, close date, and merge date`. Using GitHub issue API [93] for each project, we extracted $B_{initial}$ a total of 59,249 issues and pull requests from $Q_{P-final}$. In the rest of the chapter, we refer to the issue reports and the pull requests as issue reports or issues.

5.2.4 Identification of quantum software projects bugs

To identify the bugs in GitHub issue reports, we first map each issue with its corresponding commit. For each project, we collect the related commits of each issue report using the GitHub API. To identify the commit or list of commits of an issue report, we look into the list of events of the issue report. If an issue is referenced in a commit, the `commit url` will appear in the list of events with the tag `referenced`. Therefore we collect the list of `commit url` that appears in the issues event. We collect the commits of the issues report for each project using the `GitHub commit API` and collect the following fields of each commit: `the commit message`, the `URL`, the `number of added`, `deleted` or `changed` files, and the `contributor identification`. The next step is to classify if an issue in $B_{initial}$ describes a bug or not. For this purpose, we proceed as follows:

1. In GitHub, developers tag the issues using the `Label` field. To identify bugs, we select the issues that has the tag `Label=bug or defect` and flag the issues as a `bug`.
2. Following previous work [112], we define a set of keywords $K_{initial}$ **fix, error, crash, wrong, bug, issue, fail, correct** to detect bugs.
3. For each issue report b $\in B_{initial}$, if b's title or body or commit message contains at least a word k $\in K_{initial}$, b is flagged as a bug. We ended up with a set B_1 of 18,084 bug reports at the end of this step .
4. To extend the set $K_{initial}$ and identify further potential bugs $\in B_{initial}$, we select a random sample with 95% confidence interval level and 5% marge of error of the issues not flagged as bugs, i.e., $\in \overline{B_1}$.
5. One researcher manually inspected the commits of the random sample of issue reports and defined an extended set of $K_{initial}$. Then, during a meeting, three researchers discussed the validity of the new set of keywords. Finally, all researchers agreed on the final set K_{final} **fix, error, crash, wrong, bug, issue, fail, correct, exception, log, inf, insufficientResource, broke, resolve, abort, leak.**
6. For each issue reports b $\in \overline{B_1}$, if b's title, body or commit message contains at least a word k $\in K_{final}$, b is flagged as a bug. We obtained a set B_{final} containing 19,564 bug reports.

As a result of the process, we ended up with a total of 19,564 bug reports B_{final} that are related to quantum computing.

5.2.5 Classical projects bug extraction

We follow the same processes as described in Section 5.2.3 and Section 5.2.4 to extract the bugs of the selected classical software projects. In the 324 classical projects, we had a total of 60,779 issue reports in which we identified 13,165 bugs.

5.3 Preliminary study: Characteristics of quantum software projects

In order the understand quantum programming challenges, we first want to understand the nature of the quantum software projects. In particular, we want to understand the types of the projects and compare their characteristics (e.g., developer activities) with classical software projects.

5.3.1 Approach

Categorization of quantum software projects. To identify the categories of quantum projects in GitHub, we manually analyzed the descriptions and the documentation of all the projects. No prior taxonomy was used. For each project, we assigned one label. In case a project is associated with more than one categories, which occurred in a few cases, we chose the most representative one. Two researchers (i.e, coders) of our laboratory jointly performed the manual labeling. Each project is labeled by both researchers. We describe our labeling process below.

1. **Initial labeling.** Each coder labels all the projects independently.
2. **Discussion.** To have a consistent labeling strategy, we scheduled a meeting after the initial labeling and reached a consensus on the set of labels. A third researcher of our laboratory is involved in the meeting.
3. **Revising labels.** Each researcher updated their labeling results after the meeting.
4. **Resolving disagreement.** We had a final meeting to resolve the disagreement in the labeling results and reached a consensus for the label of each project. For each mismatched label, the three researchers discussed and resolved the conflict.

Analyzing the characteristics of quantum software projects. We study the characteristics of the quantum software projects as follows:

1. In order to understand the maturity of quantum software projects, we look into the `duration` (i.e., from the creation date of the project until the date of our data extraction), and to study the the activity in the quantum projects, we look into the `number of releases`, `number of commits`, and `number of contributors`. We used GitHub API [93] to collect these metrics for each studied project.
2. In order to understand the profile of the developers of quantum software projects, for each project we retrieve the user name of the contributors, then for each contributor,

with the help of the GitHub events API [93], we look into its created events since the contributor joined GitHub. Following GitHub documentation [93], we select the events that describe the activity of the developers. Specifically, we focus on the following events: `CommitCommentEvent`, `IssueEvent`, `IssueCommentEvent`, `PullRequestEvent`, and `PushEvent`.

Steps 1 and 2 are repeated on the sample of the classical software projects. The results of the classical software projects are used as our baseline.

5.3.2 Results

Quantum software projects have been steadily increasing in recent years. Figure 5.4 shows the number of quantum software projects over time. We observe a significant increase in the number of projects after 2016. The increase may be explained by the rapid development of quantum computing technologies in recent years. In 2017, IBM announced a working quantum computer with 50 qubits, that can maintain its quantum state for 60 microseconds [113]. Microsoft released its Q sharp programming language [11] in the same year. In 2018, IonQ released its first commercial trapped-ion quantum computer [114]. In 2019, IBM released its first commercial quantum computer, the IBM Q System One [115].

We derived 9 categories of projects in the quantum computing software ecosystem covering quantum programming frameworks, simulators, tools, implementations of quantum algorithms, and applications such as machine learning. Table 5.1 shows the characteristics of these quantum software projects by category. Below, we describe each category of projects.

`Quantum circuit simulator`: Libraries that are used to simulate quantum computation on a classical computer.

`Quantum programming framework`: A set of development kits, languages and libraries such as Q# [11], Qiskit [9] and pyEPR [1] for quantum programming and quantum circuits design.

`Quantum algorithms`: Projects that provide one or more implemented quantum algorithms (i.e., grove[2]).

`Quantum machine learning`: A set of libraries for implementing hybrid quantum-classical machine learning models. For example, TensorFlow Quantum (TFQ)[3] is a quantum machine

[1]`https://pyepr\protect\discretionary{\char\hyphenchar\font}{}{}docs.readthedocs.io/en/latest/`

[2]`https://github.com/Seeed\protect\discretionary{\char\hyphenchar\font}{}{}Studio/grove.py`

[3]`https://www.tensorflow.org/quantum`

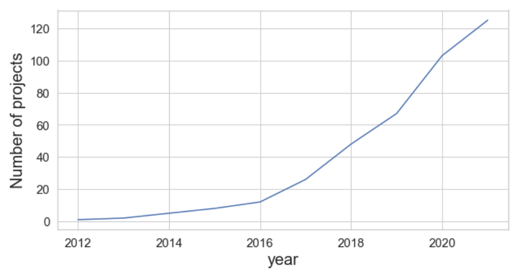

Figure 5.4 Cumulative distribution function of the quantum software ecosystem projects over time (days)

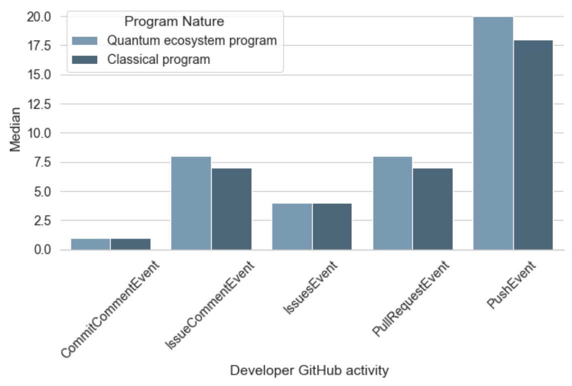

Figure 5.5 Comparison of the GitHub activities of the developers of classical and quantum projects

learning library for prototyping hybrid quantum-classical ML models.

`Quantum compilers`: Tools used for the synthesis, compilation, and optimization of quantum circuits (i.e., QGL2 Compiler[4]).

`Quantum utility tools`: Libraries and tools used to support quantum program development such as monitoring the load in quantum computers, API interface used to connect with quantum devices, or libraries dedicated for specific task development in the quantum programming workflow (i.e, state preparation, circuit visualization).

`Experimental quantum computing`: Libraries used for research and experimental calculation (i.e., artiq Compiler[5]).

`Quantum games`: Games developed using quantum programming that run on simulators or quantum computers.

`Quantum based-simulation`: Libraries used for simulating quantum physics experiments on a quantum computer.

Table 5.1 shows different metrics about the 125 studied projects and their distribution, along with the defined quantum ecosystem categories; including the number of releases, number of contributors, number of commits, number of lines of code (LOC), and number of programming languages used in the last release of the project.

As shown in Table 5.1, the largest number of projects fall into the `quantum circuit simulators` category with 26 projects. As quantum computers are still not conveniently accessible to

[4]`https://github.com/BBN\protect\discretionary{\char\hyphenchar\font}{}{}Q/pyqgl2`
[5]`https://m-labs.hk/experiment-control/artiq/`

Table 5.1 Characteristics of the studied quantum projects.

Program Category	nbr of projects	# of releases	# of commits	# of contributors	LOC	# of languages
Quantum circuit simulators	26	6.5	113.0	5.0	19,145	9.0
Quantum programming framework	22	10.0	163.5	24.0	49,981	12.0
Quantum algorithms	19	7.0	129.0	9.0	26,530	9.0
Quantum utility tools	19	7.0	126.0	6.0	27,830	9.0
Quantum compilers	9	3.0	179.0	10.0	175,364	16.0
Quantum machine learning	8	2.0	217.0	9.0	33,705	10.0
Quantum circuit simulator	7	19.0	29.0	15.0	87,316	13.0
Experimental quantum computing	7	10.0	147.0	15.0	66,088	13.0
Quantum-based simulation	2	16.5	459.0	84.5	92,819	14.5
Quantum fun	1	15.0	88.0	3.0	3,938	8.0
Quantum projects	125	8.0	131.0	8.0	34,188.0	10
Classical projects	324	12.0	268.0	2.0	15,357.0	7

# all values are median values in the projects in that category

the public, simulators are widely used to execute quantum programs. Moreover, `quantum programming frameworks` is the second most frequent project category with 22 projects. As quantum computing is getting more popular, programming frameworks are starting to emerge. `Quantum algorithms` and `quantum utility tools` are the third most occurring categories with 19 projects for each. **`Quantum programming frameworks`** and **`utility tools`** are important to unlock the full potential of quantum computing systems. In Table 5.1, we observe that quantum software projects have a lower level of maturity (in terms of development activities) with a respective median number of commits and releases of 131 and 8, in comparison to classical projects which have a 268 median number of commits and 12 median number of releases.

Quantum software project developers show a similar level of activities as that of classical software developers. In order to understand the activities of quantum project developers, in Figure 5.5 we compare the activities of quantum software projects and classical projects contributors by studying the developer's activities on GitHub since their registration in GitHub. Note that the quantum software projects and classical software projects are selected using the same criterion. We select five dimensions of activities from the GitHub `event API` : `commit on the commits`, `issues comments`, `created issues`, `created pull requests`, `push event` (i.e., commit and pull request). We chose these five dimensions because they describe how active a developer is in GitHub since joining. We observe some similarities between classical program developers and quantum program developers across the five dimensions. For example, quantum software developers and classical software developers have the same number of `issue event` with a median value of 3. However, quantum program developers are more active in other GitHub activities. From Figure 5.5, we observe that quantum developers have a slightly higher number of `issues comments event`, `pull request event`, and `push event`. These results show that developers of quantum software projects put more effort and face more issues. This observation motivated our investigation of the bug-proneness of quantum software projects. We also investigate how developers address bugs in quantum programs, as well as the types and location of bugs in these programs.

Quantum software projects become increasingly popular on GitHub. We observe a diverse range of quantum software projects, including quantum programming frameworks, tools, algorithms, and applications. We also observe that developers of quantum software projects are more active and face more issues in quantum programs.

5.4 Characteristics of Bugs in Quantum Programs

In this section, we report and discuss the results of our two research questions. For each research question, we present the motivation, the analysis approach, and the obtained results.

5.4.1 How buggy are quantum software projects and how do developers address them?

Motivation

In order to understand the characteristics of quantum software bugs, we first want to understand the extent to which quantum software projects are buggy, as well as how effectively developers address these bugs and their efforts in such activities.

Approach

Following the process described in Section 4.2, we identified the bugs in the selected quantum software projects and classical software projects. To understand how developers address them, we further identify the fixed bugs, as well as the fixing time and the associated efforts.

Identifying fixed bugs. A bug report is a GitHub issue report describing a bug. In GitHub, not all closed bug reports are solved. In some cases, a bug report can be closed while the problem persists. Thus, we had to identify the fixed and non-fixed bugs. We followed 3 steps to classify the bug reports.

1. **Filtering the open issue reports.** Using the state field in the issues reports, we kept only the closed issues.

2. **Identifying fixed bugs.** Following previous work [116] [112], we looked for the set of keywords F_{fix} in the GitHub issue report message and its corresponding commits: **`closes, close, closed, resolve, resolves, resolved, fix, fixes, fixed`**. To identify the fixed bugs, for each closed GitHub issue report, in the commit message, we look if it contains f $\in F_{fix}$. If the commit message contains at least one of the keywords we flag it as `Fixed`.

3. **Manual verification.** One researcher inspected a sample of 382 fixed bugs with a 95% confidence level and 5% marge of error. We found that our identification of fixed/non-fixed bugs has a precision of 1 and a recall equal to 0.97.

Bug fixing rate. For each quantum or classical project, we calculate the bug fixing rate by dividing the number of fixed bugs by the total number of closed bugs (including fixed bugs and bugs closed without a fix) in that project.

Bug fixing duration. For each fixed bug, we measure the bug fixing duration by calculating the difference between the bug fixing time and the bug creation time.

Bug fixing effort. To measure the effort involved in a bug fix, we collected the code changes of the commits that contributed to the bug fix. A change in a commit is an `added` or `deleted` code line. We also collected information about the number of changed files in the commits, which can be an `added`, `modified`, `renamed` or `deleted` file. For each bug fix, we measured two metrics:

- **Number of changed files**: the `added`, `modified`, `renamed`, and `deleted` files in the commits that contribute to a bug fix.

- **Number of changed lines of code**: The number of `added` and `deleted` code lines in the commits that contribute to a bug fix.

Results

Quantum software projects show more bugs (with a median number of 28 bugs) than classical software projects (with a median number of 13 bugs). Table 5.2 provides a summary of the bugs in quantum software projects and the distribution of the bugs along with the project categories. A median of 28.0 bugs per project was detected in the studied quantum software projects, which is two times higher than the median number of bugs in the classical software projects. In light of these values, quantum programs are more buggy than classical programs. The higher number of bugs in a quantum project may be explained by the fact that quantum computers are error-prone and the output of quantum software is often noisy.

In particular, the quantum projects (e.g., projects in the categories of *quantum-based simulation, experimental quantum computing, and quantum machine learning*) are the most buggy projects. This observation may be explained by the fact that these quantum projects categories rely more on quantum computing theories. Since there are no prior collection of mathematical algorithms for quantum theories and convenience functions that developers can leverage, the developers of quantum programs may be prioritizing the correct implementation of these complex theories over the quality of their code.

The median value of the bug fix ratio in Table 5.2 shows that 27.5% of the quantum software bugs are closed but not fixed, compared to 21.1% of classical software bugs which are closed without a fix. One researcher manually inspected a statistically representative sample of 382 closed-but-not-fixed bugs, representing a 95% confidence level and 5% confidence interval, to understand why these bugs are closed without a fix. We noticed that in some cases developers don't follow the best practices of GitHub and mention the issue number in the commit message with the fixing keywords, which makes detecting the bug fixes from commit messages difficult. As an example, we present issue number 5148 in Qiskit[6]. Besides, in some issue reports, enhancements were labeled as bugs for future releases and closed by developers without notice in commit messages. For example, the issue id 20 in the qutip[7]. Finally, we detected bugs that were closed because developers could not provide a fix, or it is difficult to reproduce the bug. For example the issue number 402 in `Cirq`[8].

Developers of quantum software projects are actively addressing the bugs in their projects. Figure 5.6 show a comparison of the empirical distribution function of the duration to fix a bug in days. The figure shows that bugs in quantum software projects are fixed at a similar speed as in classical software projects, with 50% of bugs fixed in less than 1 day and 90% of bugs fixed within 48 days, indicating that developers of quantum software projects are actively maintaining their projects.

`Quantum machine learning`, `Quantum based-simulation`, and `Experimental quantum`

[6]`https://github.com/Qiskit/qiskit-terra/issues/5148`

[7]`https://github.com/qutip/qutip/issues/20`

[8]`https://github.com/quantumlib/Cirq/issues/402`

Table 5.2 Distributions of bugs and bug fix ratio in quantum software projects

Project category	# of bugs	# Bug fix ratio	Hours to fix a bug	# file changed in a commit	# LOC changed in a commit
Quantum-based simulation	406.0	0.500	33.94	2.00	63.00
Experimental quantum computing	168.5	0.628	27.98	2.00	12.00
Quantum Machine learning	104.0	0.736	66.05	2.00	38.0
Quantum programming framework	88.0	0.513	34.60	2.00	30.00
Quantum algorithms	72.0	0.657	25.01	1.50	15.00
Quantum compilers	36.0	0.572	20.72	2.00	28.00
Quantum circuit simulators	26.0	0.523	20.61	2.00	20.00
Quantum utility tools	21.0	0.689	15.40	2.00	11.00
Quantum projects	28.0	0.725	23.5	2.00	19.00
Classical projects	13.0	0.789	21.1	2.00	14.00

# all values are median values in the projects in that category

`computing` are more buggy than other categories. Table 5.2 shows the median number of bugs across quantum software projects categories and the cost to fix them. We observe that `quantum based-simulation` projects have the highest median fixed bugs with the value of 406.0, however, we also observe a low median bug fix ratio compared to other categories even though this category has a high median number of contributors as observed in Section 5.3. `Quantum machine learning, quantum algorithms and application, and Experimental quantum computing` categories have the highest bug fixing rate with respectively a median bug fix proportion (ratio) of 0.736, 0.657, and 0.628, meaning that those two categories are the least difficult for developers to fix. The highest number of bugs is for `quantum-based simulation` projects. There are almost two times more bugs in this category of quantum programs in comparison to the other projects, indicating that these projects need more support from the community for testing and debugging. In the `quantum-based simulation` category, programs are trying to simulate quantum mechanics and physics in a quantum computer, for this purpose developers have to implement mathematical equations, which may be complex for developers; inducing more bugs.

As quantum programming is still low-level, bug fixing is costly in terms of code changes compared to classical programming. Fixing a bug in a quantum program requires larger code changes than fixing bugs in classical programs. Table 5.2 shows that quantum programs have a median code line change of 19, while classical programs have a median code line change of 14.00. We attribute this situation to the low-level programming language used in quantum programming. Quantum software engineers write their code at the gate level which is similar to the logic gate for classical software. This produces more coupling between the elements in the code which can lead to more changes. Since quantum programming is probabilistic in nature, classic assertions and testing techniques can't be used; resulting in a lack of support for testing and debugging.

Bugs in `Quantum machine learning`, `Quantum programming framework`, `Quantum-based simulation` are the most challenging to fix. Duration is the time difference between when an issue report was created and closed. In Table 5.2, we observe that the projects from `Quantum machine learning, Quantum programming frameworks, and Quantum-based simulation` categories appear to take the longest time to fix with the respective median number of hours to fix the bug of 66.05, 34,90, and 33,94. Also, the category `Quantum based-simulation` has the highest median number of lines of code changed in a commit with 63.00 median code line changed. Moreover, `Quantum machine learning` and `Quantum programming frameworks` show a high median number of lines of code changed to fix a bug.

The quantum software projects are more buggy than classical software projects. While developers of quantum software projects are actively addressing their bugs, fixing quantum software bugs is more costly than fixing classical software bugs. Our results indicate the need for efforts to help developers identify quantum bugs in the early development phase (e.g., through static analysis), to reduce bug reports and the cost of bug fixing. Projects in the `quantum machine learning`, `quantum programming framework`, and `quantum-based simulation` categories are the most buggy and have the most difficult bugs to fix.

5.4.2 What are the characteristics of quantum software bugs?

Motivation

Quantum programs' execution flow is different from that of classical programs. To understand the challenges in the quantum computing ecosystem, it is important to understand how bugs are distributed among the different components of the programs.

Approach

To identify the different types of bugs occurring in quantum programs and the principal components of the quantum program execution flow where the bugs occurred, we performed a manual analysis of a statistically representative sample of bugs detected in RQ1. We

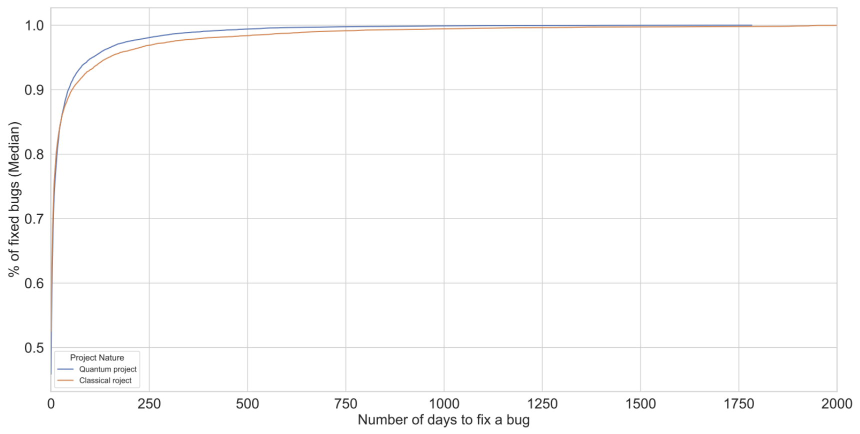

Figure 5.6 Cumulative distribution function of duration to fix a bug through time (days).

performed a stratified sampling over the quantum program categories, to ensure covering a wide range of bugs. Specifically, 1) we compute the sample size from the number of fixed bugs with a 95% confidence level and 5% interval which result in a sample size of 376. 2) We calculate the log number of issues per category. 3) We compute the project category weight: the normalized log number of bugs of the category divided by the log of the total number of fixed bugs. 4) Finally, we compute the sample size from each project category: the total sample size (376) multiplied by the corresponding weight.

After creating the sample, for each bug report, we examine the title, body, and comments to understand the bug reported by the user. We used the hybrid-card sorting approach to perform the manual analysis and assign labels (i.e., type of bug and quantum program component) to each sampled bug. To assign the bug type, we based our manual analysis on an existing taxonomy of the type of bugs reported on GitHub [116] and added new types when needed. For each bug report, we assigned one label, in case a bug is associated with two or more labels, which we found only in a few cases, we choose the most relevant one.

Three researchers (i,e coders) of our laboratory performed the labeling through two rounds as follows:

1. **First-round labeling.** One coder labels the bugs independently.
2. **First-round discussion.** In order to have a consistent labeling strategy, the first three researchers had a meeting to discuss the labeling results in the first round and reached an agreed-upon set of labels.

3. **Revising first-round labels.** The coder updated the first round labeling results based on the meeting.

4. **Second-round discussion.** The first three researchers had another meeting to discuss the revised labels, validate the updated labels and verify the consistency of the labels.

5. **Revising second-round labels.** Based on the second-round discussion, the coder revised the labels.

Results

We identified bugs in 12 quantum computing components. Figure 5.7 shows the 12 quantum computing computing components where bugs were identified which are mapped to a quantum-classical hybrid computing structure. Prior work [117] identified five components : `pre-processing`, `post-pro-`
`cessing`, `gate operation`, `state preparation`, and `measurement`. In Figure 5.7, we extended the component set to finally have 12 components. In Table 5.4, we show the distribution of the bug type across the quantum components. In the following, we discuss each quantum component in more details.

- **`Compiler (CP)`** translates the quantum program circuit into device-level language. From Table 5.3, we observe that compiler component issues are present in all the quantum ecosystem software categories with a total of 68 bugs. From table 5.4, we observe that the four most occurring bug types are program anomaly (23 bugs), test-code related (12 bugs), data type and structures (11 bugs), and configuration (11 bugs).

- **`Gate operation (GO)`** is a set of reversible operations that alter the state of the qubit and generate superposition. These reversible gates can be represented as a matrix. Gate operations are based on complex mathematical operations using imaginary numbers. Translating the maths into code is challenging for developers which increases the chances of errors. Table 5.3 shows that `gate operations` component is the second most buggy with 56 bugs detected in our sample. Table 5.4 illustrate that in this component, we have the two occurring bug types which are `program anomaly` and `data-type and data structure`. They respectively occur 27 and 7 times.

- **`Simulator (SM)`** simulates the execution of a circuit on a quantum computer. Since quantum computers are still not conveniently accessible by the public, simulators are widely used by quantum developers. We identify 10 bugs spread across the `quantum programming framework`, `experimental quantum computing`, `quantum circuit`

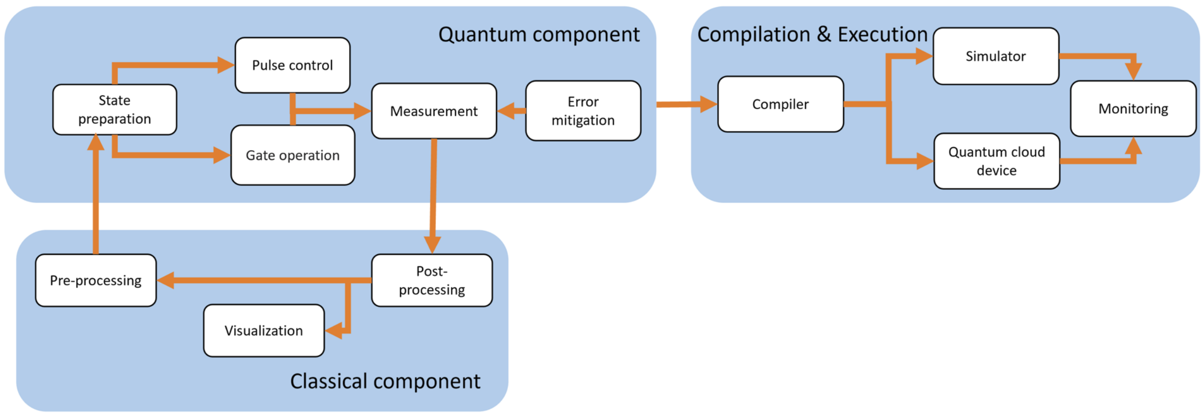

Figure 5.7 General components in a hybrid quantum program application

Table 5.3 Distribution of the quantum component bugs across the quantum software categories. (In the 376 analyzed bug reports, 43 reports were not identified as bugs.)

	VIS	SP	GO	Post-P	Pre-P	EM	CP	MS	PC	SM	MN	QCA	O	# bugs
Quantum programming framework	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	58
Experimental quantum computing	✓	✓	✓				✓	✓	✓	✓		✓	✓	47
Quantum algorithms	✓	✓	✓	✓	✓	✓	✓	✓				✓	✓	34
Quantum circuit simulators	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	43
Quantum compilers			✓				✓	✓	✓				✓	37
Quantum utility tools	✓		✓	✓	✓	✓	✓		✓	✓		✓	✓	41
Quantum-based simulation	✓	✓	✓	✓	✓		✓		✓		✓	✓	✓	38
Quantum Machine learning	✓	✓	✓	✓	✓	✓	✓	✓				✓	✓	35
Total Bugs	15	40	56	17	9	15	68	19	17	10	2	25	40	333

`simulators`, and `quantum utility tools`. We observe that the bugs are characterized mostly by `missing Error handling` and `performance bugs` with 2 bugs at each type as shown in table 5.4.

- **`State preparation (SP)`** is an operation to generate an arbitrary state of the qubit. As can be seen in Table 5.3, state preparation has the third-highest number of bugs with 40 bugs. The bugs appear in all the project categories except `quantum compilers` and `quantum utility tools` categories. During the state preparation phase, one tries to give the qubit an arbitrary state of 0 or 1. For this purpose, one tries to rotate or control the qubit into the desired state. As an example, one can use gate decomposition as a technique for state preparation. In Table 5.4, we observe that program anomaly is the most dominant bug type with 21 bugs, while data type and structure come second with only 5 bugs. However, we detected a rare bug of missing error handling type. Quantum states are mathematical entity that provide a probability distribution for each outcome of a state measurement. Since a qubit can have multiple states, mathematically they are represented as matrices. To initialize the state of a qubit, one has to manipulate that matrix. The number of state preparation bugs indicates the difficulty that developers are having with implementing the quantum algorithms. These bugs can be addressed with the help of data refinement and verification tools. Also, missing error handling bugs can be avoided by encouraging quantum developers to test their code.

- **`Measurement (MS)`** measures the final state of a qubit after superposition. We identified 19 bugs present in six of the quantum software categories as presented in Table 5.3. Six bug types were detected in our sample. From table 5.4 `Program anomaly bugs` are the most occurring with 12 bugs. Also, we identified 3 `configuration bugs` and 2 `data type and structure bugs`.

- **`Post-processing (Post-P)`** translates the quantum information (i.e., measurement probability) to classical computer bits. We detected 17 bugs related to `post-processing` distributed among all the quantum software categories except `experimental quantum computing` and `quantum compilers` as highlighted in the table 5.3. In table 5.4 we observe 9 `program anomaly` bug and 3 `gui related` bugs.

- **`Pre-processing (Pre-P)`** generates the state preparation circuit and initializes the register of the quantum computer. This component is executed on classical computer. We detected 17 bugs related to `post-processing` distributed among all the quantum software categories except `experimental quantum computing` and `quantum compilers` as highlighted in Table 5.3. In Table 5.4, we observe 9 `program anomaly` bugs and 3 `GUI related` bugs.

- **`Pulse control (PC)`** generates and controls signals to create custom gates and calibrate the qubits. From Table 5.3, the pulse control component registered 17 bugs during the manual analysis. Pulse control bugs generally appear when a developer would like to define a custom gate and manually calibrate the qubits. For this purpose, developers need to generate a pulse (signal) and a scheduler to calibrate the qubit. From Table 5.4, it appears that the two most occurring bug types in pulse control components are `program anomaly` and `test-code related`.

- **`Error mitigation (EM)`** corrects the measurement error when computing the qubit state probability. From Table 5.3 we identified 15 error mitigation bugs distributed along 5 quantum software categories. Among the identified bugs we have detected 5 `program anomaly` bugs and 4 `test-code related bugs` as shown in Table 5.4.

- **`Quantum cloud access (QCA)`** is a client interface to connect to a quantum cloud service provider. Cloud service providers like Amazon, Microsoft, and IBM provide access to quantum computers. We have detected 25 bugs related to the access to these quantum resources. `Configuration bugs` are the most occurring bug type with 8 bugs and `program anomaly bugs` are the second most frequent bug type in this category, with 5 bugs as shown in Table5.4.

- **`Visualisation (VIS)`** creates plots and visualisations. Quantum developers use visualization to represent qubit states graphically and examine quantum state vectors and the transformation actions. Drawing the circuit is the last step when building a quantum program. We identified 15 bugs in the visualization component, that appear in all the quantum programs categories except the `quantum compilers`. Among the detected bugs, 8 belong to the `GUI related bugs` category as shown in Table 5.4.

- **`Monitoring (MN)`** controls the good execution of the program and the performance of the execution environment. Only 2 bugs were assigned to the monitoring component, which appears in two quantum software categories (Quantum circuit simulation and Quantum-based simulation).

- **`Other (O)`**: Every bug that occurred elsewhere other than the defined components.

The result of our qualitative analysis for identifying the bugs in the quantum program software component is presented in the Table 5.3. Among the 376 analyzed bug reports, 43 questions reported bugs were not real bugs.

We identify 147 bugs in 5 components that are directly related to quantum computing including `gate operation`, `state preparation`, `measurement`,
`pulse control`, and `error mitigation`, ordered by their number of bugs. Moreover, the `simulator` shows a slightly high number of bugs (classical and quantum bugs), this can be explained by the complexity of the operation that has to be simulated since they are trying to simulate quantum physic phenomena using very complex linear algebra calculation. Finally, the `compilers` appear to be the most buggy with 63 bugs identified.

We identified a variety of 12 quantum computing components in which bugs were discovered, including both quantum computing-related bugs and classical-related bugs. We identified quantum-related bugs in five components (`gate operation`, `state preparation`, `measurement`, `pulse control`, and `error mitigation`.), among which `gate operation` is the most buggy component.

We identified **13 bug types in the quantum software ecosystem**. Table 5.4 shows the distribution of these bugs across quantum components. In the following, we discuss each type of bugs in more details. In our replication package [118], we provided more details and examples of different types of bugs in different quantum computing components.

`Program anomaly bugs` are introduced when enhancing existing code or caused by prior bad implementations. The manifestation of these bugs can be a bad return value or unexpected crashes due to logical issues in the code. For example in the `Pre-processing` component, to execute quantum programs efficiently, data is embedded into quantum bits. Specifically, the classical data is mapped into n-qubit quantum states by transforming data into a new space where it is linear. For example, AmplitudeEmbedding is a mapping technique in `PennyLaneAI` in which a bug[9] was reported and it took 6 days to fix. In the code from Listing 5.2(a), the normalization in the embedding function is breaking the differentiability for any data that is encoded as a `TensorFlow` or `Pytorch` tensor which triggers the error shown in Listing 5.2(b). Developers have tried pre-implemented normalization techniques in `NumPy` and `TensorFlow` but the bug persisted. The only solution left was to hand-code the normalization as illustrated in the bug fix code from Listing 5.2(c). Providing the quantum computing community with libraries dedicated to linear algebra computation in quantum will support the progress of quantum programming.

`Test-code related bugs` are happening in the test code. Problem reported due to adding

[9]`https://github.com/PennyLaneAI/pennylane/issues/365`

Table 5.4 Distribution of bug types across the quantum component

	VIS	SP	GO	Post-P	Pre-P	EM	CP	MS	PC	SM	MN	QCA	O	# bugs
Configuration bugs	3	3	4	-	-	2	11	3	1	2	-	8	10	47
Data types and structures bugs	3	5	10	1	3	-	12	2	2	1	-	2	5	46
Missing Error handling	-	1	-	-	-	1	4	-	3	2	-	2	-	12
Performance bugs	-	-	3	1	1	3	3	1	1	2	-	1	3	19
Permission/deprecation bugs	-	-	2	1	-	-	1	-	-	-	-	1	2	7
Program anomaly bugs	2	20	30	9	4	5	26	12	6	1	-	5	2	123
Test code-related bugs	-	2	4	-	-	4	12	1	4	-	1	-	5	33
DataBase related bugs	-	-	-	-	-	-	-	-	-	-	-	-	1	1
Documentation	-	1	3	-	-	-	-	-	-	1	-	1	11	17
Gui related bugs	8	3	1	3	-	-	-	-	-	-	-	-	-	15
Misuse	-	2	1	-	-	-	-	1	-	-	-	1	-	2
Network bugs	-	-	-	-	-	-	-	-	-	-	-	1	-	1
Monitoring	1	-	-	-	-	-	-	-	-	-	-	-	-	1

```
import numpy as np
import pennylane as qml
from pennylane.templates.embeddings import AmplitudeEmbedding

dev = qml.device('default.qubit', wires=3)
@qml.qnode(dev)
def circuit(data):
#Amplitude embedding normalization function not behaving as expected (Bug
    trigger)
    AmplitudeEmbedding(data, wires=[0, 1, 2])
    return qml.expval.PauliZ(0)

data = np.ones(shape=(8,)) / np.sqrt(8)
circuit(data)
```

(a) Trigger of the bug "PennyLaneAI (issue id 365)"

```
TypeError: unsupported operand type(s) for +: 'Variable' and 'Variable'
```

(b) Error message

```
#Repository: PennyLaneAI/pennylane
#Fix file: pennylane/templates/embeddings.py, method: AngleEmbedding, line
    : 118
-if normalize and np.linalg.norm(features, 2) != 1:
-    features = features * (1/np.linalg.norm(features, 2))
+norm = 0
+for f in features:
+if type(f) is Variable:
    +norm += np.conj(f.val)*f.val
+else:
    +norm += np.conj(f)*f
```

(c) Proposed bug fix

Listing 5.2 The execution of the code snippet (shown in (a)) triggers a message error (shown in (b)) because Amplitude embedding normalization in NumPy is too strict and does not allow small tolerance. The bug fix was to redefine the normalization function (as shown in (c))

or updating test cases and intermittently executed tests. For example, in the pulse control component, `Test code-related bug` is the second most occurring bug type. In `Qiskit` bug report number 2527[10], a unitest is failing because of a wrong parameter type. The expected input variable type in this test should be int, float, or complex. Therefore to fix the bug, a developer changed the parameter type from String to Integer as shown in Listing 5.3.

`Data type and structure bugs` are related to data type problems such as undefined or mismatch type and data structure bugs like bad shape bugs or the use of wrong data struc-

[10]`https://github.com/Qiskit/qiskit-terra/issues/2527`

```
1     #Repository: Qiskit/qiskit-terra
2     #Fix File: qiskit-terra/test/python/pulse/test_cmd_def.py
3     #Class: TestCmdDef, method:test_parameterized_schedule, line: 86
4     #Bug triggered in line 4 and 8
5     -sched = cmd_def.get('pv_test', 0, '0', P2=-1)
6     +sched = cmd_def.get('pv_test', 0, 0, P2=-1)
7     self.assertEqual(sched.instructions[0][-1].command.value, -1)
8     with self.assertRaises(PulseError):
9         -cmd_def.get('pv_test', 0, '0', P1=-1)
10        +cmd_def.get('pv_test', 0, 0, P1=-1)
11
```

Listing 5.3 The execution of the test building parameterized schedule fails because of the wrong parameter type (as shown in code snippet line 1 and 5). The bug fix was to change the parameter type from string to integer "qiskit (issue id 2527)"

tures. For example, in `qibo`, we identified a bug that states : "**`Probabilities do not sum to 1`**[11]" when running a circuit with more than 25 qubits. Listing 5.5(a) presents the buggy code along with the error message and the proposed bug fix. As can be seen, the problem has occurred because the `sum()` function does not return a normalized set of probabilities distribution. To fix the problem, a developer proposed to cast the tensor into type `tf.complex128`. This bug was caused by a data type and structure bug in the `measurement` component. Quantum software programs algorithms are based on probability and linear algebra which makes the implementation of quantum algorithms complex for developers. Therefore, providing libraries with array and data manipulation for quantum and data validation tools can help the quantum software engineering community.

`Missing error handling` occur when exceptions are not handled by the program. Improper error handling can lead to serious consequences for any system, and the quantum software systems are not an exception. For example, in `qutip`, we detect the absence of error handling in the simulator. An issue that led to the bug report id 396 [12]. The application crashed during the creation of a device object if the methods or attributes of this object contained an error. This issue was fixed by adding an exception block as shown in Listing 5.6. This bug fix occurred quickly (1 day) and not much code change was required. Improper error handling can be costly and lead to data leaks and many other exploits in the code. Therefore, developers must be careful regarding when, where, and how to correctly handle errors in the code. Code analysis tools for error handling and logging recommendation can support the quantum program development and help avoid `missing error handling` bugs.

`Configuration bugs` are related to configuration files building. The bug can be caused by

[11]`https://github.com/qiboteam/qibo/issues/517`
[12]`https://github.com/m-labs/artiq/issues/396`

```
NQUBITS = 26
c = Circuit(NQUBITS)
for i in range(NQUBITS):
    c.add(gates.H(i))
output = c.add(gates.M(TARGET, collapse=True))
for i in range(NQUBITS):
    c.add(gates.H(i))
# Running the circuit with more than 25 qubit return sum of qubit states
   probabilities =! 1 (Bug triggered)
result = c()
```

(a) Trigger of the bug "qibo (issue id 517)"

```
File "mtrand.pyx", line 933, in numpy.random.mtrand.RandomState.choice
ValueError: probabilities do not sum to 1
```

(b) Error message

```
#Repository:qiboteam/qibo
#Fix file: qibo/src/qibo/core/states.py
#Class:VectorState, method: probabilities(), line: 95
def probabilities(self, qubits=None, measurement_gate=None):
    unmeasured_qubits = tuple(i for i in range(self.nqubits) if i not in
   qubits)
-     state = K.reshape(K.square(K.abs(K.cast(self.tensor))), self.nqubits
   * (2,))
+     state = K.reshape(K.square(K.abs(K.cast(self.tensor, dtype="
   complex128"))), self.nqubits * (2,))
    return K.sum(state, axis=unmeasured_qubits)
```

(c) Bug fix

Listing 5.5 The execution of the code snippet (shown in (a)) triggers the message error (shown in (b)) because the sum of final state probabilities is different from 1. The bug fix was to cast the array value to complex type (as shown in (c))

the external library that must be updated or incorrect files paths or directories. For example, in the `Quantum cloud access` component, The bug in the code from Listing 5.7 happened in `DWave cloud access service` when trying to obtain the list of solvers. However, the list of solvers is filtered by the client type. To locate and fix the bug, developers spent 14 days [13]. Their proposed fix is s shown in Listing 5.7.

`GUI related bugs` are related to graphical elements such as layout, text box, and button layout. Quantum circuits are the main focus when building a quantum program. It can be challenging to debug and–or validate the structure of a quantum circuit by only reading the code. Libraries like `Cirq` and `Qiskit` offer visualization features to draw the architecture

[13] `https://github.com/dwavesystems/dwave-cloud-client/issues/457`

```
#Repository: m-labs/artiq
#Fix file: artiq/master/worker_db.py
#Class: DeviceManager, method: get, line: 144
-dev = _create_device(self.get_desc(name), self)
+try:
+    desc = self.get_desc(name)
+except Exception as e:
+    raise DeviceError("Failed to get description of device '{}'".format(
    name)) from e
+try:
+    dev = _create_device(desc, self)
+except Exception as e:
+    raise DeviceError("Failed to create device '{}'".format(name)) from e
```

Listing 5.6 Calling the device description causes the program to crash with inappropriate error message hard to understand.The bug fix was adding exception block in the get method with descriptive error message "artiq (issue id 396)".

```
#Repository:dwavesystems/dwave-cloud-client
#Fix file: dwave/cloud/cli.py, method: solvers, line:346
+@click.option('--client', 'client_type', default=None,
              type=click.Choice(['base', 'qpu', 'sw', 'hybrid'],
    case_sensitive=False),
              help='Client type used (default: from config)')
-def solvers(config_file, profile, solver_def, list_solvers, list_all):
+def solvers(config_file, profile, client_type, solver_def, list_solvers,
    list_all):
    if list_all:
+        client_type = 'base'
    with Client.from_config(
-            config_file=config_file, profile=profile, solver=solver_def)
    as client:
+            config_file=config_file, profile=profile,
+            client=client_type, solver=solver_def) as client:
```

Listing 5.7 Dwave solver is sensitive to the client type and does not return all the solvers with option -all because developers missed the client type argument in the solvers function. The bug fix was passing the client type as argument "dwave-cloud-client(issue id 457)

of circuits. The drawing is meant to depict the physical arrangement of gates, wires, and components. The visualization functions in `Qiskit` are basic support modules and can come with bugs. For example, the bug report[14] in `Qiskit` is stating misaligned barriers in the circuit, which change the interpretation of the circuit. In the code presented in Listing 5.9, we show a bug in Qiskit that occurred when a user wanted to draw a circuit, alongside with its proposed fix. Libraries dedicated to quantum circuit visualization such us `Matplotlib` and `Seaborn` in python help developers to analyze and debug the circuits.

[14]`https://github.com/Qiskit/qiskit-terra/issues/3107`

```python
qc = qk.QuantumCircuit(2)
#The circuit draw in latex places barrier in the wrong place
qc.draw(output='latex')
```

(a) Trigger of the bug qiskit (issue id 3107)

```python
#Repository: Qiskit/qiskit-terra
#Fix file:  qiskit/visualization/latex.py
#Class: QCircuitImage, method: _build_latex_array, line:772
-self._latex[start][column] = "\\qw \\barrier{" + str(span) + "}"
+self._latex[start][column - 1] += " \\barrier[0em]{" + str(span) + "}"
+self._latex[start][column] = "\\qw"
```

(b) Trigger of the bug

Listing 5.8 The execution of the code snippet (shown in (a)) plots a circuit with a barrier not aligned correctly. The bug fix was to adjust the barrier column index (shown in (b))

Performance bugs are related to the stability, speed, or response time of software resources. This category covers memory overuse, endless loops, and energy consumption. For example, in the **Qiskit** simulator, we found a bug that caused a huge performance regression in the simulation due to a large increase in serialization overhead when loading noise models from Python into C++. Even though the bug fix consisted only in one line of code, developers spent 4 days to locate and fix this bug. Quantum developers make a high use of quantum simulators, therefore, it is important to ensure that these simulators maintain a good performance. Creating a performance benchmark for quantum simulators can help the community to select the most appropriate simulators and encourage providers to improve the performance of their simulators.

```python
#qiskit/qiskit-aer
# Fix file:  qiskit/providers/aer/noise/errors/quantum_error.py, method:
   to_dict, line: 462
-instructions = [exp.to_dict()['instructions'] for exp in qobj.experiments
   ]
+instructions: [[op[0].assemble().to_dict() for op in circ.data]
                            for circ in self._circs]
```

Listing 5.9 Calling qiskit.assemble causes a huge performance regression in noisy simulator "qiskit-aer (issue id 1398)"

Permission/deprecation bugs. This category covers two type of bugs: (1) bugs related to the removal or modification of deprecated calls or APIs, and (2) bugs related to missing or incorrect API permission.

`Database related bugs` are related to the connection between the database and the main application.

`Documentation` issues are related to documentation typos, not up to date documentations and misleading function in documentation.

`Monitoring bugs` are related to bad logging practices such as wrong logging levels, too much logging, or missing log statement. Only 2 bugs were assigned to the monitoring component. For example, the bug report id 100[15] in `quisp` describes a problem of overlogging and misplaced log statements. The overloaded text has been making debugging difficult. Even though, logging is a known practice in software engineering, its effectiveness requires adequate logging statements at appropriate locations in the code. Tools for logging and log level recommendations can help support quantum developers and improve the quality of their applications.

A `Misuse` is a wrong usage of a function that leads to a bug in the code. Misuses appeared in **gate operation**, **state preparation** components. For example, in the Listing 5.12, we present a **Misuse** bug that occurred during the state preparation of the qubit. This bug occurred because the developer wrote the program in the wrong order. The purpose of the circuit presented in this example is to measure the state of qubit 0 and map the state into a classical bit. However, the program of this circuit raised a `CircuitError` as shown in Listing 5.12(b). This error is due to line `QuantumCircuit()` which is not doing anything since it is immediately over-ridden by `statepreparation()`. Moreover, the measurement is not mapped to the register (classical bit) because of the misuse of the `measure()` function. A developer proposed the following fix to correct the issue (Listing 5.12(c)).

`Network bugs` are related to connection or server issues. Network bug type is the rarest bug Where we identify one bug in the quantum cloud access component from table 5.4.

We identified 5 dominant bug type while doing the manual analysis: `program anomaly`, `data type and structure`, `configuration issues`, `test-code related` and `enhancement and feature request`. `Program anomaly` is the most dominant bug type with 123 bugs. While inspecting the bug reports and their commit message, we notice that most program anomaly bugs come from bad implementation in the quantum algorithm or mathematical formulation of the problem. In fact, gate operation and state preparation components are based on complex mathematics, and this may be challenging for developers. `Configuration issues` is the second most frequent bug type with 45 bugs. Most of the quantum concepts are based on linear algebra and use complex data type; this makes the implementation challenging for developers. Data type and structure is the third most

[15]`https://github.com/sfc-aqua/quisp/issues/100`

```
1 q = QuantumRegister(4)
2 c = ClassicalRegister(1)
3 circuit = QuantumCircuit(q,c)
4 ang = feature_train[3]
5 # Applying state preparation after quantum Circuit initialization triggers
      the bug
6 circuit = statepreparation(ang, circuit, [0,1,2,3])
7 circuit.measure(0, c)
```

(a) Trigger of the bug "qiskit (issue Id 5837)"

```
1 CircuitError: 'register not in this circuit'
```

(b) Error message

```
1 #Fix file: In the code snipped (shown in (a)), line 3 and 7
2 -circuit = QuantumCircuit(q,c)
3 -circuit.measure(0, c)
4 +circuit.measure(0, 0)
```

(c) Code to reproduce the bug

Listing 5.12 The execution of the code snipped (shown in (a)) triggers the message error (shown in (b)) because the quantum circuit (line 3 in code snipped (a)) is immediately overridden by the state preparation (line 6 in a code snipped (a)). The bug fix was to map the 0th measured qubit into the 0th classical bit and remove the circuit initialization (as shown in (c).

frequent category with 40 bugs. Again, this may be caused by the complexity of quantum operations.

We identified 13 different types of quantum bugs. The most frequent types of bugs occurring in quantum components are: program anomaly bugs with 123 bugs, configuration bugs with 47 bugs, and data type and structure bugs with 46 occurrences. Program anomaly bugs is the most spread bug type (it occurs in all components). Data type and structure bugs are mostly located in the components `State preparation` (10 occurrences) and `Compiler` (12 occurrences).

5.5 Discussion and implication

We have seen in the analysis that most of the bugs in the quantum software ecosystem are program anomalies, configuration bugs, and data type and structure bugs. These bugs can have a serious impact, causing the program to crash and leading to poor software quality.

We observe that quantum programmers have limited or no access to data, matrix, and array manipulation libraries. It is often difficult for them to implement quantum algorithms, linear algebra routines, error mitigation models, and unitary transformation with existing libraries such as `Numpy`. This finding suggests that developing **`data manipulation (e.g., array manipulation) libraries`** dedicated to quantum programming along with a collection of **`mathematical algorithms for quantum computing and convenience functions`** can help prevent some of the observed program anomalies and data type and structure bugs. Our analysis also revealed the difficulty of designing quantum circuits.. Automatic pattern recommendation tools could be developed to support this task. This can be done for example by mining a large set of expert-written quantum code to extract common code patterns that can be recommended to developers designing quantum circuits.

Because quantum programs are difficult to debug and quantum components are strongly coupled, **`circuit visualization and analysis`** approaches are needed to help developers examine the logical structure of their circuits and fix bugs early on.

Researchers and tool builders should consider contributing specialized **`data manipulation (e.g., array manipulation)`** libraries and libraries providing **`mathematical algorithms for quantum computing and convenience functions`** to support quantum software development, to help reduce the occurrence of program anomaly bugs and data type and structure bugs. **`Circuit visualization and analysis`** techniques and tools are also needed to help developers debug and fix bugs in quantum circuits.

5.6 Threats to Validity

We now discuss threats to the validity of our study.

External validity. In this work, we analyze the bugs of 125 quantum software projects on GitHub. Our studied projects may not represent the characteristics of other quantum software projects that are not public on GitHub. In addition, our studied projects may not represent all quantum software projects on GitHub. However, we followed a systematic approach to search for quantum software projects and focus on projects with relatively rich development activities.

Internal validity. In RQ1, we analyze the bug fixing efforts using the bug fix duration and the code changes in the bug fixes as proxies. However, the duration and code changes may not accurately capture the effort of developers in fixing these bugs. Future works that accurately monitor developers' development activities in fixing quantum software bugs can improve our analysis.

Construct validity. In our preliminary study and RQ2, we manually analyze the categories of the quantum software projects and the characteristics of the quantum software bugs. Our results may be subjective and depend on the judgment of the researchers who conducted the manual analysis. To mitigate this threat, two researchers collectively conducted a manual analysis and reached a substantial agreement, indicating the reliability of the analysis results. To resolve disagreements, the third researcher joined them and each case was discussed until reaching a consensus.

Conclusion validity. This threat concerns the extent to which the analyzed issues can be considered exhaustive enough. We have followed a systematic approach to identify the studied quantum projects. To identify the bug types and the components of the quantum program execution flow in which they occurred, we have manually analyzed a statistical representative sample of the detected bugs. Although it is possible that we may have missed some types of bugs, we have mitigated this threat by following a stratified sampling strategy that allowed us to cover projects from all quantum computing categories.

Reliability validity. This threat concerns the possibility to replicate this study. We have attempted to provide all the necessary details needed to replicate our study. We share our full replication package in [118].

5.7 Chapter summary

In this chapter, we perform an empirical study on 125 open-source quantum software projects hosted on GitHub. These quantum software projects cover a variety of categories, such as quantum programming frameworks, quantum circuit simulators, or quantum algorithms. An analysis of the development activity of these selected projects show a level of development activities similar to that of classical projects hosted on GitHub. We compared the distribution of bugs in quantum software projects and classical software projects, as well as developers' efforts in addressing these bugs and observed that quantum software projects are more buggy than comparable classical software projects. Besides, quantum software project bugs are more costly to fix (in terms of the code changed) than classical software project bugs. We qualitatively studied a statistically representative sample of quantum software bugs to understand their characteristics. We identified a total of 13 different types of bugs occurring in 12 quantum components. The three most occurring types of bugs are Program anomaly bugs, Configuration bugs, and Data type and structure bugs. These bugs are often caused by the wrong logical organization of the quantum circuit, state preparation, gate operation, measurement, and state probability expectation computation. Our study also highlighted the need for specialized data manipulation (e.g., array manipulation) libraries, libraries providing mathematical algorithms for quantum computing and convenience functions, as well as circuit visualization and analysis techniques and tools to support quantum software development.

CHAPTER 6 CONCLUSION

In this chapter, we summarise our results and conclude the thesis. Besides, we highlight the limitations of our approaches and outline some directions for future work.

6.1 Summary of Works

With the rapid increase of quantum computing popularity and it's active development in recent years, and with the unique advantages that this computation approach offers, it is promised to become a game changer in various fields. Similarly to classical software, it is critical for quantum software to ensure a good quality of service. In this thesis, we examined the quantum software engineering challenges in technical Q&A forums and GitHub issue reports that quantum computing practitioners discuss. We examined bugs in the quantum software ecosystem repositories in GitHub, to identify any new type of bugs and proposed a Taxonomy of bugs. We also formulated recommendations for researchers and tool builders. In chapter 2 we present background knowledge of concepts and terminologies that are helpful to better understand our empirical studies. In chapter 3 we reviewed the existing literature on quantum software engineering, software quality assurance, and topic modeling in software engineering studies. In chapter 4, we examined challenges that quantum program developers are facing by analyzing Stack Exchange forums posts related to QSE and the GitHub issue reports of quantum computing projects. Results indicate that QSE developers face traditional software engineering challenges (e.g., dependency management) as well as QSE-specific challenges (e.g., interpreting quantum program execution results). In particular, some QSE-related areas (e.g., bridging the knowledge gap between quantum and classical computing) are gaining the highest attention from developers while being very challenging to them. As the initial effort for understanding QSE-related challenges perceived by developers, our study shed light on future opportunities in QSE (e.g., supporting explanations of theory behind quantum program code and the interpretations of quantum program execution results). In chapter 5 we perform an empirical study on 125 open-source quantum software projects hosted on GitHub (same dataset used in Chapter 4). These quantum software projects cover a variety of categories, such as quantum programming frameworks, quantum circuit simulators, or quantum algorithms. An analysis of the development activity of these selected projects show a level of development activities similar to that of classical projects hosted on GitHub. We compared the distribution of bugs in quantum software projects and classical software projects, as well as developers' efforts in addressing these bugs and observed that quantum software projects are more buggy than comparable classical software projects. Besides, quantum software project bugs are more costly to fix (in terms of the code changed) than classical software project bugs. We qualitatively studied a statistically representative sample of quantum software bugs to understand their characteristics. We identified a total of 13 different types of bugs occurring in 12 quantum components. The three most occurring types of bugs are `Program anomaly bugs`, `Configuration bugs`, and `Data type and structure bugs`. These bugs are often caused by the wrong logical organization of the quantum circuit, state preparation, gate operation, measurement, and state probability expectation computation. Our study also highlighted the need for specialized data manipulation (e.g., array manipulation) libraries, libraries providing mathematical algorithms for quantum computing and convenience functions, as well as circuit visualization and analysis techniques and tools to support quantum software development.

Our study is the first that extensively investigate the quantum software engineering challenges of a large number of open-source projects and by mining the questions in Q&A forums. This work can help researchers and practitioners better understand the challenges and issues of quantum programming for which the demand is increasing rapidly. We hope that our work will encourage software engineering researchers to tackle the most important challenging tasks in quantum software engineering, such as quantum software debugging and testing.

6.2 Limitations

- We analyzed four Stack Exchange forums and 125 GitHub repositories to understand the challenges of QSE. Our studied forum posts and GitHub issues may not cover all the ones that are related to QSE. Developers may also communicate their discussions in other media (e.g., mailing lists). Also we may not represent the characteristics of other quantum software projects that are not public on GitHub

- In our thesis, we use LDA for topic modeling to cluster the forum posts and GitHub issue reports, based on the intuition that the same clusters would have similar textual information. However, different clusters of posts and issue reports may exist when a different approach is used.

- We manually analyze the categories of QSE-related questions on technical Q&A forums and classified the bug types. Our results may be subjective and depend on the judgment of the researchers who conducted the manual analysis.

6.3 Future Research

Quantum computing is starting to take giant strides but QSE is still in the beginning of the journey. There are many potential future research directions that arise from the thesis contributions. Bellow we outline some possible directions.

- We have highlighted new bugs in quantum software, and 13 quantum software engineering topics. It is, therefore, a new path for researchers in the future to explore these areas in depth.
- Evolution of the test practices in quantum program : In the future we hope to study how the quantum software are tested, maintained and evolved through out their development life-cycle. We still lack a taxonomy of test practices in the quantum software ecosystem. The community is in still in the dark about when and how quantum engineers introduce test cases, how their testing strategies may have changed overtime.
- The efficiency of the testing practices in quantum software : In the future work we plan to examine and evaluate the existing testing practices and their efficiency in the context of quantum programming. The results of such study could help quantum software engineers to choose the most optimal testing strategies, and if needed define new test practices for the quantum programming approach.
- Quantum computing proposes a different computation approach with much higher computation capability. The test minimization problem have existed since the early days of programming and it is known to be computationally expensive. In the future work, we aim to harness the computation benefits of quantum programming for test minimization.

REFERENCES

[1] S. Beyer *et al.*, "What kind of questions do developers ask on stack overflow? a comparison of automated approaches to classify posts into question categories," *Empirical Software Engineering*, vol. 25, no. 3, pp. 2258–2301, 2020.

[2] B. Sodhi, "Quality attributes on quantum computing platforms," *CoRR*, vol. abs/1803.07407, 2018. [Online]. Available: http://arxiv.org/abs/1803.07407

[3] J. Zhao, "Quantum software engineering: Landscapes and horizons," *arXiv preprint arXiv:2007.07047*, 2020.

[4] M. Fingerhuth, T. Babej, and P. Wittek, "Open source software in quantum computing," *PloS one*, vol. 13, no. 12, p. e0208561, 2018.

[5] Physicsworld.com, "Quantum-computing firm opens the box," https://web.archive.org/web/20110515083848/http://physicsworld.com/cws/article/news/45960, 2011, Accessed: 2022-01-10.

[6] L. Mueck, "Quantum software," *Nature*, vol. 549, no. 171, 2017.

[7] M. Piattini *et al.*, "Toward a quantum software engineering," *IT Professional*, vol. 23, no. 1, pp. 62–66, 2021.

[8] Q. O. S. Foundation, "List of open quantum projects," https://qosf.org/project_list/, 2021, Accessed: 2022-01-10.

[9] H. Abraham *et al.*, "Qiskit: An open-source framework for quantum computing," 2019.

[10] "Cirq," Mar. 2021, See full list of authors on Github: https://github.com/quantumlib/Cirq/graphs/contributors. [Online]. Available: https://doi.org/10.5281/zenodo.4586899

[11] IonQ, "MS Windows NT kernel description," https://ionq.com/, 2018, accessed: 2021-10-20.

[12] E. Moguel *et al.*, "A roadmap for quantum software engineering: applying the lessons learned from the classics," in *Proceedings of the 1st International Workshop on Software Engineering ++& Technology*, ser. Q-SET '20, 2020.

[13] B. Nachman *et al.*, "Unfolding quantum computer readout noise," *npj Quantum Information*, vol. 6, no. 1, pp. 1–7, 2020.

[14] M. Piattini *et al.*, "The talavera manifesto for quantum software engineering and programming." in *The First International Workshop on the Quantum Software Engineering & Programming*, ser. QANSWER '20, 2020, pp. 1–5.

[15] E. Allen, "Bug patterns in java," 2002.

[16] S. Beyer *et al.*, "What kind of questions do developers ask on stack overflow? a comparison of automated approaches to classify posts into question categories," in *Software Engineering*, 2021.

[17] P. Kaye *et al.*, *An introduction to quantum computing.* Oxford University Press on Demand, 2007.

[18] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014.

[19] IBM, "Survey of ibm-q, circuit composer, backends. install qiskit locally and get started with tutorials," http://www.web.uvic.ca/~rdesousa/teaching/P280/L15_280.pdf, 2021, accessed: 2021-10-20.

[20] R. Fein, "The evolving quantum computing ecosystem," https://medium.com/@russfein/the-evolving-quantum-computing-ecosystem-bed1805007f8, Medium, 2021, accessed: 2022-04-15.

[21] P. Forn-Diaz, "Superconducting qubits and quantum resonators," Ph.D. dissertation, Delft University of Technology, 09 2010.

[22] Y. Gao *et al.*, "Pulse-qubit interaction in a superconducting circuit under frictively dissipative environment," *arXiv preprint arXiv:2002.06553*, 2020.

[23] M. F. Brandl, "A quantum von neumann architecture for large-scale quantum computing," 2017.

[24] A. Roy and M. Devoret, "Quantum-limited parametric amplification with josephson circuits in the regime of pump depletion," *Physical Review B*, vol. 98, no. 4, p. 045405, 2018.

[25] J. Cramer *et al.*, "Repeated quantum error correction on a continuously encoded qubit by real-time feedback," *Nature communications*, vol. 7, no. 1, pp. 1–7, 2016.

[26] TechTarget, “classical computing,” https://www.techtarget.com/whatis/definition/classical-computing, 2018, Accessed: 2022-04-18.

[27] M. Piattini, G. Peterssen, and R. Pérez-Castillo, “Quantum computing: A new software engineering golden age,” *ACM SIGSOFT Software Engineering Notes*, vol. 45, no. 3, pp. 12–14, 2020.

[28] L. S. Barbosa, “Software engineering for’quantum advantage’,” in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 2020, pp. 427–429.

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

[30] R. LaRose, “Overview and comparison of gate level quantum software platforms,” *Quantum*, vol. 3, p. 130, 2019.

[31] D. Deutsch and R. Penrose, “Quantum theory, the church&#x2013;turing principle and the universal quantum computer,” *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, vol. 400, no. 1818, pp. 97–117, 1985. [Online]. Available: https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1985.0070

[32] E. Knill, “Conventions for quantum pseudocode,” 6 1996. [Online]. Available: https://www.osti.gov/biblio/366453

[33] J. W. Sanders and P. Zuliani, “Quantum programming,” in *Mathematics of Program Construction*, ser. Lecture Notes in Computer Science, vol. 1837. Springer, 2000, pp. 80–99.

[34] H. MLNAŘÍK, “Semantics of quantum programming language lanq,” *International Journal of Quantum Information*, vol. 06, no. supp01, pp. 733–738, 2008. [Online]. Available: https://doi.org/10.1142/S0219749908004031

[35] K. Svore *et al.*, “Q#: Enabling scalable quantum computing and development with a high-level dsl,” in *Proceedings of the Real World Domain Specific Languages Workshop 2018*, ser. RWDSL2018. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3183895.3183901

[36] B. Weder *et al.*, “Quantum software development lifecycle,” *CoRR*, vol. abs/2106.09323, 2021. [Online]. Available: https://arxiv.org/abs/2106.09323

[37] J. Campos and A. Souto, "Qbugs: A collection of reproducible bugs in quantum algorithms and a supporting infrastructure to enable controlled quantum software testing and debugging experiments," *2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE)*, pp. 28–32, 2021.

[38] P. Zhao, J. Zhao, and L. Ma, "Identifying bug patterns in quantum programs," in *2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE)*. IEEE, 2021, pp. 16–21.

[39] M. Paltenghi and M. Pradel, "Bugs in quantum computing platforms: An empirical study," *CoRR*, vol. abs/2110.14560, 2021.

[40] W. K. Wootters and W. Zurek, "A single quantum cannot be cloned," *Nature*, vol. 299, pp. 802–803, 1982.

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

[42] G. Li *et al.*, "Projection-based runtime assertions for testing and debugging quantum programs," *Proceedings of the ACM on Programming Languages*, vol. 4, no. OOPSLA, pp. 1–29, 2020.

[43] N. Yu and J. Palsberg, "Quantum abstract interpretation," in *Association for Computing Machinery*, ser. PLDI 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 542–558.

[44] A. Chou *et al.*, "An empirical study of operating systems errors," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, p. 73–88, oct 2001.

[45] C. Sun *et al.*, "Toward understanding compiler bugs in gcc and llvm," 07 2016, pp. 294–305.

[46] M. J. Islam *et al.*, "A comprehensive study on deep learning bug characteristics," *CoRR*, vol. abs/1906.01388, 2019.

[47] T. Zhang *et al.*, "A novel developer ranking algorithm for automatic bug triage using topic model and developer relations," in *2014 21st Asia-Pacific Software Engineering Conference*, vol. 1. IEEE, 2014, pp. 223–230.

[48] X. Xia *et al.*, "Accurate developer recommendation for bug resolution," in *2013 20th Working Conference on Reverse Engineering (WCRE).* IEEE, 2013, pp. 72–81.

[49] H. Naguib *et al.*, "Bug report assignee recommendation using activity profiles," in *2013 10th Working Conference on Mining Software Repositories (MSR).* IEEE, 2013, pp. 22–30.

[50] X. Xia *et al.*, "Improving automated bug triaging with specialized topic model," *IEEE Transactions on Software Engineering*, vol. 43, no. 3, pp. 272–297, 2016.

[51] A. Hindle, A. Alipour, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection and ranking," *Empirical Software Engineering*, vol. 21, no. 2, pp. 368–410, 2016.

[52] A. T. Nguyen *et al.*, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering.* IEEE, 2012, pp. 70–79.

[53] J. Zou *et al.*, "Duplication detection for software bug reports based on topic model," in *2016 9th International Conference on Service Science (ICSS).* IEEE, 2016, pp. 60–65.

[54] A. T. Nguyen *et al.*, "A topic-based approach for narrowing the search space of buggy files from a bug report," in *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011).* IEEE, 2011, pp. 263–272.

[55] L. Martie *et al.*, "Trendy bugs: Topic trends in the android bug reports," in *2012 9th IEEE Working Conference on Mining Software Repositories (MSR).* IEEE, 2012, pp. 120–123.

[56] A. Aggarwal, G. Waghmare, and A. Sureka, "Mining issue tracking systems using topic models for trend analysis, corpus exploration, and understanding evolution," in *Proceedings of the 3rd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, 2014, pp. 52–58.

[57] B. Vasilescu, "Academic papers using stack exchange data," https://meta.stackexchange.com/questions/134495/academic-papers-using-stack-exchange-data, Last accessed 05/04/2021.

[58] S. Wang, D. Lo, and L. Jiang, "An empirical study on developer interactions in stackoverflow," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 2013, pp. 1019–1024.

[59] H. Chen, J. Coogle, and K. Damevski, “Modeling stack overflow tags and topics as a hierarchy of concepts,” *Journal of Systems and Software*, vol. 156, pp. 283–299, 2019.

[60] M. Allamanis and C. Sutton, “Why, when, and what: analyzing stack overflow questions by topic, type, and code,” in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 53–56.

[61] M. Linares-Vásquez, B. Dit, and D. Poshyvanyk, “An exploratory analysis of mobile development issues using stack overflow,” in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 93–96.

[62] C. Rosen and E. Shihab, “What are mobile developers asking about? a large scale study using stack overflow,” *Empirical Software Engineering*, vol. 21, no. 3, pp. 1192–1223, 2016.

[63] P. K. Venkatesh *et al.*, “What do client developers concern when using web apis? an empirical study on developer forums and stack overflow,” in *2016 IEEE International Conference on Web Services (ICWS)*. IEEE, 2016, pp. 131–138.

[64] M. Alshangiti *et al.*, “Why is developing machine learning applications challenging? a study on stack overflow posts,” in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2019, pp. 1–11.

[65] S. Ahmed and M. Bagherzadeh, “What do concurrency developers ask about? a large-scale study using stack overflow,” in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2018, pp. 1–10.

[66] X.-L. Yang *et al.*, “What security questions do developers ask? a large-scale study of stack overflow posts,” *Journal of Computer Science and Technology*, vol. 31, no. 5, pp. 910–924, 2016.

[67] J. Zou *et al.*, “Towards comprehending the non-functional requirements through developers’ eyes: An exploration of stack overflow using topic analysis,” *Information and Software Technology*, vol. 84, pp. 19–32, 2017.

[68] ——, “Which non-functional requirements do developers focus on? an empirical study on stack overflow using topic analysis,” in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 446–449.

[69] Y. Zhang *et al.*, “Multi-factor duplicate question detection in stack overflow,” *Journal of Computer Science and Technology*, vol. 30, no. 5, pp. 981–997, 2015.

[70] C. Treude and M. Wagner, “Predicting good configurations for github and stack overflow topic models,” in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 84–95.

[71] W. Knight, “Serious quantum computers are finally here. what are we going to do with them,” *MIT Technology Review*, vol. 30, 2018.

[72] D. Maslov, Y. Nam, and J. Kim, “An outlook for quantum computing [point of view],” *Proceedings of the IEEE*, vol. 107, no. 1, pp. 5–10, 2018.

[73] IBM, “IBM Quantum Computing,” https://www.ibm.com/quantum-computing/, Last accessed 05/04/2021.

[74] P. A. M. Dirac, *The principles of quantum mechanics.* Oxford university press, 1981, no. 27.

[75] E. Schrödinger, “Discussion of probability relations between separated systems,” in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 31, no. 4. Cambridge University Press, 1935, pp. 555–563.

[76] B. Ömer, “Qcl-a programming language for quantum computers,” *Software available on-line at http://tph. tuwien. ac. at/˜ oemer/qcl. html*, 2003.

[77] Google, “Google QuantumAI,” https://quantumai.google, Last accessed 05/04/2021.

[78] Microsoft, “Microsoft Azure Quantum Service,” https://azure.microsoft.com/en-ca/services/quantum/, Last accessed 05/04/2021.

[79] J. Biamonte *et al.*, “Quantum machine learning,” *Nature*, vol. 549, no. 7671, pp. 195–202, 2017.

[80] G. G. Guerreschi and M. Smelyanskiy, “Practical optimization for hybrid quantum-classical algorithms,” *arXiv preprint arXiv:1701.01450*, 2017.

[81] L. O. Mailloux *et al.*, “Post-quantum cryptography: what advancements in quantum computing mean for it professionals,” *IT Professional*, vol. 18, no. 5, pp. 42–47, 2016.

[82] M. Reiher *et al.*, “Elucidating reaction mechanisms on quantum computers,” *Proceedings of the National Academy of Sciences*, vol. 114, no. 29, pp. 7555–7560, 2017.

[83] “Stack Overflow forum,” https://stackoverflow.com, Last accessed 05/04/2021.

[84] "Stack Exchange Quantum Computing forum," https://quantumcomputing.stackexchange.com, Last accessed 05/04/2021.

[85] "Stack Exchange Computer Science forum," https://cs.stackexchange.com, Last accessed 05/04/2021.

[86] "Stack Exchange Artificial Intelligence forum," https://ai.stackexchange.com, Last accessed 05/04/2021.

[87] "stack exchange data explorer."

[88] G. Uddin *et al.*, "Understanding how and why developers seek and analyze api-related opinions," 2021.

[89] M. Openja, B. Adams, and F. Khomh, "Analysis of modern release engineering topics : – a large-scale study using stackoverflow –," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2020, pp. 104–114.

[90] B. Cartaxo *et al.*, "Using q&a websites as a method for assessing systematic reviews," 05 2017.

[91] J. Businge *et al.*, "Clone-based variability management in the android ecosystem," *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 625–634, 2018.

[92] ——, "Studying android app popularity by cross-linking github and google play store," *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 287–297, 2019.

[93] "GitHub REST API," https://developer.github.com/v3/, Last accessed 05/04/2021.

[94] P. Willett, "The porter stemming algorithm: Then and now," *Program electronic library and information systems*, vol. 40, 07 2006.

[95] V. Gurusamy and S. Kannan, "Performance analysis: Stemming algorithm for the english language," *International Journal for Scientific Research and Development*, vol. 5, pp. 2321–613, 08 2017.

[96] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.

[97] T.-H. Chen, S. W. Thomas, and A. E. Hassan, "A survey on the use of topic models when mining software repositories," *Empirical Software Engineering*, vol. 21, no. 5, pp. 1843–1919, 2016.

[98] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? an analysis of topics and trends in stack overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014.

[99] R. Rossi and S. Rezende, "Generating features from textual documents through association rules," 01 2011.

[100] A. K. McCallum, "Mallet: A machine learning for language toolkit," 2002, http://mallet.cs.umass.edu.

[101] M. Röder, A. Both, and A. Hinneburg, "Exploring the space of topic coherence measures," *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, 2015.

[102] "Gensim coherencemodel implimentation," https://radimrehurek.com/gensim/models/coherencemodel.html, Last accessed 05/04/2021.

[103] J. Han *et al.*, "What do programmers discuss about deep learning frameworks," *Empirical Software Engineering*, vol. 25, pp. 2694–2747, 2020.

[104] H. Li *et al.*, "A qualitative study of the benefits and costs of logging from developers' perspectives," *IEEE Transactions on Software Engineering*, 2020.

[105] M. McHugh, "Interrater reliability: The kappa statistic," *Biochemia medica : časopis Hrvatskoga društva medicinskih biokemičara / HDMB*, vol. 22, pp. 276–82, 10 2012.

[106] X.-L. Yang *et al.*, "What security questions do developers ask? a large-scale study of stack overflow posts," *Journal of Computer Science and Technology*, vol. 31, pp. 910–924, 09 2016.

[107] M. Bagherzadeh and R. Khatchadourian, "Going big: A large-scale study on what big data developers ask," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 432–442. [Online]. Available: https://doi.org/10.1145/3338906.3338939

[108] E. Noei, F. Zhang, and Y. Zou, "Too many user-reviews, what should app developers look at first?" *IEEE Transactions on Software Engineering*, 2019.

[109] F. Khomh *et al.*, "An entropy evaluation approach for triaging field crashes: A case study of mozilla firefox," in *2011 18th Working Conference on Reverse Engineering*, 2011, pp. 261–270.

[110] E. Farhana, N. Imtiaz, and A. Rahman, "Synthesizing program execution time discrepancies in julia used for scientific software," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2019, pp. 496–500.

[111] E. Kalliamvakou *et al.*, "An in-depth study of the promises and perils of mining github," *Empirical Software Engineering*, vol. 21, pp. 2035–2071, 2016.

[112] L. Tan *et al.*, "Bug characteristics in open source software," *Empirical Software Engineering*, vol. 19, pp. 1665–1705, 2013.

[113] W. Knight, "MS Windows NT kernel description," https://www.technologyreview.com/2017/11/10/147728/ibm-raises-the-bar-with-a-50-qubit-quantum-computer/, MIT Technology Review, 2017, accessed: 2021-10-20.

[114] Physicsworld.com, "Quantum-computing firm opens the box," https://physicsworld.com/a/ion-based-commercial-quantum-computer-is-a-first/, 2011, Accessed: 2022-01-10.

[115] J. Aron, "Ibm unveils its first commercial quantum computer," https://www.newscientist.com/article/2189909-ibm-unveils-its-first-commercial-quantum-computer/, New Scientist, 2019, accessed: 2022-01-16.

[116] G. Catolino *et al.*, "Not all bugs are the same: Understanding, characterizing, and classifying bug types," *Journal of Systems and Software*, vol. 152, pp. 165–181, 2019.

[117] B. Sodhi, "Quality attributes on quantum computing platforms," *arXiv preprint arXiv:1803.07407*, 2018.

[118] Replication Package, "Study of the bug charateristics in the quantum ecosystem," https://github.com/raed1337/Quantum-bug-study/tree/master, SWAT, 2022, accessed: 2022-04-20.