UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

An Exploratory Study on the Usage of Quantum Programming Languages

Felipe Cavalcanti Ferreira

Mestrado em Engenharia Informática

Dissertação orientada por:
Prof. Doutor José Carlos Medeiros de Campos

2022

Acknowledgments

Initially, I would like to thank my family for all the support, always encouraging me and celebrating my victories. Especially my wife, for her patience, understanding, love, and taking care of our baby; this project would not be possible without her. To my family, I thank the encouragement for completing this thesis, which means the conclusion of a journey of academic and personal growth. To my dissertation supervisor, Prof. José Campos, I would like to thank him for the guidance provided and for his insights, availability, support, and sharing of knowledge. Here I express my gratitude to him. Thanks also to all the others I did not mention here but who contributed to the realization of this thesis.

I dedicate this thesis to my wife and my son. Thank you for understanding my hours of absence.

Resumo

O uso de linguagens de programação quântica na computação quântica permite instruir um computador quântico para executar certas tarefas. Na computação clássica, os computadores manipulam bits individuais, armazenando dados nos estados binários de 0 e 1. Os computadores quânticos são construídos utilizando bits quânticos (qubits), e seguem as leis quânticas em vez das leis de Newton, o que lhes permite ter a probabilidade de serem 0 e 1 simultaneamente. Na computação quântica, essa informação é manipulada aproveitando as principais propriedades da mecânica quântica, como superposição e emaranhamento. A superposição é a capacidade de um sistema quântico estar em vários estados ao mesmo tempo até que seja medido. Já o emaranhamento quântico é o fenômeno físico que ocorre quando um grupo de partículas, ao ser gerado, interage ou compartilha proximidade espacial de tal forma que o estado quântico de cada partícula do grupo não pode ser descrito independentemente do estado das outras, inclusive quando as partículas são separadas por uma grande distância.

As linguagens começaram a ser desenvolvidas já no início da criação dos dispositivos quânticos e são usadas para simular algoritmos quânticos. Nesta investigação, identificamos 37 linguagens de programação quântica propostas desde 1996. É provável que muitas outras estejam em desenvolvimento, dado o rápido desenvolvimento da área de computação quântica. O grande número de linguagens de programação quântica chama atenção para melhor investigar por que existem tantas linguagens; como essas linguagens estão sendo usadas e por quem; o que torna uma linguagem quântica popular; entre outros aspectos. Embora várias linguagens de programação quântica com diferentes recursos e objetivos tenham sido desenvolvidas nos últimos 25 anos, até o momento nenhum estudo havia sido realizado sobre quem, como e para quê se usa uma linguagem de programação quântica.

Nesta tese, identificamos e descrevemos várias linguagens de programação quântica, resumindo as características e principais funcionalidades de cada linguagem, bem como o ano em que foram criadas; se são open source, ativas, acadêmicas ou industriais; seu autor; e qual o tipo da linguagem (imperativa, declarativa ou multiparadigma). Também entrevistamos 251 desenvolvedores para responder a várias questões propostas neste estudo relacionadas ao uso de linguagens de programação quântica. O inquérito foi divulgado em canais específicos de redes sociais sobre computação quântica, a exemplo de

grupos no Facebook, Linkedin, Discord e Twitter. A divulgação também foi feita em outros sítios online, como Slack e mailing. Também foi enviado o inquérito, por e-mail, para autores de artigos sobre linguagens de programação quântica e para outros pesquisadores no campo da computação quântica.

O inquérito foi realizado em duas fases: a primeira, com apenas 2 participantes para pré-testar as perguntas, validá-las e calibrar o tempo estimado para respondê-las. Após essa primeira fase, algumas correções foram necessárias e o inquérito foi atualizado com as lições aprendidas. Passou-se, então, à segunda fase, quando o inquérito foi divulgado e realizado com 251 participantes.. O inquérito ficou aberto por aproximadamente 60 dias, e um lembrete foi enviado para os possíveis participantes após 10 dias. Depois deste periódo, o inquérito foi encerrado e os dados coletados foram submetidos a análises para ajudar a responder as questões propostas neste estudo.

Após a análise dos resultados, o estudo mostra que quase 90,0% dos participantes que responderam ao inquérito e utilizam linguagens de programação quântica são do sexo masculino e 22,2% deles moram nos Estados Unidos da América. 58,2% têm entre 25 e 44 anos. 63,0% possuem mestrado ou doutorado. 86,2% têm mais de cinco anos de experiência utilizando linguagens de programação clássicas. 42,8% dos participantes usam linguagens de programação quântica para pesquisa, enquanto 34,6% as usam porque gostam de aprender novas linguagens, 16,4% as usam para trabalhar e 6,3% as usam para outras tarefas diversas.

De acordo com as respostas dos participantes, Qiskit (Python) é a linguagem de programação quântica mais utilizada, sendo utilizada por mais de 85,0% dos participantes que responderam ao inquérito, seguida por Cirq (Python) com 43,8% e OpenQASM com 37,0% dos participantes. Python, por sua vez, é a linguagem clássica mais utilizada para construir programas quânticos, utilizada por 92,3% dos participantes. Os participantes mencionaram várias razões pelas quais escolhem utilizar uma linguagem de programação quântica. As mais mencionadas foram o fato de apresentarem sintaxe de código aberto, de serem fáceis de aprender (baseadas em uma linguagem de programação existente), a disponibilidade de documentação, exemplos e recursos, bem como de uma comunidade ativa.

Cirq, com 25,5%, foi a linguagem de programação quântica mais escolhida pelos participantes como aquela que gostariam de aprender em um futuro próximo, seguida de Qiskit com 20,2% e Q# com 18,8%. O principal motivo dessas escolhas, segundo os entrevistados, foi que “ouviram falar da linguagem”, indicando a credibilidade e a visibilidade mundial das grandes empresas de tecnologia, como Google, IBM e Microsoft, responsáveis pelo desenvolvimento de tais linguagens.

Um percentual de 31,7% dos participantes do inquérito respondeu que entende necessário desenvolver outra linguagem de programação quântica. Em contrapartida, 25,0%, deles responderam que não pensam desse modo. A principal razão apontada para a ne-

cessidade de outra linguagem de programação quântica foi o desenvolvimento de uma linguagem de programação quântica mais madura, padronizada, de alto nível e com novos recursos (por exemplo, manipulação de correção de erros em dispositivos reais). Uma fatia de 42,8% dos participantes respondeu que existem muitas linguagens de programação quântica; em contrapartida, 18,3% responderam que não existem, enquanto 38,9% não responderam a esta pergunta por diferentes motivos, como não saber ou não ter conhecimento para respondê-la.

Ao final da tese, com base nos resultados da investigação, fazemos recomendações para o desenvolvimento mais aprimorado de linguagens de programação quântica. Em primeiro lugar, sugerimos que linguagens de programação quântica adotem certos recursos básicos amplamente adotados nas linguagens clássicas (por exemplo, generalidade, completude, extensibilidade e expressividade). Ainda, sugerimos que, ao desenvolver uma nova linguagem, os autores forneçam tutoriais, materiais de ensino e recursos para que os desenvolvedores possam aprender como usar a linguagem para desenvolver seus algoritmos. Essa documentação deve estar disponível para que os desenvolvedores a consultem sempre que necessário. Os exemplos de código também beneficiam os desenvolvedores, especialmente para aprender uma nova linguagem. Para incentivar o uso da linguagem, sugerimos que possua uma comunidade extensa e ativa, permitindo que os desenvolvedores troquem informações entre si, ajudando-os a solucionar dúvidas e problemas encontrados em seus códigos.

Sugerimos, ademais, que as linguagens de programação quântica podem ter patrocínio corporativo, inclusive para que sempre haja desenvolvedores trabalhando para evoluir e dar-lhes suporte. Ainda, sugerimos a possibilidade de que programas quânticos rodem em computadores quânticos reais, ao invés de serem usados apenas em simuladores, de modo que os desenvolvedores sejam motivados a aprender e a usar a linguagem de programação quântica para desenvolver seus programas. Ademais, construí-las sobre linguagens de programação clássica, caso suportem, torna provável que os desenvolvedores as utilizem mais facilmente, porque tendem a adotar linguagens com sintaxe que lhes é familiar.

Concluímos, ainda, que uma linguagem de programação quântica, para ser bem-sucedida, deve fornecer o maior número possível de recursos e ferramentas para os desenvolvedores construírem um programa quântico, como IDE's especializados em programas quânticos, simuladores e bibliotecas para programação quântica. Uma última e essencial recomendação feita na tese é a de consultar os desenvolvedores de programas quânticos para entender suas reais necessidades, nomeadamente quais os principais recursos de que precisam, quais os principais problemas encontrados nas linguagens que utilizam atualmente e o que pode ser implementado para melhorá-las.

Para trabalhos futuros, pensamos que seria interessante estender o estudo com linguagens de programação quântica adicionais e incluir linguagens recém-projetadas, repetindo o inquérito de forma a incluir tais linguagens.

Palavras-chave: Computação quântica, Programação quântica, Linguagens de programação quântica, Linguagens quânticas, Inguérito

Abstract

As in the classical realm, the usage of quantum programming languages in quantum computing allows one to instruct a quantum computer to perform certain tasks. Although several imperative, declarative, and multi-paradigm quantum programming languages with different features and goals have been developed in the last 25 years, no study has been conducted on who, how, and what for does one use a quantum programming language. In this thesis, we first identified and described several quantum programming languages and then surveyed 251 quantum practitioners to answer several questions related to the usage of quantum programming languages. Further, an analysis of the results obtained is presented and shows that most of the quantum practitioners use the languages for research and that Qiskit (Python) is the most used one. Finally, we make recommendations for further development of quantum programming languages, such as building on top of a classical programming language, running in real quantum computers, supporting language documentation, and consulting developers' needs.

Keywords: Quantum computing, Quantum programming, Quantum programming languages, Quantum languages, Survey

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

All computers are linked to the ability to store and manipulate information. In classical computing, computers manipulate individual bits, storing data in the binary states of 0 and 1. In quantum computing, this information is manipulated by taking advantage of key properties of quantum mechanics, like superposition and entanglement [1]. To take this advantage, quantum computing uses quantum bits or qubits. Therefore, quantum software allows manipulating qubits, translating a few lines of code into real quantum physical phenomena in quantum hardware. In this context, programming languages are needed to communicate with these computers using different levels of programming, from the lowest level, such as assembly languages, to the highest levels, where we can program algorithms for quantum hardware. A quantum computer is composed of a hybrid machine with a quantum device and a traditional computer that sends information to these devices and processes the result. Quantum programming languages relate to the capability of classical software capable in help developers implement quantum algorithms. These languages began to be developed from the beginning of the creation of quantum devices and are used to simulate quantum algorithms.

1.2 Problem

Over the last 25 years, researchers and worldwide leading software companies (which are also developing quantum computers) have developed several quantum programming languages and libraries on top of the existing classical languages for their quantum devices. For example, Google has developed Cirq, IBM has developed Qiskit, and Microsoft has developed a new language named Q#. In this thesis (see Figure 4.2), we have identified 37 quantum programming languages that have been proposed since 1996. It is likely that many others are under development, given the rapid development of the quantum computing area.

A large number of quantum programming languages call attention to investigate further why there are so many languages, how are quantum programming languages being used and by whom, what makes a quantum language popular, and others. The answer to these and other questions might shed light on, e.g., whether it would be interesting to focus our efforts on developing a new language with a universal syntax to make life easier for developers, whether it would be interesting to focus efforts on the enhancement of existing ones, or simply provide information for the developing of the next generations of quantum programming languages that will be more adapted to the developers' needs.

1.3 Approach

This thesis intends to investigate the usage of quantum programming languages. To achieve this goal, we first searched for quantum programming languages that have been proposed either in the literature or computing blogs/forums. The search was conducted at Google Scholar, Google search engine, and arXiv, and then identified their main functionalities and characteristics. Secondly, we conducted a survey with 251 quantum developers and researchers familiar with quantum programming languages to assess their opinion on languages. Finally, we analyzed the survey data and answered our research questions.

1.4 Contributions

The main contributions of this thesis are as follows:

- An overview of the state of art of quantum programming languages.
- A comparison of 37 quantum programming languages, including their main functionalities and characteristics.
- A survey with 251 developers and researchers to assess what quantum programming language they use, how they use them, and their thoughts on quantum programming languages in general.
- A detailed set of implications of our findings for the further development of quantum programming languages.

1.5 Structure of the document

This document is organized as follows: In Section 2, we describe the background of our work. Section 3 describes the related work and their contributions. In Section 4, we provide a brief introduction to the fundamental concepts of quantum programming languages, investigate and compare quantum languages that have been proposed by others. Then, in Section 5, we describe the structure and details of our survey to assess research and developers' thoughts on quantum programming languages. Section 6 presents the

results of the survey and answers our proposed research questions. Section 7 details a few implications for future development of quantum programming languages. Finally, in Section 8, we summarize the work conducted in this thesis and provide some ideas for future work.

Chapter 2

Background

2.1 Quantum Computing

Quantum computers are expected to reduce power consumption by 100 to 1000 times because the processor operates as a superconductor (that is, it can conduct electricity with almost no resistance), operating at a very low-temperature [2, 3]. Quantum computers, in contrast, can increase processes, such as machine learning, reducing thousands of years of learning to a few seconds [4, 5, 6]. Quantum computers are built on quantum bits (qubits), and they follow quantum laws instead of Newton's laws, which allows them to have the probability of being 0 and 1 simultaneously. Quantum computing studies the type of information processing that can be performed by physical systems governed by the laws of quantum mechanics. Big technology companies like Google and IBM are developing their own quantum computers because of the potential of quantum computing. Furthermore, these companies provide access to their quantum computers for developers to learn how to program quantum circuits using quantum programming languages.

2.2 Qubits

The smallest unit of information in classical computers is the bit. The quantum computing counterpart is a quantum bit or qubit. A classical bit has a discrete value of 0 or 1; consequently, a qubit does not have a discrete value. Instead, it represents 0 or 1 with some probability. More formally, a qubit is simultaneously in a superposition of the states 0 and 1.

A pure qubit state is a coherent superposition of the basis states, meaning that a single qubit can be described by a linear combination of $|0\rangle$ and $|1\rangle$:

$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$

The ket notation, Dyrac notation ($|\rangle$), is used in quantum mechanics for describing quantum states. The qubit represents a state in the simplest possible quantum system. Two examples of such systems in real quantum computers are the vertical and horizontal

polarization of light and the up and down spin of electrons. For each case, the principle is the same: there are two levels to the system between which the quantum state can be measured.

The qubit can be represented with a three-dimensional unit sphere (usually named as the *bloch sphere*), which is a geometrical representation of the pure state space of a two-level quantum mechanical system.

When the quantum computer measures the value of the quantum bit, it forces it to collapse out of superposition into a classical value (e.g., '0' or '1'). It is impossible to pause the execution and observe the values of qubits while the programs run because the measurement disturbs the values of the variables.

An essential aspect of qubits is that they cannot be duplicated, giving rise to the no-cloning theorem. This theorem states that there is no quantum gate acting on two arbitrary qubits such that the state of the first qubit is copied onto the other, which in essence, means that qubits cannot be duplicated.

2.3 Measurement

Measurement and gate transformations are operations that can be performed on qubits. Applying a gate to a qubit transforms its state, changing the qubit probabilities to collapse to either $|0\rangle$ or $|1\rangle$. The measurement operation is done by projecting the mathematical representation of a qubit onto one of the basis vectors. The qubit, which is the vector that describes the quantum state, then collapses to its projection onto any of the base vectors depending on its associated probability.

The measurement operation involves physically checking the value of some property of a quantum system. An example can be the direction of a particle's spin. The quantum state is in a superposition of spin-up and spin-down states before measurement, each with a corresponding amplitude. After measurement, the state collapses to either up or down-spin with a probability according to its amplitudes.

The measurement either takes the value zero if the qubit is measured in the state $|0\rangle$, and the value one if the qubit is measured in the state $|1\rangle$. So, it is crucial to remember that measurement is an irreversible operation, destroying quantum information and converting it into classical information.

2.4 Entanglement of States

An entangled state of a composite system can be defined as a state that cannot be written as a product state of the component systems. Because of the relationship between logical and physical states, a quantum register that contains more than one qubit cannot be described by merely listing each qubit state. The state of individual qubits can be entangled

together; as more qubits come into play in a quantum computer, the number of states that information can be in increases exponentially. For instance, a two qubits system can take on some values, along with superpositions among these values; also, the two qubits can even be in a state of entanglement where the two cannot be treated as independent pieces of information. A system with three qubits has potential superpositions of eight states. This exponential growth of possible values can show the power that quantum computing has.

2.5 Programming Languages

A program needs to be implemented using a process that its execution can be simulated physically. The physical mechanism that is used to simulate program executions is the computer. A programming language determines how the resources available can be used to build programs so that they can be simulated on computers. Languages are a set of resources that can be put together to build specific programs, plus a set of composition rules that ensure that all programs can be implemented on computers.

Programming languages [7, 8, 9, 10] are designed as formal languages that, through a series of instructions, permit a developer to write a set of orders, sequential actions, data, and algorithms to build programs that control a machine's physical and logical behavior. In other words, a programming language is a structured interaction system composed of sets of symbols, keywords, and semantic and syntactic rules that allow an understanding between a developer and a machine.

Programming languages are the basis for building applications used in everyday life, and they can be classified into two main types: low-level and high-level language [9]. Low-level programming languages are built to be entirely machine-oriented languages. These languages are used as an interface that links hardware and software and directly control the equipment and its physical structure. To apply it correctly, the developer must know the hardware very well. Some examples of low-level programming languages are machine and assembly languages. High-level programming language aims to help the developer's job, as they use instructions that are much easier to understand and allows them to write code using a more understandable human language.

2.6 Summary

In this chapter, we establish the context of the research. Quantum computing is a type of computing that is based on the principles of superposition and quantum entanglement. Superposition is the ability of a quantum system to be in multiple states at the same time until it is measured. Quantum entanglement is the physical phenomenon that occurs when a group of particles is generated, interacts, or shares spatial proximity in a way such that

the quantum state of each particle of the group cannot be described independently of the state of the others, including when a large distance separates the particles. Finally, we describe the concept of a programming language to link quantum computing and programming languages.

Chapter 3

Related Work

3.1 Classical Programming Languages

According to the online historical encyclopedia of programming languages HOPL [11], nearly 9000 languages have been created since the 18th century. However, according to GitHub [12] only about 370 are still active.

According to Lagutin [13], technological evolution is one of the reasons that many programming languages were created, considering that with technological advancement, we need new tools to develop new systems for these technologies. The program can be so unique that to create a solution for it, researchers and companies have to create a new language to develop it. Another point is that different types of developer jobs need different languages, as there are different types of software and platforms, and they may require their own tools and resources. Also, some programming languages have different needs and goals and are better suited for certain types of tasks than others. Each programming language has certain features and characteristics that make it suitable for specific tasks.

In Sherman's stack overflow post [14], there are four primary points that could answer how programming languages are being used and why they were created. The first point is that different tools are needed for different jobs, e.g., Ruby is a very popular language for developing websites, and R is very popular in statistics. Second, every developer has different tastes. As programming languages are used for humans to express ideas to computers, it is only natural that a developer might like to use a specific language for specific reasons. Third, a language can be used because if it was the company's choice based on what the individuals who work there know best. For example, C# is mainly used on Stack Overflow because it was the primary language used by the founders. Fourth, variety is a strength, there are many programming languages out there because proposing, implementing, and distributing a new one is *easy and cheap*.

Hope [15] proposed a list of significant points about programming languages:

- **Readability and maintainability** of some languages are easier to learn than others.

- **Performance** different languages have different performances, e.g., some languages are interpreted, and others are compiled, resulting in different execution times.
- **Specific use cases** languages can be specialized for different types of programs.
- **Ease of prototyping** the developers can start writing code faster in languages that allow rapid prototyping.
- **Available libraries** in different languages provide libraries with standard functions that allow faster development.
- **Security** some languages are more secure than others. For example, C is well known for having many vulnerabilities.
- **Community support** developers prefer languages with more community support to help them.
- **Expressiveness** developers tend to prefer languages they are more familiar with and feel comfortable with.

Scott's [16], the main reasons for the variety of programming languages are evolution, how to learn better ways of doing things, economic advantage factors, such as commercial and industrial, hardware, and unique purpose orientation; and the diverse ideas of what developers most like to use.

3.2 Towards a Quantum Programming Language by Selinger (2004)

According to Selinger [17], quantum algorithms are often expressed at the hardware level, such as in the quantum circuit model or quantum Turing machines. Structured programming or abstractions such as data types are not encouraged by these methods. Selinger [17] proposed the design of a quantum programming language, called QPL, defining the syntax and semantics of a functional quantum programming language with characteristics such as loops, recursive procedures, and structured data types. The language has some essential characteristics, for example, it is statically typed, and the author guarantees as it is a functional language that any well-type program does not have run-time errors. In terms of super operators completing partial orders, it has denotational semantics.

Selinger [17] work proposed a point of view where quantum computing is expressed with data and control flow and does not rely on any specific hardware model. The control stage of the program is classical, but the information manipulated by the programs can have quantum superposition. In the language proposed, there is no notion of quantum branching and the superposition of two distinct statements because even if the manipulated data involves quantum superposition, the control state is classical. The author used the slogan "quantum date, classical control" for the language.

The author reviewed some basic concepts from linear algebra and quantum computation, presented a view of quantum programming language in terms of flow charts, and presented it in its formal semantics that shows a syntax more textually for quantum programs. This textual semantics is also more structured in terms of structured programming languages such as Pascal. Selinger also proposed possible extensions to QPL.

3.3 A Brief Survey of Quantum Programming Languages by Selinger (2004)

Selinger [18] provided a brief review of quantum programming language research. Some quantum virtual hardware models are described, such as the quantum circuit model made up of quantum gates in the same way a classical logic circuit is made up of logic gates. The model emphasizes the unitary transformations with measurements carried out as the very last step in a computation. Another model that Selinger summarized is the QRAM model of Knill [19], which permits unitary transformations and measurements. In this model, a quantum device is controlled by a universal classical computer and contains addressable qubits, like a memory in a classical computer. The model contains a classical controller that sends a sequence of instructions and a quantum device that passes out these instructions. A third virtual hardware model is the quantum Turing machine; in this model, the entire operation of the machine is assumed to be unitary, and measurements are never performed.

The author briefly commented on some semantic projects, such as Girard [20] with the definition of coherent quantum spaces as possible semantics for higher-order quantum computation. Abramsky and Coecke [21], which models a high-order function and applications that rely on entanglement and quantum measurement. Edalat [22] with a domain-theoretic interpretation of Gleason's theorem and Coecke and Martin [23] with a domain-theoretic treatment of the von Neumann entropy of a quantum state.

Selinger [18] further raised three challenges for quantum programming languages; the first is a denotational semantics for a higher-order quantum programming language; the second is a theory of quantum concurrency, as the network of quantum processes that exchanges classical and quantum data can be a challenge. The third one is the development of quantum programming languages on imperfect hardware. In real hardware implementations, random errors and decoherence can be predicted, and the challenge is the extent to which known error detection techniques can be automated.

3.4 Quantum Programming Language by Unruh (2006)

Unruh [24] investigated the development of quantum programming languages and gave an overview of the current work. Quantum programming languages are divided into two

types: the first one that targets practical application and the second one that targets the theoretical analysis of quantum programs.

For practical programming languages, including simulation or programming in quantum computers, there are several possible features of quantum programming languages and some essential features that a language should have from a developer's point of view.

- **Simple and powerful**. Simple means that the language is easy to understand, and no great effort is needed before using it. Powerful means that it possesses the features necessary so the developer can concentrate their work on the algorithmic.
- **Technology independent**. The language should be able to translate the code to a sequence of instructions in a way that the code is not written depending on this technology.
- **Transparently implement optimization and Error correction techniques**. The language should handle optimization and error corrections transparently from the developer.
- **Using simulators**. The developers should be able to run the quantum programs using simulators on a classical computer, making the programs easy to test and debug.

For the formal programming languages, Unruh [24] separated the quantum programming languages by the syntax and semantics. The author also presented possible features of quantum programming languages, such as:

- **Quantum branching**. The problem of branching in quantum languages is that the value of a qubit in a superposition may be used in the branch conditions. In this case, the measurement would destroy the superposition. However, the CNOT-gate is a simple example of quantum branching because it flips the value if the other has value 1.
- **Continuous classical output**. Some quantum algorithms may require that the output is given before the program's termination, even if most algorithms take the inputs and return the output after its execution.
- **Concurrent processes**. The language needs to be able to interact with concurrent processes.
- **Infinite data types**. Most of the languages proposed have data types only for the finite-dimensional Hibert space. Moreover, thus, elementary data such as integers cannot be represented in this model. In classical machines, integers are also finite data types, but in designing a new algorithm, it might be necessary to use unlimited integers in the development stage.
- **Higher-order data types**. Complex quantum data types, like lists, tuples, and records, can be required for the future development of quantum algorithms.

- **Powerful reasoning about programs**. The language must have a collection of rules that allows the developers to focus on the algorithms instead of the specific details of the language.

3.5 Quantum Programming Language, Survey and Bibliography by Gay (2006)

Gay [25] briefly summarized the basic concepts of quantum computing, surveyed the literature on quantum programming languages, and classified the papers studied according to the central theme of each paper. The classification proposed is the following: programming language design, semantics, and compilation.

For the programming language design, Gay divided into imperative languages, functional languages, and λ-calculi and other language paradigms. In the imperative languages, he reviewed the quantum Turing Machines of Deutch [26] as the first model for general quantum computation, which has the property of superposition of machine states. Gay also described the work of Knill [19] that defined a proposal for a formalized quantum programming language that implements an imperative pseudo-code on a quantum random-access machine (QRAM). The QRAM model consists of registers that can execute quantum operations. The author also summarized other imperative quantum programming languages as QCL [27], Betteli et al. [28], and qGCL [29].

For the functional languages and Lambda-Calculi, the author summarized both extensions of λ-calculus of Maymin [30], the quantum λ-calculus of Van Tonder [31] and also the work on the definition of the first-order functional programming language QML of Altenkich et al. [32]. He also mentions several works that investigated quantum programming within Haskell.

For the other languages paradigms, he summarized the work of Gay et al. [33], the definition of the process calculus CQP (Communication Quantum Process), and Jorrand et al. [34], the definition of QPAlg (Quantum Process Algebra). The two languages describe systems combining classical and quantum computation and communication, and both aim to support the formal specification and verification of quantum cryptography protocols.

In the semantics classification, Gay referred to denotation techniques as many papers emphasizing language design also defined semantics in an operational style. The papers which do not define languages (for example, the semantic studies that focus on protocols) and for papers including language definitions but whose emphasis is on denotational semantics. He also included papers that apply linear logic to the structural aspects of quantum computation.

For compilation classification, Gay summarized some quantum compilers that can be used to compile quantum programming languages. The work of Altenkirch et al. [32] have developed a compiler for their QML language into a representation of quantum circuits,

using categorical semantics as an intermediate form.

3.6 A Survey of Quantum Programming Languages: History, Methods, and Tools by Sofge (2008)

Sofge [35] researched some of the essential quantum programming languages in terms of their history, methods, and proposed tools. He also mentions Feynman's proposal in 1982 to build a quantum computer to simulate quantum systems. Making this simulation in a classical computer would require the exponential use of resources in terms of memory and computational time. He also described the work in linear logic by Girard [36] in 1987, which played an essential role in designing quantum programming languages, particularly those based on lambda calculus.

According to Donald, the first step to creating a quantum programming language was a work from Knill in 1996, which defines a quantum random-access machine model (QRAM). According to the author, the proposal described by Knill does not have all the necessary characteristics to be considered a quantum programming language because of the informal definition of its structure and also the lack of strong typing, and because it does not represent some of the necessary quantum properties. Sofge defined a proposal for a taxonomy to represent quantum programming languages, grouping them into three different types: imperative, functional, and other quantum programming languages (which include mathematical formalism that was not defined to run on computers).

The author defined some challenges in quantum programming languages as the lack of quantum computing hardware to execute quantum algorithms, the lack of a definition regarding the data structures that need to be implemented, and the operations on top of this data structure. Which operations should be allowed and which should not, and more information on how to better design a quantum language to better use quantum computing. Another challenge is that quantum mechanics is still incomplete and, by extension, the theory concerning quantum computing. There is still much research going on to understand the physics behind quantum computing better.

3.7 The Modern State of Quantum Programming Language by Rojas (2019)

Rojas [37] summarized the core ideals found in a successful quantum programming language as proposed by the community of authors and developers. Also, a high-level look into domain-specific quantum languages and their different expectations. He mentioned some important designs that a quantum programming language should have, like completeness, expressivity, efficiency, and hardware independence.

The author also provided a table with a historical overview of the development of quantum programming languages. It shares the conclusion that it is necessary for much more time to be spent focusing on handling quantum error correction within quantum programs and designing a software framework that can accommodate the various physical constraints of quantum devices. With so many languages to choose from and build off, it is hard to pick a single language with the optimal design format for developers and quantum computers.

3.8 Quantum Programming Languages: A Systematic Review of Research Topic and Top Cited Languages by Garhwal et al. (2019)

Garhwal et al. [38] gave an overview of the state of the art in the field of quantum programming languages and focused on actual high-level quantum programming languages, their features, and comparisons. The author also discussed some research questions (e.g., what are the different types of Quantum Programming Languages; what are the recent trends in the development of quantum programming languages; which major Companies, Groups, Institutes, and Universities are working on developing new quantum languages; what are the most popular publication venues for quantum programming; and what are the most cited papers in the area of a quantum programming language).

The authors divided the types of quantum programming languages into multi-paradigm, imperative, functional, quantum circuit language, and quantum object Language. The authors described various types of quantum imperative and quantum functional languages, respectively. They also showed the year-wise distribution of papers considered in their review process, showing that in the recent 4 to 5 years, significant progress was made in developing new quantum programming languages. However, many research groups are working in the area of quantum programming languages. Garhwal et al. [38] highlighted the following groups: Advanced Research Projects Activity (IARPA); Quantum Architectures and Computation Group (QuArC) at Microsoft; The University of Nottingham QML Harvard University; Center for Quantum Software and Information at the University of Technology, Sydney, Australia.

They stated that most of the researchers in quantum programming language prefer to publish on arxiv.org. The top three quantum languages in terms of the number of citations for the main paper published for quantum languages as per Google and as per Web of Science based on his study are QFC/QPL, Quantum Lambda Calculus, and Q.

3.9 Software Engineering for Quantum Programming: How Far Are We? by De Stefano et al. (2022)

De Stefano et al. [39] researched GitHub repositories to make code analyses to create a taxonomy of the quantum technologies that are used and also conducted a survey with developers that uses quantum programming languages to get their opinion on the current adoption and challenges of the quantum programming field. The authors analyzed the GitHub repositories that use one of these three quantum programming frameworks and languages: Qiskit, Cirq, and Q#, because they stated that they are the most widely used. The survey proposed has three sections:

- Background, which aims to get data about the background of the participants.
- Current Adoption, which aims to get the participant's opinions on the use of quantum programming, focuses on the use of Qiskit, Cirq, and Q#, and the context of using, i.e., academic, research, industry.
- Potential Adoption and Challenges, which aims to get the developer's opinion on the challenges in quantum computing technology.

The author proposed to respond to two research questions: what extent and for what purposes are quantum programming frameworks being used; and what are the main challenges that quantum developers are experiencing when interacting with quantum frameworks. For the first research question, according to the survey, most of the quantum programming technology is being used for personal study purposes (41%), and significant effort is put into the framework and research repositories. For the second research question, the main challenges related to quantum frameworks are the comprehension of quantum programs and difficulty setting up hardware and software infrastructures.

3.10 Summary

In this chapter, we surveyed the literature that is most related to our main contributions or that is in line with the research topics studied in this thesis. In [18], Selinger provided a brief review of quantum programming language research. In [24, 35, 37], the authors provided an overview of the development of quantum programming languages. In [39], De Stefano et al. researched GitHub repositories to create a taxonomy of the quantum technologies and investigate the most used quantum programming framework.

Chapter 4

Quantum Programming Languages

Quantum programming languages are responsible for translating algorithms developed by developers into instructions that a quantum computer can execute. They are intended to facilitate the development of quantum algorithms by providing high-level languages with quantum data types, operators, and functions. They are crucial for making it possible to perform quantum computer programming. These languages control physical devices, estimate the computational costs of running programs using simulators and verify and implement quantum algorithms. The languages are also used by many professionals, such as researchers, developers, and experienced professionals, to apply quantum computing concepts to solve real-world problems and make innovative discoveries.

To find the quantum programming languages that have been proposed either by researchers (e.g., on research papers) or companies, we searched for the following keywords: quantum programming languages, quantum languages, and quantum programming in `https://scholar.google.com`, `https://google.com`, and `https://arxiv.org`. We limited our search to 2021 when our study was conducted.

Figure 4.1 shows a classification of some quantum computing languages according to the level of programming, and Figure 4.2 traces the roots of quantum computing languages through the ages.

Table 4.1 reports a summary of the quantum programming languages. It summarizes some information, such as the year they were created, whether they are open-source, active, academic, or industrial, the author, and type (imperative, declarative or multi-paradigm). The classification, if they were active or not, was based on the contribution or support on their page (e.g., GitHub for the open-source). If it passes two years without issues/comments and new language changes, they are considered inactive for the open-source. The non-open source is considered inactive when no more support is given by searching on the website of the company that created the language. For the columns, without an answer, it means that it was not possible to confirm if they are open source or active.

Several researchers, like Sofge [35], divided quantum programming languages into

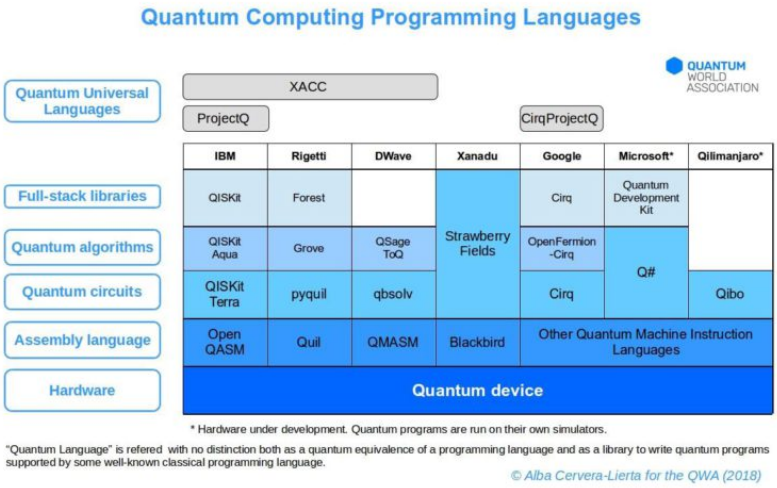

Figure 4.1: Classification of quantum programming languages according to their programming level. Figure taken from [40].

imperative, declarative (functional), and other paradigms. In this work, we used the exact taxonomy of quantum programming languages.

4.1 Imperative Quantum Programming Languages

The imperative programming paradigm describes computation as instructions, actions, or commands that are carried out step-by-step and change the state or variables of a program to achieve the desired result. Some of the best-known imperative programming languages are C, Python, Pascal, and others. As a common characteristic among them, the languages visualize a program in terms of a sequence of operations. When executing each instruction, it updates the global state of the system. Quantum programming languages have these same imperative paradigm features as classical languages and can be compiled or interpreted using the virtual hardware model QRAM. In the following sections, we describe the imperative quantum programming languages.

4.1.1 QCL

QCL (Quantum Computation Language) [27] was the first quantum programming language created. It was developed and improved by Bernhard Ömer between 1998 and 2003. The language allows for the simulation and implementation of quantum algorithms and is independent of the high-level architecture of computers. The language's syntax is based on classical programming C and Pascal and has a coherent formalism. Among the main features of QCL, we can highlight flow control, functions, classical data types, quantum data types (qubit registers), quantum operators, functions to manipulate quantum registers, and quantum memory management, among others.

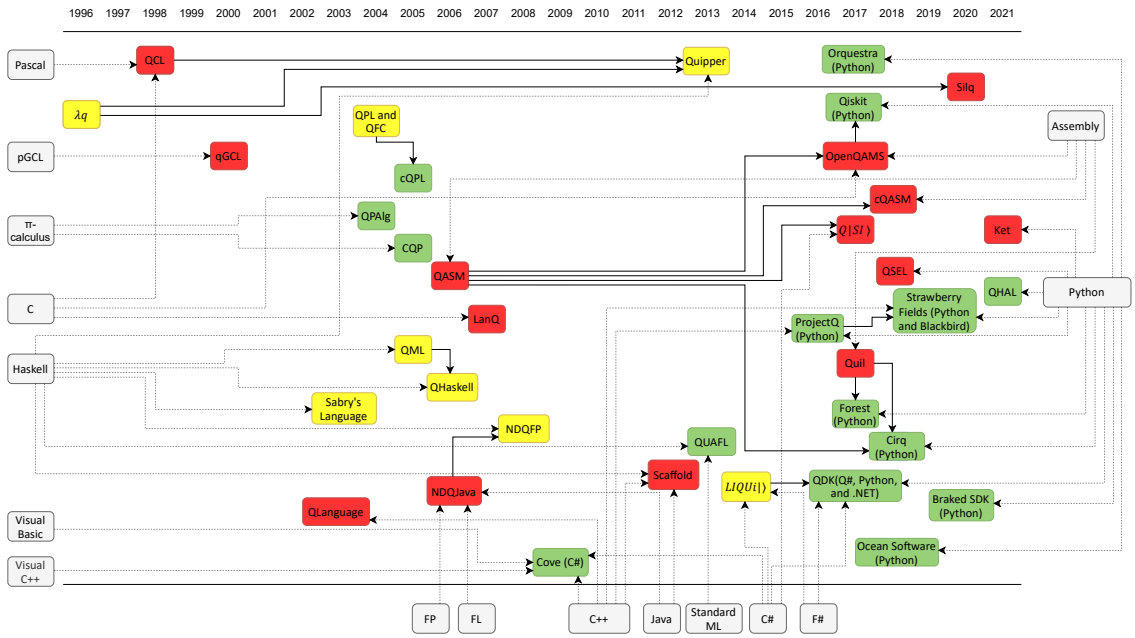

Figure 4.2: Evolution of quantum computing languages.

4.1.2 QASM

QASM (Quantum Macro Assembler) [70] is a low-level quantum programming language developed in Python to be used in D-Wave's quantum computers. It aims to create an abstraction, so developers do not need to know specific hardware details; developers can use it to have high control over the hardware or high-level language compilers.

4.1.3 Silq

Silq [46] is a high-level quantum programming language, developed at ETH Zürich, and whose main features are a robust static type system, variable assignment, conditionals, generic parameters, classic types, loops, superposition, and others. It was published in 2021, written in the D language, and designed to automatically compute temporary values without inducing an implicit measurement.

4.1.4 Q Language

Bettelli presents an extension of the C++ language as a model for a high-level quantum programming language [28]. It has a set of quantum primitives and a simulator with a runtime environment to calculate and optimize quantum operations. The language has register handling, manipulation of quantum operators (like QHadamard, QFourier, QNot, and QSwap, new operators can also be defined using C++ class mechanism), and low-level primitives.

4.1.5 qGCL

The language for quantum programs specification, qGCL [29] (Quantum Guarded Command Language), presented by Sanders and Zuliani, was based on Dijkstra's Guarded Command Language. It is used to express quantum algorithms and contains resources to develop a "universal" quantum computer. It exhibits several features such as expressivity, simplicity, control structures, data structures, formal semantics, and uniform observation treatment. It also contains a new datatype with a vector from a finite-dimensional Hilbert space.

4.1.6 LanQ

Mlnarik [82] introduced in 2007 a high-level quantum programming language called LanQ. The language has a syntax similar to C, which is used to prove the correctness of quantum algorithms. The main features of the language are the possibility of combining quantum and classical calculations, communication, and parallel execution of processes.

4.1.7 $Q|SI\rangle$

$Q|SI\rangle$ [68] is a quantum programming environment embedded in the .Net language. It is an extension of while-language that includes a compiler and a suite of tools for simulation quantum programs. The language has a measurement-based case statement and a measurement-based while-loop, and these two features help developers describe large-scale quantum algorithms.

4.1.8 OpenQASM

OpenQASM [69] (open quantum assembly language) is a simple language that defines different gate sets using a mechanism to specify unitary gates. It was created with syntax with elements of C and assembly and provides instructions to quantum devices. OpenQASM is based on QASM, a language that describes quantum circuits and is used by IBM through Qiskit, which has functionality for generating OpenQASM code from a specific quantum circuit.

4.1.9 Scaffold

Scaffold [79] is a programming language for expressing quantum algorithms that compiles QASM and OpenQASM. It is very similar to the C language and facilitates the expression of quantum algorithms in data types and operations. Although Scaffold provides a coding style similar to C, it also incorporates features that make it appropriate for coding quantum algorithms. The main features provided by the language are quantum and

classical data types (e.g., quantum registers, arrays, quantum struct, and quantum union), quantum gates, loops, and control constructs.

4.1.10 cQAMS

The cQASM [51, 52] (Common Quantum Assembly Language) language describes simple circuits, ensuring interoperability between quantum compilation and simulation tools, and also aims to abstract details from qubit technology. This assembly language is based on QASM, which originated to define a quantum circuit to render images for visualization purposes. The syntax definition is also based on QASM for language standardization. The cQASM instructions can be used as input to a quantum computer simulator or a low-level compiler that generates specific hardware instructions suitable for execution by the target quantum computer.

4.1.11 Quil

The Quil [62] language is a quantum instruction language analogous to an assembly language on classical computers. It has an abstract machine architecture for quantum computers that instructs the quantum computer on which physical ports implement specific qubits. Quil was created by Smith et al. and introduced a shared quantum and classical memory model that can be used for many quantum algorithms. It has classic feedback and control and is used as an intermediate format to be a compilation target for quantum programming languages. Quil's main features are arbitrary quantum gates, measuring qubits and saving measurements in classical memory, and synchronizing the execution of classical and quantum algorithms, among others.

4.1.12 QSEL

QSEL [57] (Quantum Super Entangled Language) is a quantum programming language focused on superposition and entanglement. The language compiler was written in Python with a model where the circuits can create superpositions and entanglement. QSEL only describes three commands: superposition, entanglement, and measurement.

4.1.13 Ket

Ket [42, 43, 41] is a high-level classical-quantum programming language that provides rapid learning development and testing of quantum programs using Python constructs with the addition of quantum specifics. As a Python-embedded language, Ket offers Python types and two new quantum types, which implement an array reference of qubits and another for storing variables in a quantum computer. Ket provides a universal set of

quantum gates and a quantum measurement that does not directly return the result but rather a future variable with the promise that the value will be available when needed.

4.1.14 NDQJava

NDQJava [84] is a quantum programming language created in 2006 based on Java. The language has two parts: a classical part, just Java, and a quantum part, which has quantum components (quantum data type, quantum variable, quantum declaration, and quantum expressions). NDQJava was implemented by simulation on classical computers.

4.2 Declarative Quantum Programming Languages

Unlike the imperative in which instructions are executed step-by-step, the declarative programming paradigm describes the desired results without explicitly listing the steps that must be performed. This style of language use programming logic that depends on mathematical functions, which means that inputs are converted into outputs using mathematical logic. Among the types of declarative languages, the most used are the functional ones that do not support loops and if/else statements, unlike the imperative ones. All functions in this type of language have no side effects, and state changes are represented as functions that transform the state. Examples of classic declarative programming languages are Haskell, Lisp, Erlang, and others. In the following subsections, we describe the declarative quantum programming languages.

4.2.1 QPL and QFC

QFC and QPL are two functional quantum programming languages introduced by Selinger [17] in 2004. The main difference between the two languages is that QFC uses a syntax based on flowcharts and QPL uses a textual syntax as a base. QFC flowcharts consist of elementary building blocks having multiple input and output edges, representing the flow of program control. Loops and recursion, for example, can be represented by blocks where one of the output edges is simultaneously the input edge. In QPL, its syntactic structures are represented through textual representation. Both languages have classical control flow and can operate quantum and classical data, with unitary operations and measures safely integrated into the language. There are no runtime type checks or errors.

4.2.2 QML

Altenkirch [32] developed a functional Haskell-like quantum programming language called QML, the same as QFC and QPL from Selinger. QML is based on linear logic and supports classical and quantum operators, allowing both data and program control to be quantum. The language allows an if-then-else to be used with a classical condition or condition

that measures the qubit value. The language does not support duplication of quantum data, but two or more variables are allowed to relate to the same quantum system.

4.2.3 Sabry's Language

Sabry [88] created a functional quantum computing model embedded in Haskell to write quantum algorithms. Sabry's model differs from classical programming languages, where expressions can be grouped into introduction constructs and elimination constructs for the language's type connectives. While in the quantum model, it can only have virtual elimination constructs since the elements of an entangled data structure cannot be divided.

4.2.4 Lambda Calculi (λ_q)

Lambda calculi (λ_q) languages were defined to describe quantum algorithms, and they are based on the classic lambda calculus introduced in 1930. This type of language supports high-order computational functions, and it was first defined for quantum calculations in 1996.

In 2004, Van Tonder [31] made the first effective effort to create a functional quantum programming language, λ-calculus. This language uses quantum lambda calculus for quantum computation. It has a variable substitution rule and a function definition scheme; as a rule, any computable function can be expressed using language formalism. The only disadvantage is that it has no measurements.

In 2006, Selinger and Valiron [93], developed a λ-calculus language that is contrary to van Tonder's language, supports the measurement of a quantum program as a primitive in the language. The type of system is differentiated between the types in which the values are duplicated and those that are not. This system guarantees that violating the principles of not cloning and deleting quantum data occurs. The authors had to build a type-inference algorithm that can verify whether a particular term is capable of being identified as a particular type in the linear system type and find its type.

4.2.5 Quipper

Quipper [77] is a functional quantum programming language, published in 2013, based on Haskell. The language has the particularity of being suitable for programming physics applications and provides a high-level circuit language as well as operators for manipulating these circuits. It has libraries for quantum integer and fixed-point arithmetic manipulation. One of the language's main features is that it has all Haskell calculations and is not dependent on any specific quantum hardware.

4.2.6 NDQFP

NDQFP [81] is a modular functional quantum programming language created in 2008, where each program is composed of one or several modules. It is a language with classical and quantum data types, and there are also input/output components and an exception feature defined. The design considered languages like FP, FL, Haskell, and NDQJava, but with differences in the language overview.

4.2.7 $LIQUi|\rangle$

$LIQUi|\rangle$ [73, 74, 75, 76] is a quantum programming language created by Microsoft for quantum computing. The name stands for Language-Integrated Quantum Operations and translates quantum algorithms into low-level machine instructions for a quantum device. It is an extension of F# language and was designed to simulate complex quantum circuits in different environments. The language has many gates that can be overridden or extended and three different classes of simulators for different run times.

4.2.8 QHaskell

QHaskell [83] is a functional quantum programming language implemented in Haskell and inspired by QML that follows the "quantum data and control" paradigm. The language has a syntax for handling potentially entangled quantum data based on Haskell's arrow notation. It has the typing rules of QML with a type system that is based on linear logic to control the use of quantum variables.

4.3 Multi-paradigm and domain-specific languages

A programming language can be described as a multi-paradigm when it supports more than one different programming paradigm. The objective of a language being a multi-paradigm is to offer the developers several different styles in the same language, in which they can freely mix the styles, making it possible to develop programs more easily and efficiently. There are also quantum programming languages that are multi-paradigm and domain-specific, as shown in the examples in the following sections.

Regardless of the type of language and paradigm used to develop a quantum algorithm, an SDK (Software Development Kit) is required to create, simulate, and execute code in quantum devices. Many of these software environments are open-source and use Python programming language as a basis, e.g., Strawberry Fields, Orquestra, Cirq, and others. Some of the biggest technology companies like IMB, Google, and Microsoft are developing open-source development kits for their quantum devices. These quantum SDKs can be used in computers with quantum registers or even in simulator programs that provide an implementation of quantum gates using classical ones. They also provided

some tools for the developers to implement their quantum algorithms by accessing computers and quantum simulators. The main SDKs use classical languages such as Python or quantum programming languages explicitly developed for quantum computing, such as the language created by Microsoft, Q#.

4.3.1 QDK (Q#, Python and .NET Languages)

QDK (Quantum Development Kit) [64], developed by Microsoft, provides tools to support developers in quantum development. This quantum SDK includes libraries to help developers create quantum operations and have simulators to execute and test quantum programs that can run in several environments. The programs can use Python or .NET host programs to execute as a console application, as QDK supports interoperability with Python and other .Net languages. The quantum programming language Q# is part of the Microsoft QDK. QDK also has an integration functionality that allows developers working with Qiskit and Cirq to integrate with QDK and execute their programs on Azure Quantum, the Microsoft cloud service for quantum computing.

Q# [65, 66] is a multi-paradigm quantum programming language developed by Microsoft. It is open-source and part of the Quantum Development Kit (QDK). The language is used to implement and execute quantum algorithms, exploring quantum computing phenomena, such as superposition, entanglement, Grover's quantum algorithm, and others.

QDK provides a quantum simulator for running and testing Q# programs. In Q#, qubits are an opaque data type that refers to a two-state quantum system, and both states can be physical or logical and are used to perform quantum operations. The Q# programs describe how a classical control computer interacts with qubits rather than directly modeling the quantum state.

With Q#, the developers can implement quantum algorithms using qubits that use uncontrolled gates, Hadamard gates, and others. The language uses quantum properties for qubits like entanglement and superposition and has many quantum operations.

4.3.2 cQPL

Mauerer's [94] presents an extended version of QPL, a functional language defined by Peter Selinger, called cQPL. This language is used to support communication between distributed processes, which allows the exchange of data (classical and quantum) between an arbitrary number of members. A language compiler is also defined to generate code to be used in a quantum simulator.

4.3.3 QPAlg

QPAlg (Quantum Process Algebra) was created by Jorrand and Lalire [86] to describe the interactions between quantum and classical processes using an algebraic process ap-

proach. The processes communicate over named gates that are static and given before the process execution. The language is based on π-calculus. From a quantum point of view, some of the features of the language are variable entanglement and management; unitary operations; measurement and probabilistic processes; and communication and physical transport of qubits to classical or quantum systems.

4.3.4 CQP

CQP [33] (Communicating Quantum Processes) is a quantum process algebra like QPAlg, defined by Gay and Nagarajan. The language's syntax is based on an expression language and π-calculus. CQP was created for modeling the communication of classical and quantum processes. It has a static type and formal operational semantics to transmit a qubit using a communication channel. The language can be used to analyze and verify quantum protocols.

4.3.5 QualFL

QualFL, created by Lapets et al., is a type of domain-specific quantum programming language (not designed for a general purpose) that have as targets physicists and mathematicians to work on quantum algorithms by focusing on the abstract description of quantum computation. It can be compiled into logical quantum circuits and defines superposition and unitary transformations on data.

4.3.6 QHAL

QHAL [44] is a hardware abstraction layer for quantum computers created by Riverlane to be a universal quantum language. The main object was to define a multi-level hardware abstraction layer (HAL) to build software portable across platforms and allows developers to abstract the hardware implementation by providing a set of commands which could be implemented on most quantum devices. The main features of the language are: define a multi-level HAL; be portable with minimal loss of performance; have typical features; and minimum hardware-dependent features and have support to advanced features, like optimization, measurement-based control, and error correction.

4.3.7 QISKIT (Python)

Qiskit [67], founded by IMB, is an open-source SDK (software development kit) used to perform quantum computations using the main properties of quantum mechanical principles at the level of quantum circuits. Qiskit uses Python programming language and allows the developers to use their tool to create quantum programs and execute them on

a quantum device on IBM Quantum Experience, an online platform to access the cloud-based quantum computing of IBM.

4.3.8 Cirq (Python)

Cirq [55, 56] is an open-source framework developed by Google AI Quantum Team, which is a Python library for manipulating quantum circuits. The framework provides valuable hardware abstractions, where the developers can run their codes on quantum computers and simulators. The developers can build quantum circuits from gates acting on qubits. Cirq has built-in simulators, such as a wave function simulator called qsim. Google provides a quantum computer service to run experiments in their quantum processors using Cirq.

4.3.9 Braket SDK (Python)

Amazon Braket [48, 49] is an open-source Python library with a fully managed quantum computing service. The Braket SDK provides tools to build, test, and run quantum algorithms on AWS. It can be used to design and build quantum circuits and send them as quantum tasks to Amazon Braket devices. The framework has two types of simulators, a fully managed one available through Amazon services and a local simulator within the SDK. The main features are:

- Hardware-independent developer framework to simplify the process of designing and running quantum algorithms.
- Fully managed runs of classical-quantum algorithms with hybrid jobs.
- Fully managed Jupyter notebooks to build quantum algorithms and manage experiments.
- Use quantum processing units from different vendors such as IonQ, Rigetti, or D-Wave.

4.3.10 Strawberry Fields (Blackbird and Python)

Strawberry field [50] is an open-source, cross-platform Python library developed by Xanadu to simulate and run quantum programs. The platform has three main components: an API for quantum programming based on the Blackbird quantum programming language, three virtual quantum computer backends, and an engine that can compile Blackbird quantum programs on many different backends. The main features of this platform are:

- Integration with Xanadu Quantum Cloud, where developers can submit their quantum programs to run on Xanadu's photonic hardware.
- High-level functions to aid in the development of quantum programs.
- A simulator for photonic algorithms.

Blackbird [50] is a quantum assembly language for basic continuous variables states, gates, and measures. The Strawberry Fields framework uses it and is designed to represent continuous-variable quantum programs that can run on photonic quantum hardware. The Blackbird language is built into Strawberry Fields but also exists as a separate Python package. Blackbird has four types of operations (state preparation, port application, metering, and subsystem addition and removal).

4.3.11 Forest (Python)

Forest [60, 61] is a quantum software framework developed by Rigetti. The Forest suite includes a QUIL compiler (quilc), a quantum virtual machine (qvm), and pyQuil, an open-source Python library, for constructing, analyzing, and running quantum programs. The pyQuil library is built on top of Quil, a quantum instruction language explicitly designed for near-term quantum computers and based on a shared classical/quantum memory model, which means that the memory has both qubits and classical bits. The main pyQuil functions are: generating Quil programs from quantum gates, classical operations Compiling and simulating Quil programs, and the Quantum Virtual Machine to execute Quil programs on real quantum processors.

4.3.12 DWave Ocean (Python)

Ocean software development kit (SDK) [53, 54] in an open-source framework written in Python developed by D-Wave. It is used for developing quantum applications to run in the D-Wave quantum computer. D-Wave provides a quantum cloud service where the developers can submit problems to their quantum computers using Ocean's framework. Ocean provides packages for quadratic models, building hybrid solvers, simulated annealing samplers, maps constraints to binary quadratic models, and others.

4.3.13 Orquestra (Python)

Orquestra [58, 59] is a framework developed by Zapata Computing. It is a unified workflow-based toolset for quantum computing. It enables developers to build and run quantum workflows across multiple quantum and classical devices in a unified quantum operating environment. Orchestra is based on Python code and libraries and integrates with many vendors, e.g., Qiskit (IBM), Amazon Braket, IonQ, Rigetti, Cirq (Google), D-Wave, and others.

4.3.14 Cove (C#)

Cove [80] is a software framework that allows quantum computing to be performed using a classical language. It is an object-oriented framework implemented in C# that targets

commercial software developers. Cove has two main components: interfaces that specify what must be provided to program quantum computers and implementations that determine how. Cove is designed to be independent of quantum hardware and for users not to worry about error correction.

4.3.15 ProjectQ (Python)

ProjectQ [72] is a framework for quantum computing that allows developers to implement quantum algorithms using Python. This open-source framework was started at ETH Zurich. It can translate quantum programs to many backends such as IBM Quantum Experience chip, AQT devices, AWS Braket, or devices provided by the IonQ service. The main features that ProjectQ offers are:

- Developers can use Python, a high-level language, to write quantum programs.
- Users can implement their own compiler engine.
- Many backends such as simulator, emulator, resource counter, drawing engine, and command printer.
- A library to help developers solve fermionic problems.

4.4 Summary

In this chapter, we identified and described several quantum programming languages that have been proposed since 1996, summarizing the characteristics and main features of the languages, the year they were created, whether they are open-source, active, academic, or industrial, the author, and type (imperative, declarative or multiparadigm). We also showed the evolution of quantum programming languages and how they are related to other quantum and classical programming languages.

Name	Year	Open source	Active	Academic or Industrial	Author	Type
Ket [41, 42, 43]	2021	Yes	Yes	Academic	Rosa et al.	Imperative
QHAL [44, 45]	2021	Yes	No	Industrial	Riverlane	Multi-paradigm
Silq [46, 47]	2020	Yes	Yes	Academic	Bichsel et al.	Imperative
Braket SDK (Python) [48, 49]	2020	Yes	Yes	Industrial	Amazon	Multi-paradigm
Strawberry Fields (Python and Blackbird) [50]	2019	Yes	Yes	Industrial	Killoran et al. (Xanadu)	Multi-paradigm
cQASM [51, 52]	2018	Yes	Yes	Academic	Khammassi et al.	Imperative
Ocean Software (Python) [53, 54]	2018	Yes	Yes	Industrial	D-Wave	Multi-paradigm
Cirq (Python) [55, 56]	2018	Yes	Yes	Industrial	Google AI Quantum Team	Multi-paradigm
QSEL [57]	2018	Yes	No	Academic	Bacon	Imperative
Orquestra (Python) [58, 59]	2017	Yes	Yes	Industrial	Zapata	Multi-paradigm
Forest (Python) [60, 61]	2017	Yes	Yes	Industrial	Rigetti	Multi-paradigm
Quil [62]	2017	Yes	Yes	Industrial	Smith et al.	Imperative
QDK (Q#, Python and .NET Languages) [63, 64, 65, 66]	2017	Yes	Yes	Industrial	Microsoft	Multi-paradigm
Qiskit (Python) [67]	2017	Yes	Yes	Industrial	IBM	Multi-paradigm
$Q\|SI\rangle$ [68]	2017	-	-	Academic	Duan et al.	Imperative
OpenQASM [69]	2017	Yes	Yes	Industrial	Bishop et al.	Imperative
QASM [70, 71]	2016	Yes	Yes	Academic	Pakin	Imperative
ProjectQ (Python) [72]	2016	Yes	Yes	Academic	Haïner et al.	Multi-paradigm
$LIQUi\|\rangle$ [73, 74, 75, 76]	2014	Yes	No	Industrial	Wecker et al.	Functional
Quipper [77]	2013	Yes	No	Academic	Green et al.	Functional
QUAFL [78]	2013	-	-	Academic	Lapets et al.	Multi-paradigm
Scaffold [79]	2012	Yes	Yes	Industrial	Abhari et al.	Imperative
Cove (C#) [80]	2009	Yes	No	Academic	Purkeypile	Multi-paradigm
NDQFP [81]	2008	No	No	Academic	Xu et al.	Declarative
LanQ [82]	2007	Yes	No	Academic	Mlnarík	Imperative
QHaskell [83]	2006	-	-	Academic	Vizzotto et al.	Functional
NDQJava [84]	2006	No	No	Academic	Xu et al.	Imperative
cQPL [85]	2005	Yes	No	Academic	Mauerer	Multi-paradigm
QML [32]	2005	Yes	No	Academic	Altenkirch et al.	Functional
CQP [33]	2005	-	-	Academic	Gay et al.	Multi-paradigm
QPAlg [86]	2004	-	-	Academic	Jorrand et al.	Multi-paradigm
QPL and QFC [17]	2004	-	-	Academic	Selinger	Functional
Q Language [87]	2003	Yes	No	Academic	Bettelli et al.	Imperative
Sabry's Language [88]	2003	-	-	Academic	Sabry	Functional
qGCL [89]	2000	-	-	Academic	Sanders et al.	Imperative
QCL [27, 90, 91]	1998	Yes	No	Academic	Ömer et al.	Imperative
λ_q [92]	1996	-	-	Academic	Maymim et al.	Functional

Table 4.1: Quantum programming languages, ordered descending by year, that have been proposed by others.

Chapter 5

Methodology

In this study, we investigated the following research questions:

RQ1: Who is using quantum programming languages?
It aims to profile the participants who use quantum programming languages and find out in which field of study they work.

RQ2: How are quantum programming languages being used?
It aims to identify how the participants use quantum programming languages and how languages are being used in practice (e.g., to work, study, and research).

RQ3: What are the most used quantum programming languages? Why?
In this research question, we aim to obtain an overview of the most used quantum programming languages and classify the most used quantum programming paradigm. Based on the answers in the survey, we categorized the languages by the ones the developers claim to use the most.

RQ4: What makes a person choose a quantum programming language? Why?
It aims to identify which functionalities and characteristics participants are most interested in and what makes them choose to use one language over the others by analyzing what they prefer the most in the languages they use.

RQ5: Which of the studied quantum programming languages have the best chance of being imposed over the rest?
It aims to find out which language is more likely to stand out over the others and the reasons that lead the language to impose itself on others.

RQ6: What makes someone propose a new quantum programming language?
It aims to discover the reasons the participants use to propose and create a new quantum programming language.

RQ7: Are there too many quantum programming languages?
It aims to identify the participants' opinions concerning the number of quantum programming languages created and why.

5.1 Questionnaire Platforms

We conducted a study to identify the best questionnaire platform to conduct then our survey with quantum developers and researchers (more information in Section 5.4.

We identified six platforms SoGoSurvey [95], Google Forms [96], Survio [97], MindMiners [98], Typeform [99] and SurveyMonkey [100]. All are free to use or with a limited free license. These platforms have the main features needed to prepare a survey, such as usability, practical data collection, the possibility to add collaborators, add images and videos, integrate with emails and websites, insert themes, and viewer-collected data. The only platform adherent with all the features mentioned and no limitation on the number of questions, answers, and collaborators was the Google Forms platform, which is totally free. For these reasons, we elected Google Forms for our survey.

5.2 Questionnaire Guidelines

The paper Survey Methods: Questionnaires and Interviews [101] presented some steps for the elaboration of a questionnaire, such as: setting the goals, deciding on the target population and sample size, determining the questions, pre-test the survey, testing the questions, conducting the survey, analyze the collected data and produce the report. Others [102, 103] have proposed methods to conduct a survey with human developers.

Dalati and Gómez [102] presented the advantages and disadvantages of a self-administered survey questionnaire and presented the following 6-W Model to design and questionnaire:

- What is the appropriate type of scale required to perform the research analysis?
- What type of response strategy is required (structured or unstructured)?
- What type of communication approach is required (disguised or non-disguised)?
- What is the unit of analysis for which the questionnaire is designed?
- What questionnaire sequence should the questions be arranged?
- What questionnaire layout is required to accomplish research objectives?

Regmi et al. [103] showed a methodological aspect of online questionnaire surveys and some ethical concerns, privacy, and confidentiality that must be considered.

These guidelines that were studied are essential and helped us define and prepare all stages of the process of the elaboration and execution of the survey, taking into account how to define and who were the target population of the survey, how to prepare the questions and to perform a pre-test to validate the format and questions of the survey.

5.3 Programming Languages Survey Examples

Some important surveys about programming languages were studied to see what questions are used in these surveys, e.g., Stackoverflow Annual Developer Survey [104]. Stack Overflow is one of the largest professional software developers' communities; their survey contains nearly 80,000 responses from over 180 countries. It examines all aspects of the developer experience, from career satisfaction and job search to education and opinions on open-source software. Jetbrains' annual report [105] has reported that 31,743 developers from 183 countries or regions answer this survey. The Jetbrains' survey has found the latest trends in the tech industry and facts about tools, technologies, programming languages, and other facets of the programming world. Other web pages that proposed questions about programming languages were also studied as Computer Science.org [106] and Increment.com [107], to identify the most used questions about programming language surveys.

The survey examples studied help us identify the main questions about programming languages and the type and scale of the predefined answers to the proposed questions.

5.4 Our survey

In this thesis, we conducted a survey with questions about quantum programming languages to help understand the basis of quantum language uses.

5.4.1 Structure

The questions were divided into six groups:

- **Group 1**, describes a small introduction about the questionnaire, the scope, the confidentiality agreement, the estimated duration, and assesses if the participant ever used any Quantum Programming Languages.
- **Group 2**, summarizes demographic data about the participants, like age, place where they live, gender, and ethnicity.
- **Group 3**, summarizes information about the education and experience of the participants.
- **Group 4**, questions about the quantum programming languages the participant uses.
- **Group 5**, questions about the usage and tools for quantum programming languages.
- **Group 6**, wrap-up group where the participants answer questions related to the field of quantum programming languages.

Table A.1 describes, by group, the questions that were asked in the survey, the reason for each question, and the type/domain of each answer.

5.4.2 How the survey was conducted

The survey was published on specific channels of social networks about Quantum Computing, such as in groups on Facebook, LinkedIn, discord, and Twitter. The publication was also carried out in other online places, such as Slack and in mailing lists. It was also emailed to authors of papers on quantum programming languages and other researchers in the field of quantum computing. Table A.2 shows the information about the social networks contacted for the survey.

To obtain more answers for the survey, a mining was performed on GitHub to find developers of quantum programs. For this, a piece of code written in Python was implemented using a library called PyGitHub [108], a Python client library for the GitHub REST APIs. This library was used to search a list of repositories with topics related to quantum computing (e.g., quantum computing, quantum algorithms, quantum programs, and quantum programming languages). Afterward, we extract the public mailing list of their developers.

This survey was conducted in two phases. The first phase was done with 2 participants to pre-test the survey, validate the questions and calibrate the estimated time for answering this survey. After the first phase, some corrections were necessary, and the survey was updated with the lessons learned in the first phase. The survey was released and conducted with 251 participants in the second phase. The survey was open for 60 days, and one reminder was sent after ten days

After 60 days, the survey was closed, and the data collected was used for analyses and to help answer our research questions.

5.4.3 Data Analysis

The data analysis of this survey was made after we collected all data from the question answers. The first question, “Have you ever used any Quantum Programming Language?” is used to assess if the participant is familiar with quantum programming languages and enables the participant to go to all of the questions from the survey. We analyzed the answers to the survey questions to answer each research question.

Questions 2 to 13 were analyzed to identify the profile of those using quantum programming languages to answer RQ1. They identify their age, gender, country of living, education level, profession, experience in coding, and how they learned quantum programming languages and quantum physics, among other aspects.

In questions 15, 16, and 28 to 33, aspects of how quantum programming languages are being used were analyzed to answer RQ2. It analyzed how developers and researchers use languages in practice, such as for work, study, and research, among others. Questions about how tests are being performed were also analyzed to answer the RQ2.

Questions 14, 17, 18, 19, 21, and 22 were analyzed to answer RQ3 to identify which

programming languages are most used and which ones developers like the most. Questions 20, 23, and 27 were analyzed to answer the RQ4 and aim to answer why a developer chooses to work with a particular language and the main features they look for in the language.

To answer RQ5, questions 24 and 25 were analyzed to identify which quantum programming language is more likely to be used soon and the main reasons for choosing this specific language. Questions 26 and 35 were analyzed to discover the reasons that led someone to propose the creation of a new quantum programming language and answer RQ6. Finally, the analysis of the answers to question 34 aims to verify the participants' opinion regarding the number of quantum programming languages developed so far, analyzing whether they think there are too many or too few languages.

5.5 Threats to Validity

Based on the guidelines reported Wohlin et al. [109], we discuss below the threats to validity of our study.

5.5.1 Threats to External Validity

Threats to external validity are related to the generalization of our results. The study may not contain all the quantum programming languages ever proposed. However, extensive research was conducted to find as many languages as possible to mitigate this risk. The survey allowed the participants to use the "Others" text box to inform a different programming language. Some languages were informed in the "Others" text box, such as Xanadu's Pennylane, QUTIP, FunQy, SQIR, Quingo, and Perceval. These languages represent a ratio of 16.2% based on the 37 languages studied in this thesis. The number of participants who carried out the study may not contain a representation of the quantum program developers. Still, to reduce this risk, the survey was sent to many participants and shared on several social networks, as seen in Table A.2. In summary, we can see in Figure 6.3 the diversity of countries where the participants live, in Figure 6.11 the diversity of majors, and in Figure 6.16 the current job, which may indicate the amplitude of participants.

5.5.2 Threats to Internal Validity

Threats to internal validity are associated with uncontrollable internal factors that may influence our results. Our research questions were analyzed using plots and reports generated from scripts written using the R programming language [110], which is typically used for statistic analysis. In the construction of these scripts, there is a risk of implementation errors, which can generate mistakes in the analysis of research questions. To

minimize these risks, the scripts were reviewed by all authors of this thesis.

5.5.3 Threats to Construct Validity

Threats to construct validity are related to the design of the experiment. There is a risk that the questions used in the survey were not enough to analyze and answer all the questions proposed in this work. To mitigate this risk, a study was carried out with other surveys, as we can see in Section 5.2, to identify the main questions used in surveys for programming languages, as well as a study of methods on how to carry out surveys, as we can see in Section 5.3. The questions used in this study were designed based on the research questions we proposed to study. The survey was tested with a few participants before being made available to the public, so we could obtain feedback and make the necessary adjustments.

5.6 Summary

In this chapter, we described the methodology of our study and how it was conducted. We surveyed 251 quantum practitioners to answer the research questions proposed in this study related to the usage of quantum programming languages. The survey was published on specific channels of social networks about Quantum Computing, such as in groups on Facebook, LinkedIn, Discord, and Twitter. The publication was also carried out in other online places, such as Slack and in mailing lists. It was also emailed to authors of papers on quantum programming languages and other researchers in the field of quantum computing. This survey was conducted in two phases. The first phase was to pre-test and validated the survey, and the second was with all 251 participants. We also discussed the threats to the validity of our study.

Chapter 6

Results

In this section, we reported the findings of our study based on the information we gathered from the survey that was applied between March 31, 2022, and May 31, 2022. We had 251 responses, of which 208 (82.9%) answered that they had already used quantum programming languages and could move on to answering the following sections of the survey (see Figure 6.1). Furthermore, 43 (17.1%) answered that they had never used quantum programming languages and thus could not answer the questionnaire.

6.1 RQ1: Who is using quantum programming languages?

This research question aims to identify the profile of participants using quantum programming languages, whether academically, researching, or professionally. To determine their profile, we asked how old the participants were, what country they lived in, and what were their current job, among others.

Regarding gender, we can see that almost 90.0% of participants are male, as shown in Figure 6.2 and regarding the country where they live, we can see a great diversity, with the United States being the country with the most participants (22.1%), see Figure 6.3.

Figure 6.4 shows that 39.9% of the participants have the age from 25 to 34 years old, while 26.4% have the age from 18 to 24 years old and 18.3% have the age from 35 to 44 years old. Figure 6.5 shows that 27.9% have a doctoral degree, 35.1% have a master's degree, and 21.6% have a bachelor's degree. The reason that most of the participants have between 25 and 44 years old (58.1%) could be that they are more able to work with quantum programming in their master's and doctoral degrees since 63.0% of the participants have one of these two degrees, and 42.8% of them use it for research. The relation between participants' age and education level is shown in Figure 6.6. It shows that more than 80.0% of the participants between 25 and 64 years old have a master's or doctorate degree.

Regarding the participants' experience, they are coding using classical programming languages, both in total and professionally, as part of their work. We can see that most par-

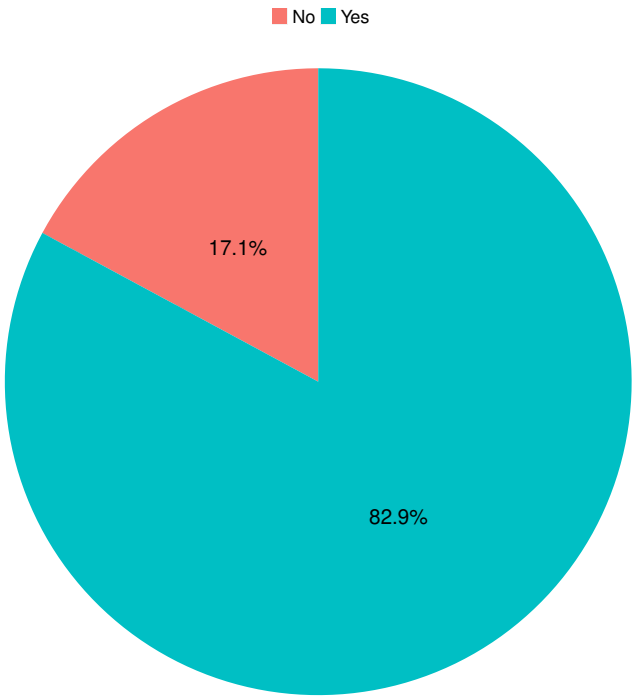

Figure 6.1: Distribution of participants that worked with quantum programming language and could answer the survey.

ticipants have many experiences, with 86.2% having more than five years of experience, and 50.0% over ten years, as shown in Figure 6.7. Regarding professional experience, we have a slight decrease in experience, with 50.0% having over five years of experience and 28.8% over ten years, as per Figure 6.8.

We also added a question to the survey to find out how participants learned to code and that most learned from Books and other online resources. In Figure 6.9, we can see that most participants used books to learn to code (66.8%), learned using an online forum (61.5%), and learned at school (60.6%). Most participants program in Python (92.3%), a classical programming language. Consequently, this is the classical programming language on which most quantum programming languages are based, as shown in Figure 4.2, which describes the roots of programming languages. Other languages are also widely used like C++ (60.1%) and C (55.3%), as we can see in Figure 6.10.

As expected from the participants who completed a major, most are related to Computer Science (47.1%) and Physics (37.6%), as shown in Figure 6.11. Figure 6.12 shows the relation between the primary quantum programming language used by the participant and their major.

An important question regarding the profile of the participants is related to their knowledge of quantum physics, considering that for the development of quantum algorithms, it is necessary to know this area. In the survey, we asked the participant's level of knowledge concerning quantum physics, classifying them from 0 (no knowledge) to

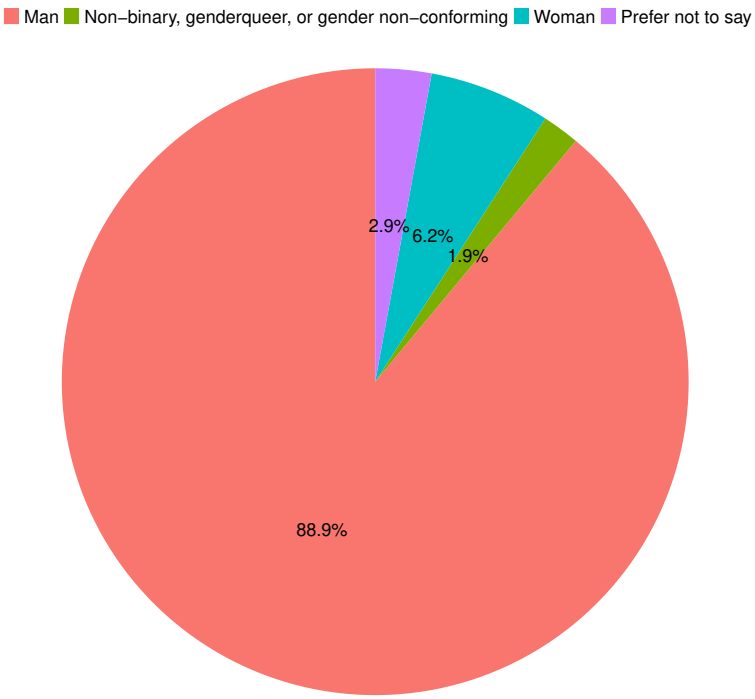

Figure 6.2: Distribution of the percentage of the gender of the participants.

5 (expert). We found out that all participants have some knowledge of quantum physics, with the level of experience being quite balanced between 1 and 5, as shown in Figure 6.13. We also asked how they learned quantum physics, with most of them having learned using books and/or at the university with 69.6% and 64.7%, respectively, according to Figure 6.14.

As shown in Figure 6.15, there is a relation between the participants' language and their knowledge of quantum physics. Some languages might not require an expert level in quantum physics, Braket SDK, QASM, QML, and Q# are used by participants with a novice knowledge in quantum physics, while others like Strawberry fields, Quil, and Quipper are used by participants with expert knowledge in quantum physics. Qiskit is used for participants with different knowledge of quantum physics, which may indicate that it has features that participants can use with different levels of knowledge.

Regarding the current job of the participants, as shown in Figure 6.16, most are Developers/Programmers/Software Engineers (37.5%), students (35.1%), or Scientists/Researchers (33.2%). As participants can choose more than one item, we verified in the answers that most of them have more than one role, such as Academic Research and Student, Academic Research and Teacher, Scientist/Researcher and Developer, and Developer and Other roles like Team Lead.

Summary RQ1: Almost 90.0% of the participants that answered the survey and are using quantum programming languages are male, and 22.2% of them live in the

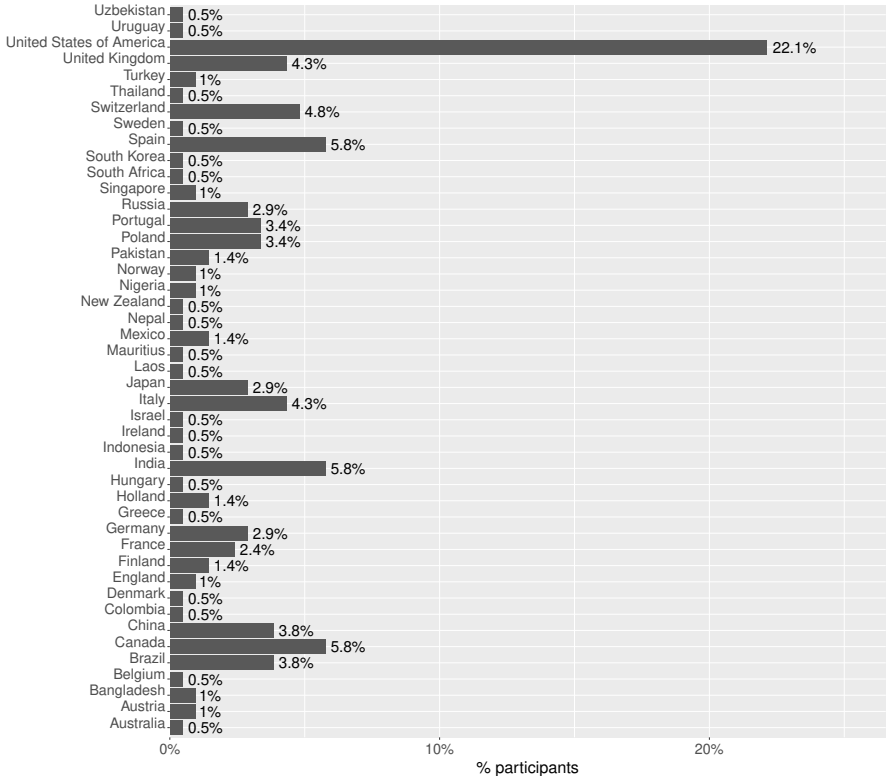

Figure 6.3: Distribution of the percentage of countries where participants live in.

United States of America. 58.2% have between 25 and 44 years old. 63.0% have master or doctoral degrees. 86.2% have more than five years of experience using classical programming languages.

6.2 RQ2: How are quantum programming languages being used?

The main purpose of this research question is to identify how the participants use quantum programming languages in practice. According to the survey's data, 42.8% use quantum programming languages for research, while 34.6% because they like to learn new languages, and 16.4% use them for work. The fact most use quantum programming languages indicates that this is a relatively new field as many students and researchers carry out innovative research in this area. Figure 6.17 shows the participants' responses. De Stefano [111] conducted an empirical study on the current adoption of quantum programming to answer to what extent and to what kind of tasks are quantum programming languages being used. In that study, the author searched GitHub repositories that use Qiskit, Cirq, and Q#. Then he analyzed the README file in each repository to identify what these languages are used for. As a result, he identified that the main purpose of the use is for exercise, research, and teaching. The other primary purpose for quantum programming is to develop quantum libraries or frameworks, which represent 16.0% of the

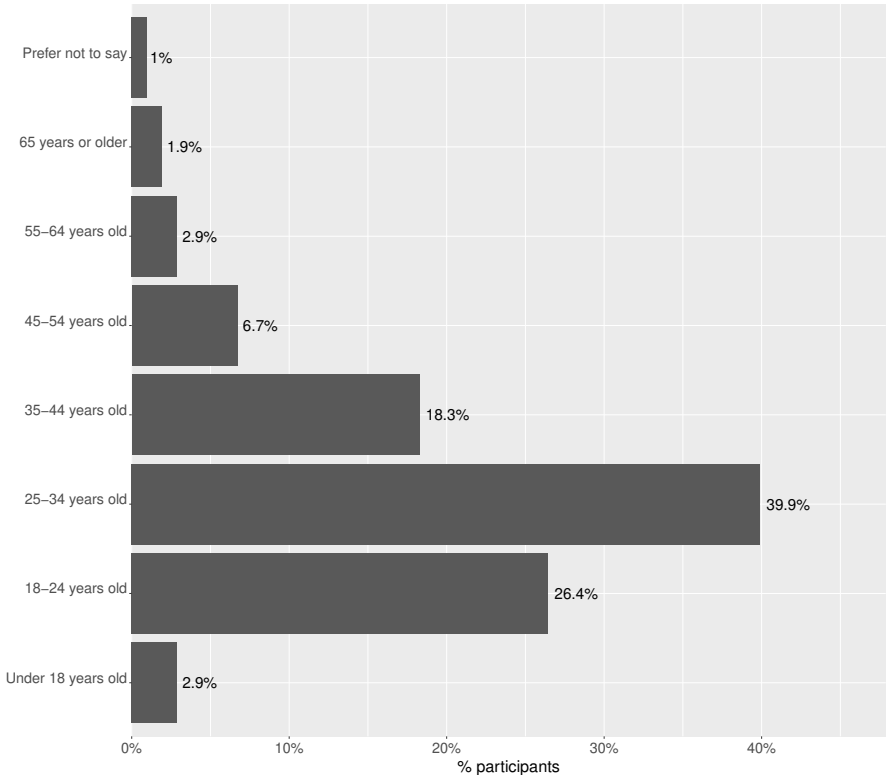

Figure 6.4: Age of the participants grouped by category and percentage.

total repositories analyzed. In line with our study, we can verify that the use of quantum programming languages in GitHub repositories is related to the participant's responses to the survey, in which most use them to research and learn it.

The relation between the quantum programming language used by the participants and for what they use is shown in Figure 6.18. Some languages like Ket, QASM, Quipper, Silq, and Strawberry Fields are used mainly for research, while Orquestra and Quil are used for work.

In terms of the use of quantum programming languages in general and professionally, according to Figures 6.19 and 6.20. We can see that in terms of experience, the participants have much less experience in quantum languages than in classical languages. Overall, 83.7% of participants have less than four years of experience with quantum programming languages, and they have the same percentage for classical languages with more than five years of experience. Only 16.3% of participants have more than five years of experience with quantum programming languages. Regarding their professional use, this number is much lower, with 38.0% never used quantum programming languages for work and 22.6% having less than one year of experience.

Figure 6.21 and Figure6.22 show the experience and professional experience of the participants in terms of quantum programming language. Most participants have less than four years of experience, which might be because most languages were created recently.

We asked the participants what tools they think are necessary or missing to develop

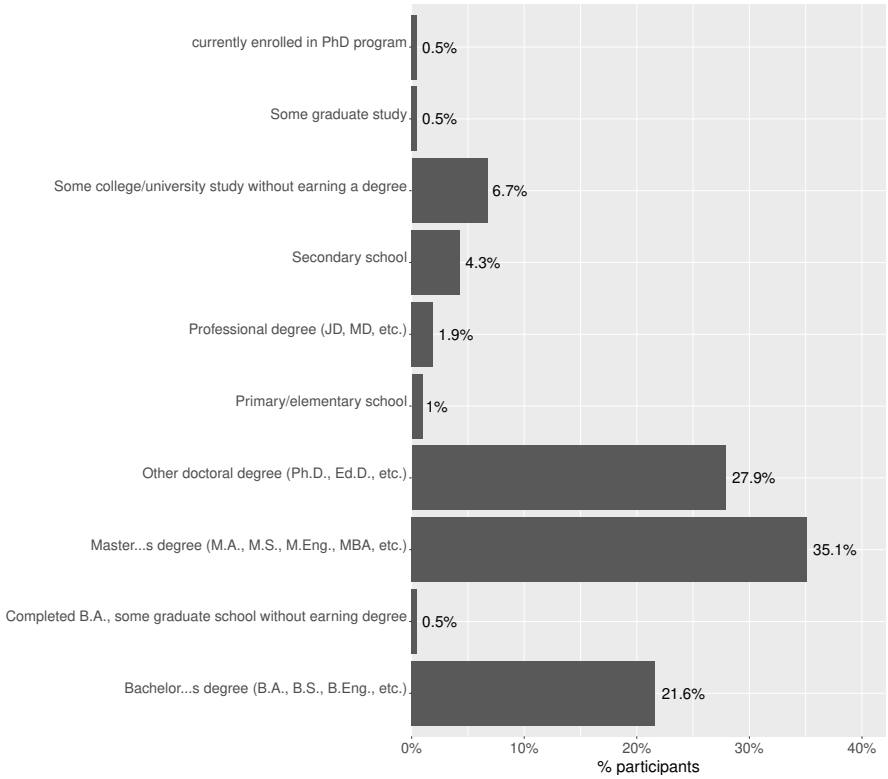

Figure 6.5: Formal education of the participants by percentage.

better and faster Quantum Programs. We found out that most of them think that a tailored quantum IDE and tools for testing and debugging are currently missing. Another point already seen in other answers is the lack of a higher-level quantum programming language, indicating that some participants think creating a new high-level language is necessary to facilitate the effective development of quantum programs. The participants also raised some other points, such as:

- Quantum debugging tools and classical simulations running parallel to a quantum IDE.
- Quantum IDEs, with proper debug and circuit visualization tools.
- Implementation of error correction.
- Better device level optimizers, simulators, and device access. Improving device specifications for cross-QPU development.
- Meaningful standardised benchmark suites.
- Easy interoperability between frameworks.

Many participants complained about tools to test and debug their quantum programs. As we can see in Figure 6.23, 76.4% of participants test their quantum programs while 23.6% do not. Figure 6.24 shows that for some quantum programming languages, all participants perform tests, e.g., Braked SDK, QHaskell, Orquestra, and other languages they do not perform a test, such as QASM and Ket. The number of participants who do

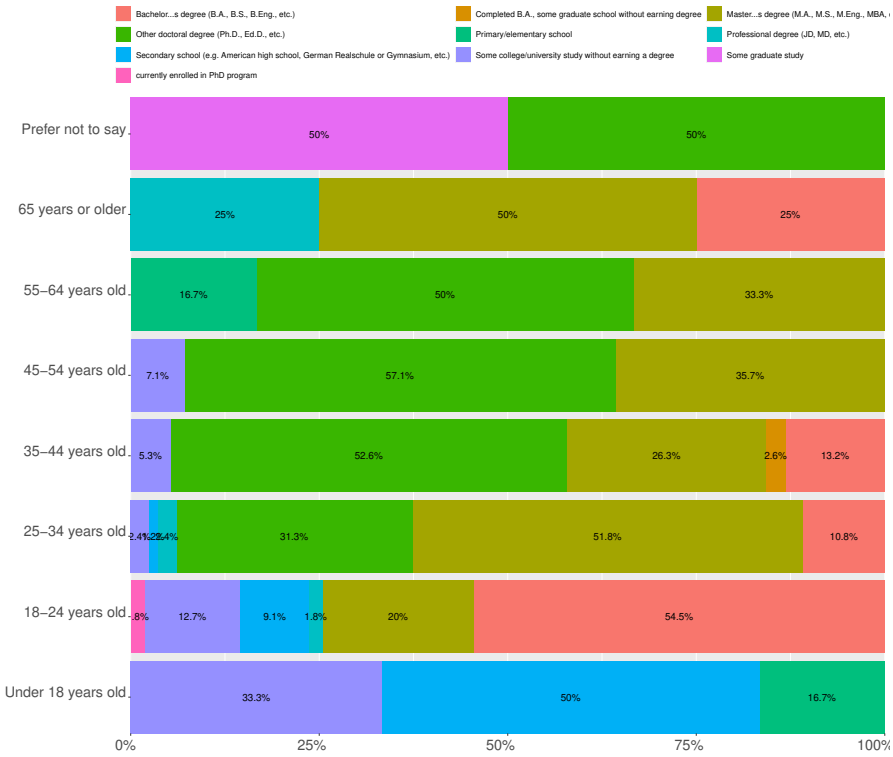

Figure 6.6: Relation between the age and education level of the participants by percentage.

not perform tests is very high, considering that 100.0% of quantum programs should be tested to ensure they are correct. Of the participants who perform tests, we can see that the majority, 64.7%, perform tests every time they change their code. In contrast, 16.2% only perform tests before going to production, and 6.6% perform tests every day, according to Figure 6.25.

Another point that drew attention, according to Figure 6.26, was that 57.4% of the participants performed tests manually, while 42.6% performed them automatically, which indicates a lack of test automation tools for the development of quantum programs. Also, Figure 6.27 shows that in languages such as QML and Orquestra, Cirq, and Quil, the participants mainly perform automatic tests. In contrast, the other participants perform both manual and automatic tests, mainly manual ones.

To identify which tools participants use to test their programs, we listed the main testing tools for quantum software, with the option for participants to enter another tool to find out which tools they use. The tool most used by participants is Qiskit - QASM Simulator, which is used by 77.1%, as shown in Figure 6.28. This number is coherent, given that Qiskit is the primary language used by participants (64.9%), as we can see in Figure 6.33. Figure 6.29 shows the relation between the quantum programming language and the tool used for tests.

The participants use quantum programming languages for different reasons. As we

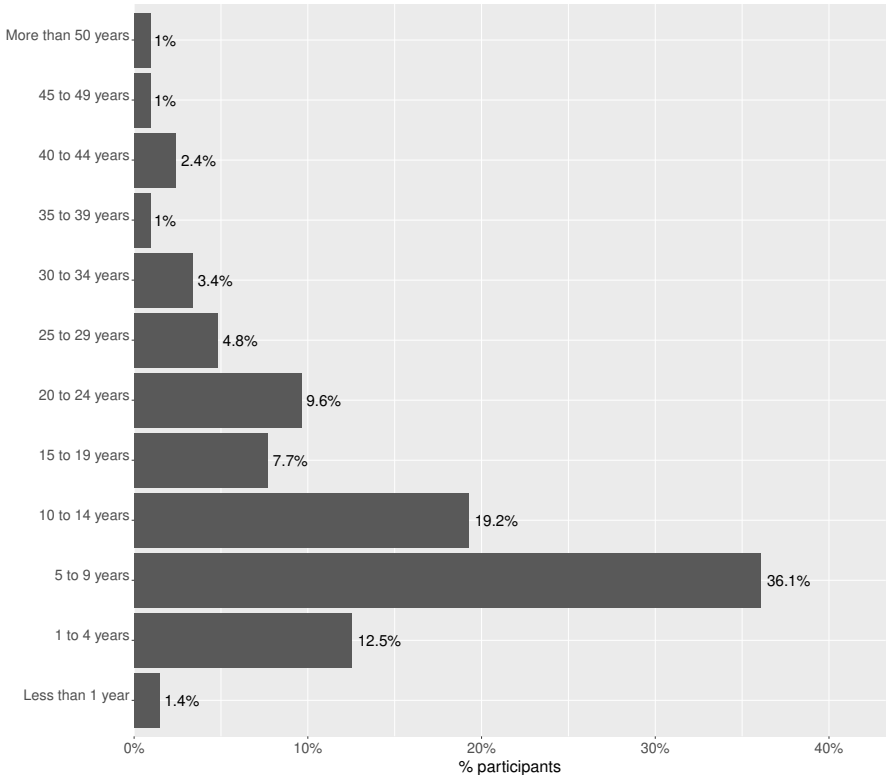

Figure 6.7: Experience of the participants in terms of coding by percentage.

can see in Figure 6.17, most participants use the languages for research or because they like to learn. Figure 6.30 shows the relation between the reason the participants use the languages and their work field. Most participants who work with economics use quantum languages because they like to learn, while those who work with physics use them for research. The interesting point is that most of the participants who have a major in art/humanities use quantum programming languages for other reasons than to learn, research or work.

Figure 6.31 shows the relation between the current job of the participants and the reason they use the quantum programming languages. The participants who are working as Architect, Product Managers, Technical Support and Technical Writer are mainly using the language for work, while those who are working as Academic Research, Instructor/Teacher/Tutors, Scientist/Researchers, and Students are mainly using it for research; and Tester/QA Engineer, UX/UI Designer and DBA (Database Administrator) are using it because they like to learn the languages.

Summary RQ2: 42.8% of the participants use quantum programming languages for research, 34.6% use them because they like to learn new quantum languages, 16.4% use them for work, and 6.3% use them for other miscellaneous tasks.

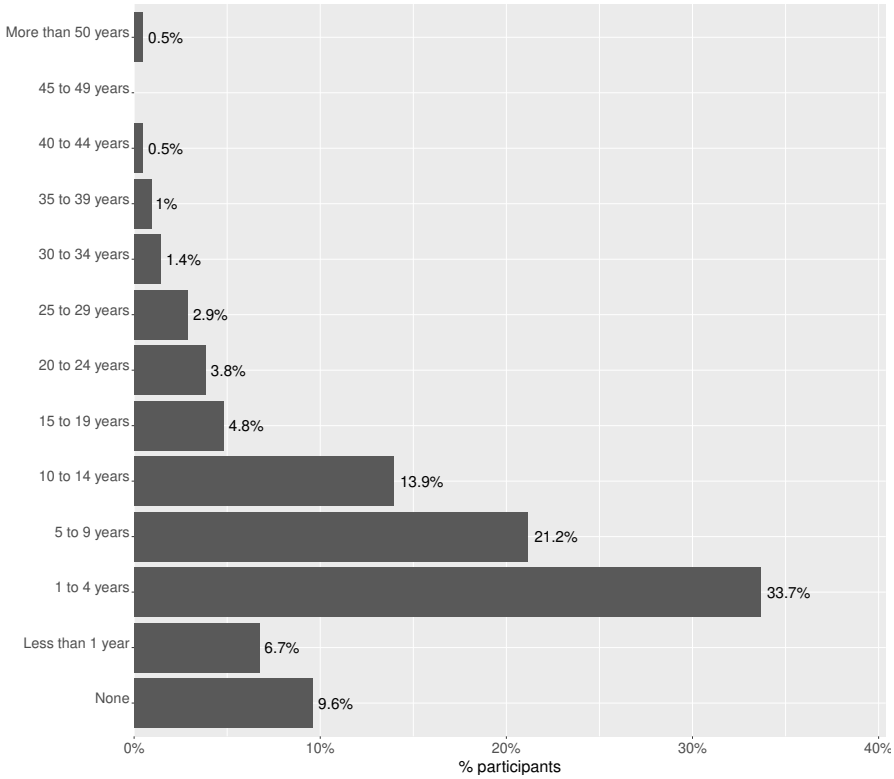

Figure 6.8: Professional experience of the participants in terms of coding by percentage.

6.3 RQ3: What are the most used quantum programming languages? Why?

In the survey, questions were carried out to identify which quantum programming languages the participants had already used and for how long they had used the languages, as the question's answers are shown in Figure 6.32. According to the participant's responses, the vast majority of participants, 177 (85.1%), have already used Qiskit (Python), the most used quantum programming language. Qiskit is also the most used language by participants for up to 6 years. From 7 to 8 years, cQAMS, OpenQAMS, and Qiskit (Python) are tied as the quantum programming language most used. From 9 to 10 years and more than 11 years, Quipper is the most used. Overall the second most used language is Cirq (Python) with 91 (43.8%) responses, and in third, OpenQASM with 77 (37.0%) responses, as shown in Table 6.1.

We also asked what the primary quantum programming languages the participants use are. We can see that most participants use Qiskit (64.9%) as their primary quantum programming language, the second most used being Cirq (5.3%), as we can see in Figure 6.33.

Some participants mentioned some languages that were not among the options in the survey. Among the most mentioned are Xanadu's Pennylane (4 participants) and QUTIP (2 participants). Others were mentioned only once, e.g., FunQy, SQIR, Quingo, and

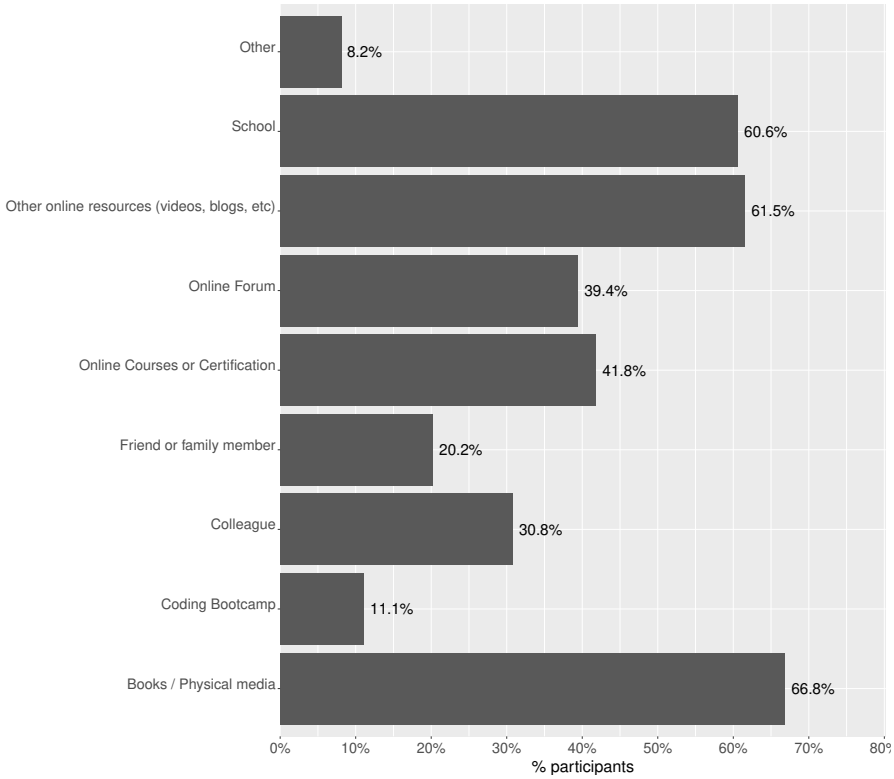

Figure 6.9: How the participants learned to code by percentage.

Perceval.

We also asked how the participants learned quantum programming languages, and we concluded that most of them, 60.6% learned from the languages' documentation. In contrast, 41.3% learned through online courses, and 38.9% learned from books, according to Figure 6.34. It is evident the importance of language documentation concerning learning, and the more documentation the language has, the more likely participants will use it. Figure 6.35 shows where the participants learned their language. For the Orquestra language, all participants learned using the language documentation, which might indicate that the language provides good documentation, while the participants at the University learned QHaskell.

There are several reasons why Qiskit (Python) is the most used quantum programming language by participants. An important factor is the large number of tutorials, course materials, and resources for learning quantum programming and Qiskit. It is a framework based on Python, and the participant is not limited to using only the features of Qiskit. However, it can also use all the Python language and its libraries, making it possible to start learning and developing quickly and easily. Furthermore, another point is the policy of democratizing the use of quantum computers through free access to real quantum hardware, making it possible for users to write code and run their codes on IBM's quantum computer.

Although not a quantum programming language, Python is the most used classical

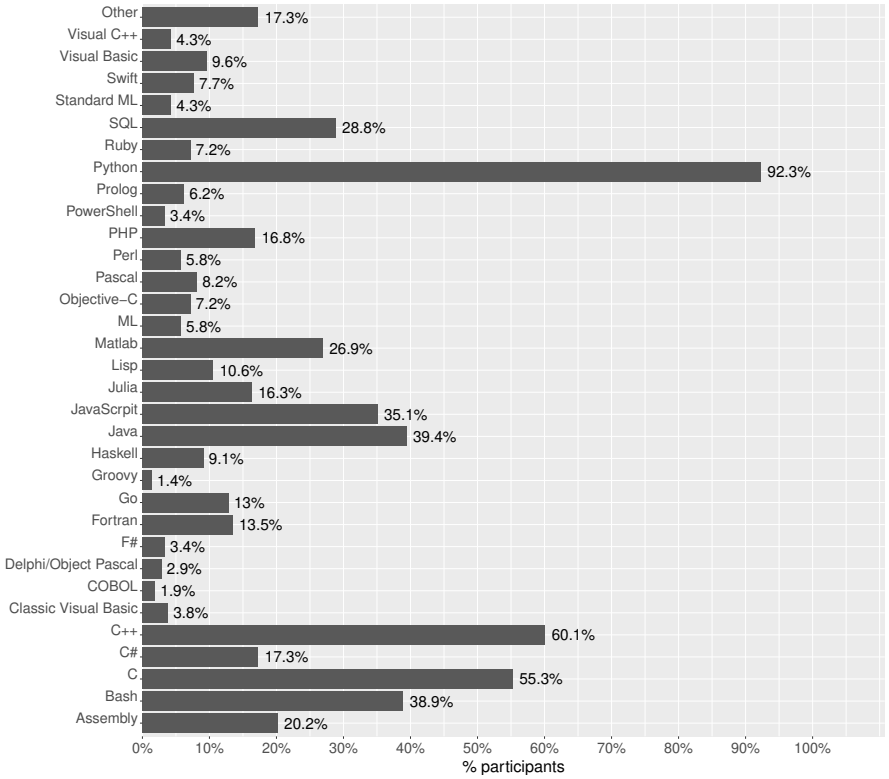

Figure 6.10: Classical programming languages the participants used by percentage.

language for building quantum programs, making use of quantum frameworks, libraries, and language extensions. Python is used by 92.3% of the participants in general and 73.6% as a primary language.

Summary RQ3: According to the participant's responses, Qiskit (Python) is the most used quantum programming language, used by more than 85.0% of the participants that responded to the survey, followed by Cirq (Python) with 43.8% and OpenQASM with 37.0% of the participants. Python is the most used classical language to build quantum programs, used by 92.3% of the participants.

6.4 RQ4: What makes a person choose a quantum programming language? Why?

Several reasons lead a person to choose which programming language he or she wants to use, e.g., easy-to-learn syntax, online documentation, examples, and several forums. Regarding quantum programming languages, we can see a significant lead from Qiskit (Python) in terms of usage and understanding of why the participants use this quantum programming language the most. We asked them to rate it using a scale of one to five in terms of features, available documentation, code examples, several forums, support, and

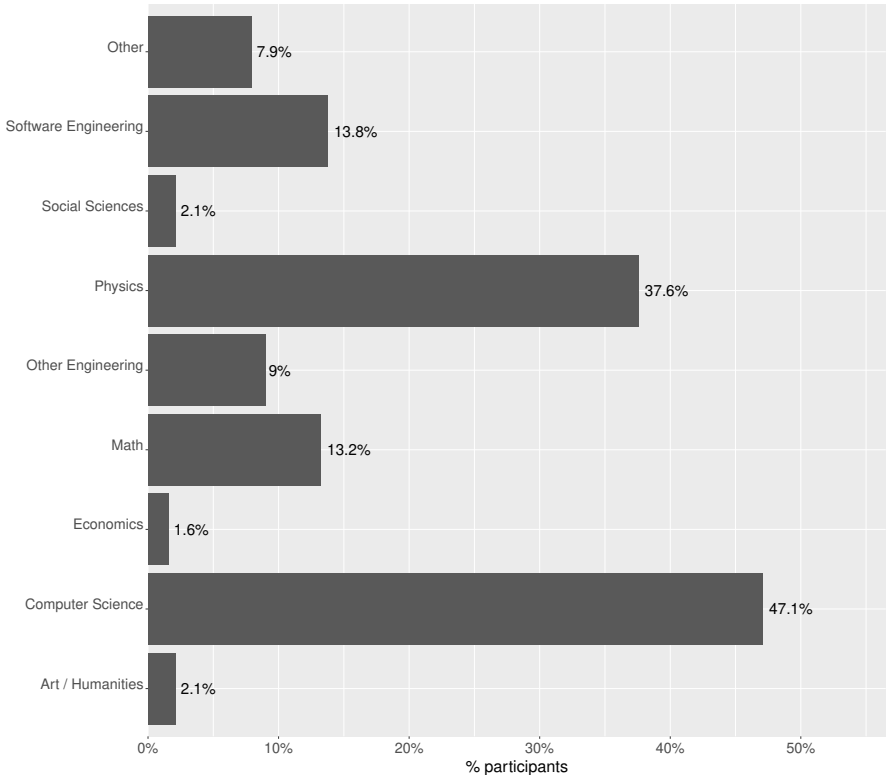

Figure 6.11: The work field of the participants by percentage.

easy-to-code. As shown in Figure 6.36, Qiskit was rated with a three or higher rank by more than 88.0% of the participants.

Most participants who have chosen Qiskit reported that it is open-source, easy to understand, easy to code, Python-based, has many tutorials, continuous updates, operators available, and has a very active community. Some participants also reported that the fact that the framework is from IBM was also a positive factor, and others, the number of modules that the language provides, as one participant mentioned: “There are many modules help us to run real quantum hardware.”.

The points most mentioned by the participants about Qiskit were that the framework uses Python, a language in which more than 92.3% of the participants who responded to the survey have already used it, and also the fact that it has a large and active community.

Regarding what participants do not like or miss about Qiskit, we can highlight the performance and consumption of RAM memory, over-complicated architecture, the fact that it is still low-level, and the fact that it has few examples to run on current quantum hardware.

An extensive and active community was one of the biggest reasons participants’ chose a programming language. We asked the participants which are the most used forums, and the answers were StackOverflow, used by 76.2% of the participants, Slack by 47.1%, and QOSF by 22.7%, as we can see in Figure 6.37. StackOverflow is the only forum used for some languages, e.g., QML, QHaskell, and QASM, as shown in Figure 6.38.

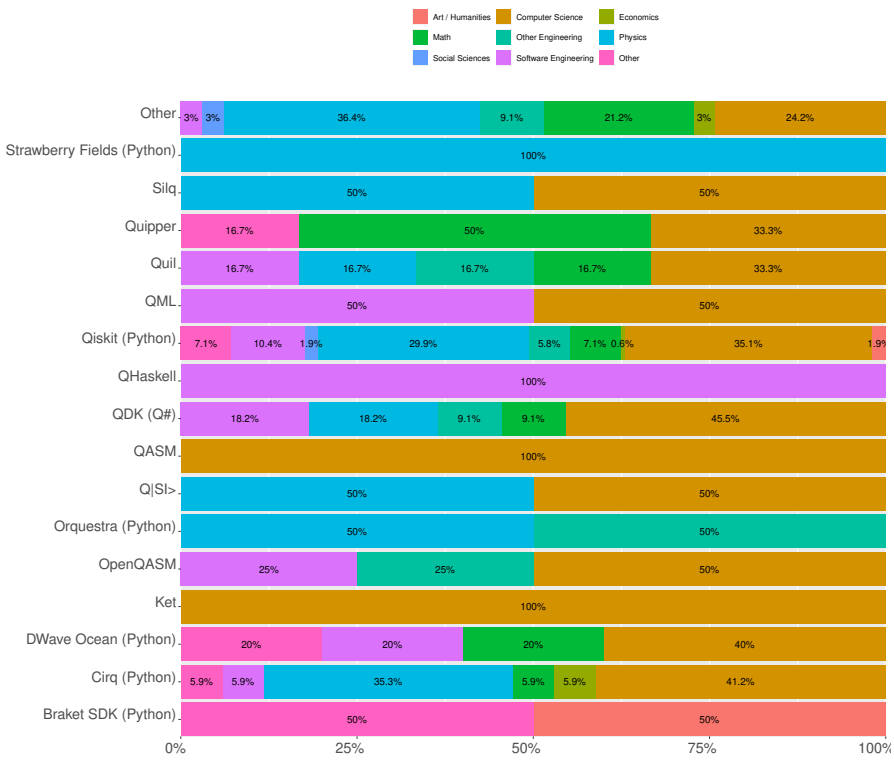

Figure 6.12: Relation between the primary quantum programming language chosen by the participants and their major by percentage.

An essential factor to verify what makes a participant choose a quantum programming language or to know their opinion on why it is essential to learn quantum programming languages and to find out the participants' opinions concerning this topic, we asked the question "What makes learning Quantum Programming Language important" in the survey. We observed several answers, e.g., "very interesting"; "research"; "quantum computers are the next evolutionary step"; "solving optimization problems"; "opportunities in the future". However, the most common answers given by the participants were:

- It is an important emerging field.
- Development of quantum algorithms.
- Use familiar programming languages, such as Python.
- Features to implement quantum concepts.
- Their future application in Software Engineering.
- It is necessary in order to manipulate the technology hands-on.
- They are the future of programming.
- Technological applications, fundamental research, and democratization of quantum computing.

The fact that quantum computing is an innovative area with much potential in the future is a significant factor for the participants to work with a quantum programming

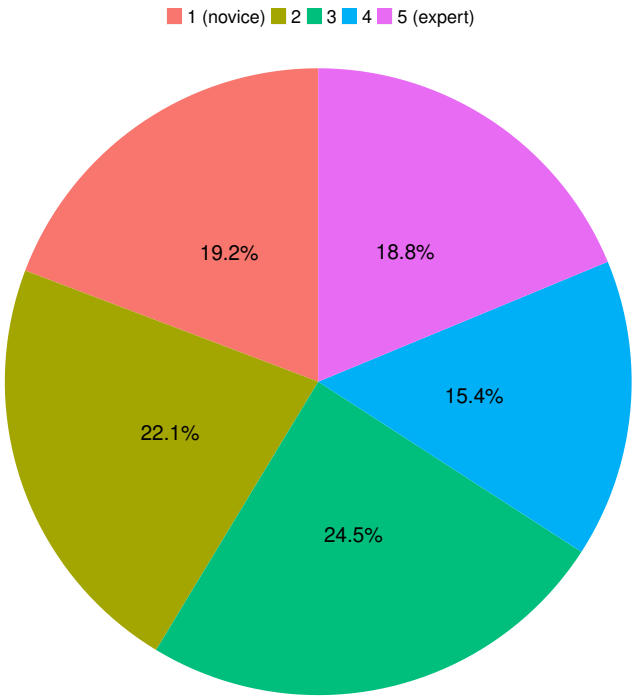

Figure 6.13: Level of education in terms of knowledge in quantum physics by percentage.

language. We can conclude that the closer a language gets to the real world and be used in actual quantum hardware, the more motivated the participants are to learn the quantum programming language.

Summary RQ4: Participants mentioned several reasons for choosing a quantum programming language. The most mentioned ones were open-source, easy-to-learn syntax (based on an existing programming language), available documentation and examples, an active community, and available features.

6.5 RQ5: Which of them has the best chances of being imposed over the rest?

To answer this research question on which quantum programming language has the greatest chance of imposing itself on top of the others, two questions were added to the survey to find out which language the participants would like to use in the near future. The answers can be found in Figure 6.39 and also what leads them to try a new language, as we can see the answers in Figure 6.40.

Regarding the question on which quantum programming languages the participants would like to try in the near future, we can see that the most languages chosen in the answers were Cirq (25.5%), Qiskit (20.2%), and Q# (18.8%), which are the languages of the largest technology companies (Google, IBM, and Microsoft, respectively). These

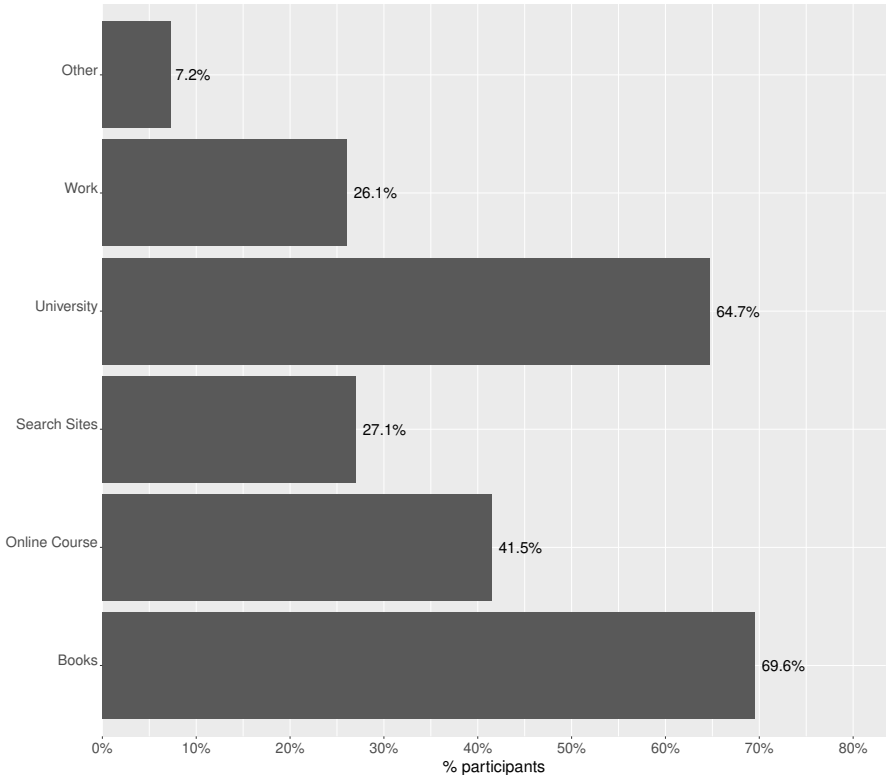

Figure 6.14: Education of the participants in terms of learning quantum physics by percentage.

choices are strongly linked to the main reason they chose why they would like to try these languages, in which 63.9% of participants said they would choose them because they "heard about the language.". Figure 6.41 also shows the relation between the quantum programming languages and the reason why the participants want to use the language, which also shows that for most of them, it is because they "heard about the language". The big companies have excellent worldwide visibility, and their software also has great visibility and a good chance of being tried by developers. Other reasons that participants chose why they would try one of these languages were: Widely used (24.0%) and Reading an article about the language (23.6%).

The quantum programming languages created by large companies that have high visibility and are aligned with the hardware of a real quantum computer have the best chances of being used in the near future. The reason is that most users want to work with cutting-edge technology and a real quantum computer instead of simulators. The language that aligns with these expectations and with an easy syntax based on existing programming languages, like Python, has significant changes to impose itself concerning the others.

Summary RQ5: Cirq with 25.5% was the most chosen quantum programming language by the participants, followed by Qiskit with 20.2% and Q# with 18.8%. The main reason for these choices was that they "heard about the language", indicating

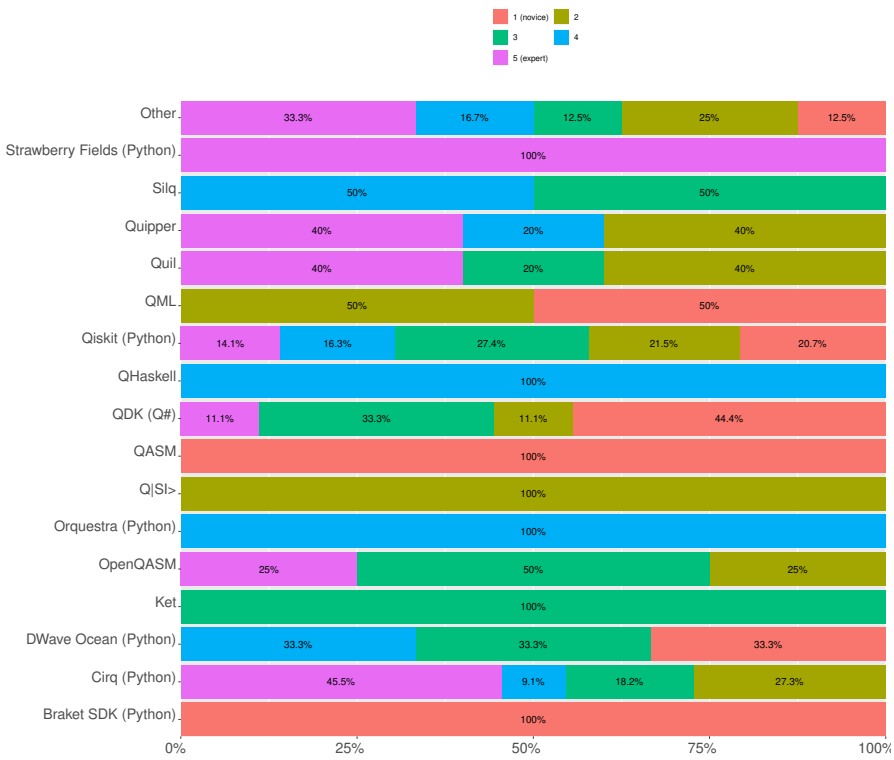

Figure 6.15: Relation between the primary quantum programming language chosen by the participants and their knowledge in quantum physics by percentage.

big technology companies' credibility and worldwide visibility.

6.6 RQ6: What makes someone propose a new language?

To answer the question of what makes someone propose a new language, we first investigated the main challenges developers find in quantum programming languages and if they think it is still necessary to create other languages.

Regarding the main challenges they encounter, many participants responded that the lack of documentation and examples to use the language is one of the main challenges they currently face. One participant responded, "Lack of documentation and examples," while another responded, "The lack of documentation and videos, tutorials, etc. showing the basics of these languages.". Many of the participants have difficulties in finding documentation online as well as some participants also reported a lack of communities to ask questions about the languages.

Another challenge reported by the participants is that many quantum programming languages are very different, making a choice very difficult for them. One participant said, "too many choices, and they are all different." while another "There are so many frameworks and difficulty to try and find the best one.". In addition to many languages and lack of documentation, participants reported that many languages have design limitations

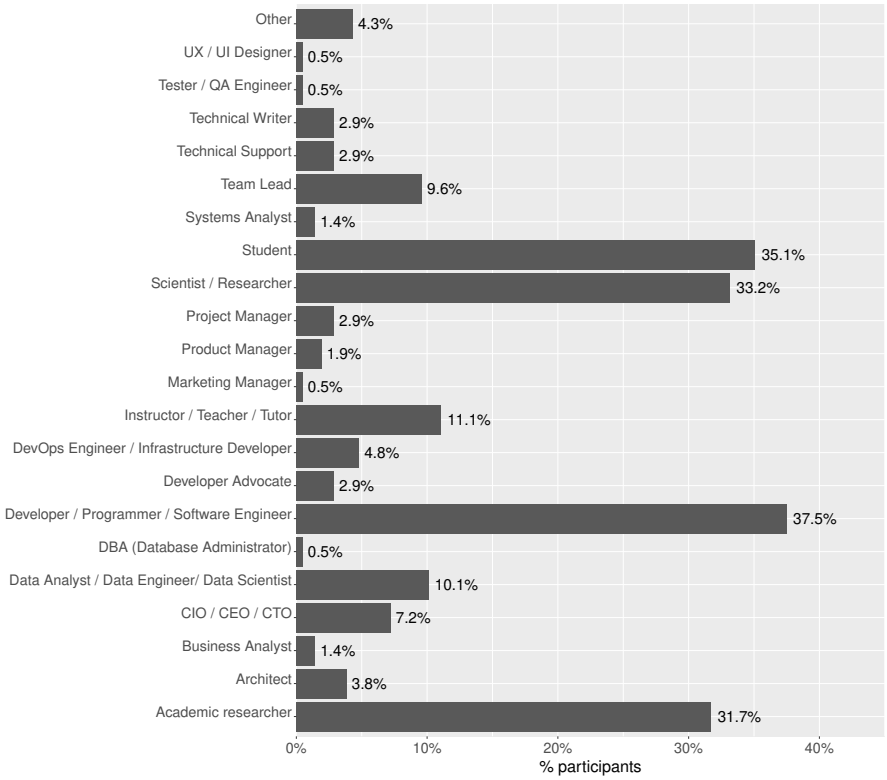

Figure 6.16: Current job of the participants by percentage.

and missing features.

The lack of conceptual knowledge about qubit, measurement, and quantum physics was also reported as a factor that made it very difficult for the participants. According to the participants, it is difficult to choose a language and write algorithms/programs with it as most of them also lack tooling for abstractions and high-level reusability.

A summary of the main challenges encountered by the participants:

- Lack of online documentation, videos, tutorials, and examples.
- Lack of conceptual knowledge about quantum physics.
- Design limitations and missing features in the languages.
- Not having a universal language.
- Not enough software engineering concepts.
- Manufacturer dependency on the languages and lack of interoperability.
- The variety of available languages.
- Lack of tooling for abstractions and use of quantum programming languages.
- Small community of users to help use the languages and answer questions.
- Syntax of the languages is very different from language to language.
- Availability to test the languages in real quantum computers instead of simulators.

Regarding the participants' opinion, if they would need yet another Quantum Programming Languages in the near future, we can see in Figure 6.42. While some par-

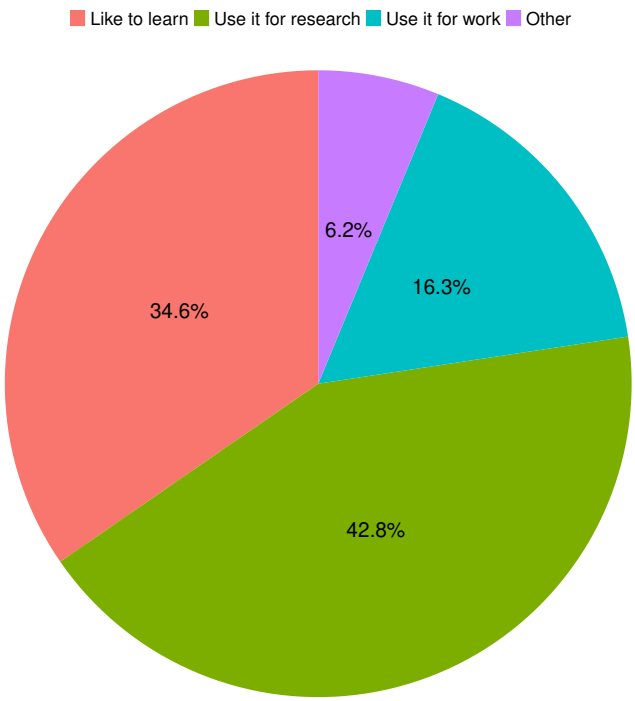

Figure 6.17: Reason the participants use quantum programming languages (e.g., Like to learn, use it for research, use it for work) by percentage.

ticipants think yes, new languages will be needed shortly, as one participant said, "Yes. Higher-level programming. Right now, everything is low level with very little optimization for compilation." others think it is unnecessary, as one of the participants said: "No, it is better to learn the current languages and, if necessary, contribute to their improvement.".

Figure 6.43 shows that depending on the language used, the participant's opinion regarding the need for another quantum programming language can differ. All of the participants who use OpenQASM, QHaskell, Silq, and Strawberry fields responded that there is a need to create a new language. However, the participants who use Quipper, QML, Ket, Ocean Software, and $Q|SI\rangle$ do not think another language is necessary. Based on the open answers, we can conclude that these languages have most or all of the features they think are necessary to develop their quantum programs or because they believe that adding new features to the existing languages is better.

The main reasons that participants reported as being necessary for the creation of a new language for quantum programming are:

- The creation of a universal language.
- More standardized quantum programming languages.
- High-level programming languages are needed instead of low-level languages.
- The languages are not yet mature enough, and the hardware is quickly changing.
- New algorithms will require new languages.
- To support cryptography and post-quantum cryptography concepts.

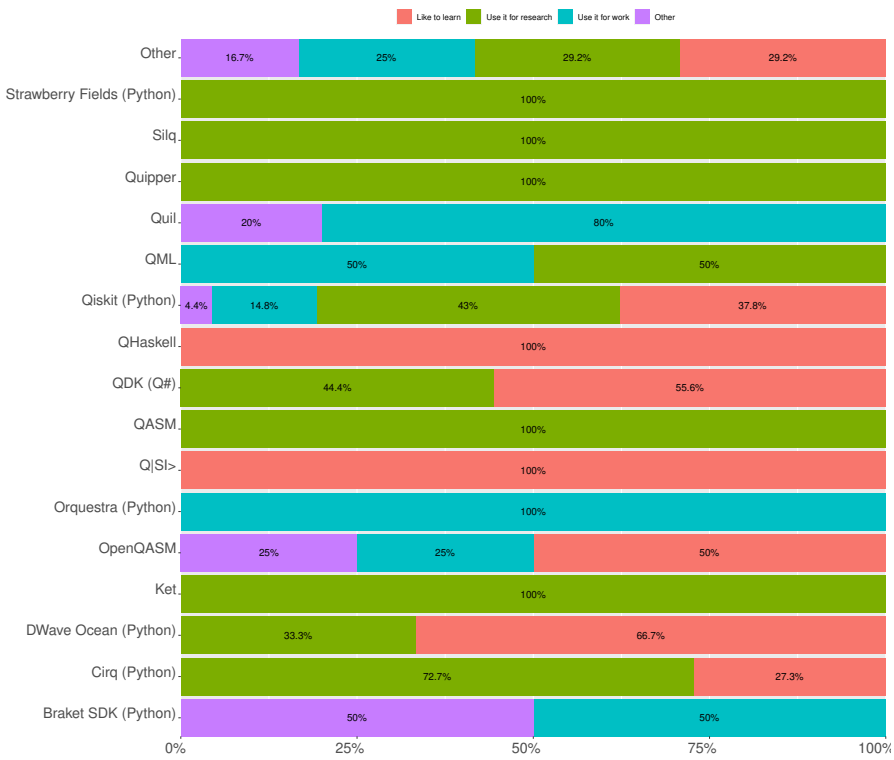

Figure 6.18: Relation between the primary quantum programming language chosen by the participants and for what they use the language by percentage.

- Many ideas need to be explored, especially as the number of qubits scales.
- Most of the current programs are based on circuits that are way too low-level.
- Languages for handling error correction in real devices.
- There is room for a language that enables orchestration between quantum expression and control hardware more effectively.

As for the participants who disagreed with the need to create new quantum programming languages, the main reasons were:

- The problem concerns standard libraries of generic, optimized, and verified subroutines and languages rather than programming languages.
- A new language is unnecessary unless it adds something new to the quantum field.
- Many languages overlap, and a standard language for quantum computation could accelerate development.
- It is better to learn the current languages and, if necessary, contribute to their improvement.
- The popular languages like Qiskit (Python), Q#, and Cirq (Python) are enough.
- Too many already, and consolidation is needed.
- The existing ones are sufficient.

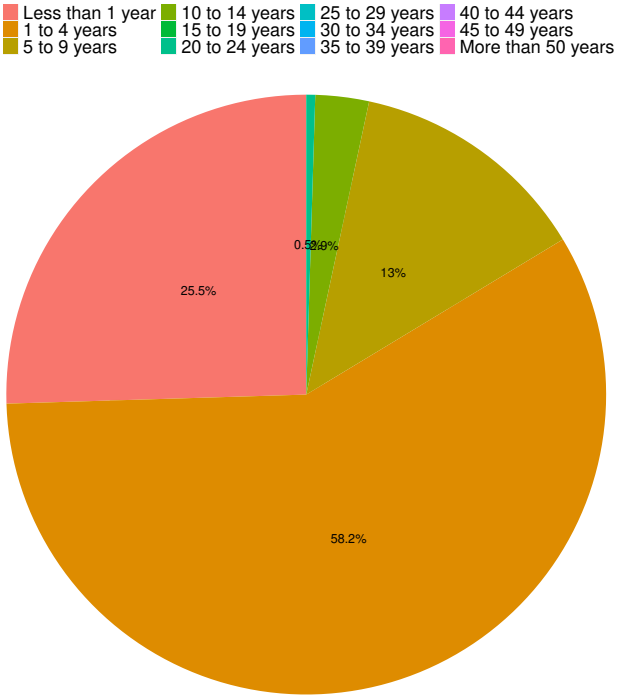

Figure 6.19: Experience of the participants in terms of using quantum programming languages by percentage.

Summary RQ6: Most participants responded that developing another quantum programming language is necessary, 31.7%. In contrast, 25.0% of them responded not. The main reasons for the need for another quantum programming language were a more mature, standardized, high-level quantum programming language and new features (e.g., handling error correction in real devices).

6.7 RQ7: Are there too many quantum programming languages?

Regarding this research question, we aim to find out if there were too many quantum programming languages created and, if so, why. In the survey, a question about the participants' opinion regarding the number of quantum programming languages, we try to find the answer to this question. As shown in Figure 6.44, most participants (42.8%) answered yes. In contrast, 18.3% answered no, and 38.9% did not answer this question for different reasons like they did not know or did not have the knowledge to answer it.

One of the participants answered the question, "Too many - there's a lot of overlap between the languages, most could be done if a single language was settled on.".

Several important points were raised as reasons by the participants:

- There are many languages with overlapping features that are not standardized.

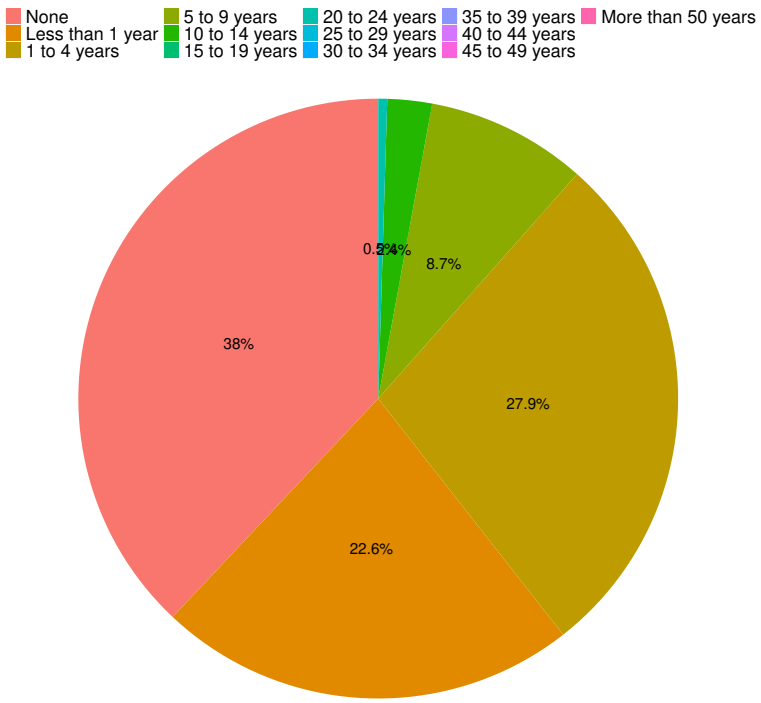

Figure 6.20: Professional experience of the participants regarding quantum programming languages by percentage.

- Because quantum programming is still in the early stages of development, there are many different proposals for languages.
- Many of these languages are research languages that do not have great community support.
- Many languages are needed until we find which features are required in a quantum programming language.
- Many participants want to make their own language or have a better abstraction for them. In classical computing, everyone tries to write their language, interpreter, or compiler, which is the same as quantum computing.
- The languages have too little interoperability, and too many were created by start-ups that will die soon.
- There must be a standard for the creation of a universal language.
- There are many overlaps between the languages, which could be solved if a single language were established.
- Many languages are created because the developers encounter problems within a specific language.
- Many companies run hardware platform competitions and create languages for their hardware.

Figure 6.45 shows the relation between the quantum programming language used by the participant and their opinion regarding the number of languages. The participants

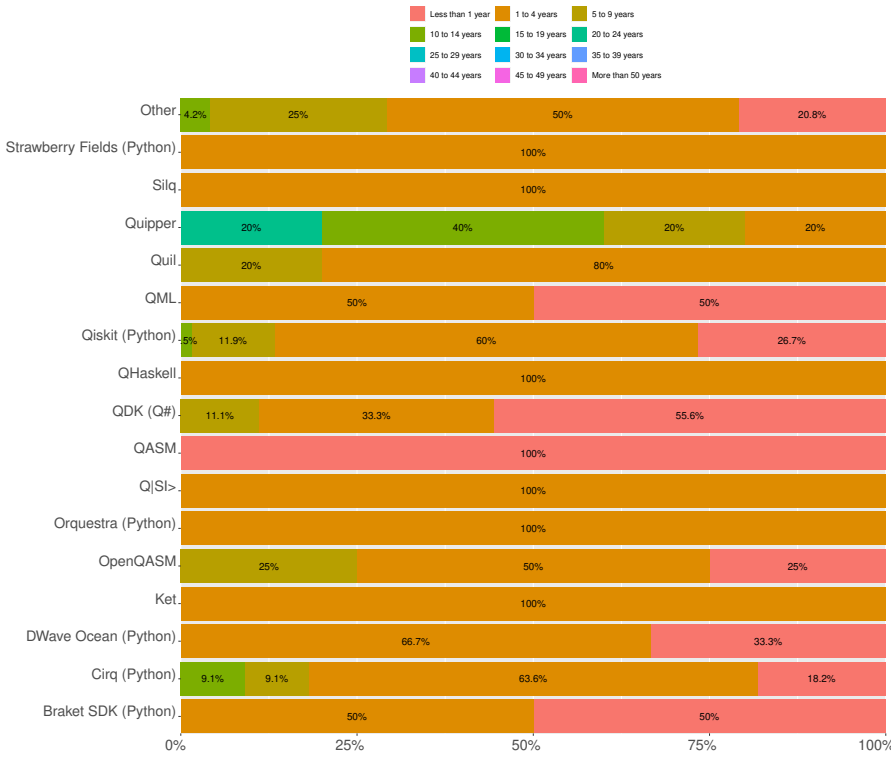

Figure 6.21: Relation between the primary quantum programming language chosen by the participants and their experience by percentage.

responded that there are too many languages for most of the languages used. For some of them (e.g., QASM, QML, Braket SKD), all participants answered that there are too many languages currently available.

There are several reasons to justify a large amount of quantum programming language. For example, many languages were created for an application area, and others were created to take advantage of some specific quantum computing properties. Another point is that as it is a recent and innovative area with great growth potential, many companies are investing in creating hardware (quantum computers) and software. According to most participants' opinions, there are many quantum programming languages, which does not mean that many others will not be created for different reasons, such as the natural evolution of languages as quantum programming languages emerged for many reasons and as developers realized that things could be done from better, more intuitive, and therefore faster, they were creating their own languages. Adaptation to the historical moment, for example, with the creation of new hardware with different technologies, new languages will be needed to operate them. A universal language for quantum computing would be fascinating, which is unlike to happen because specialization is always needed to deal with specific problems.

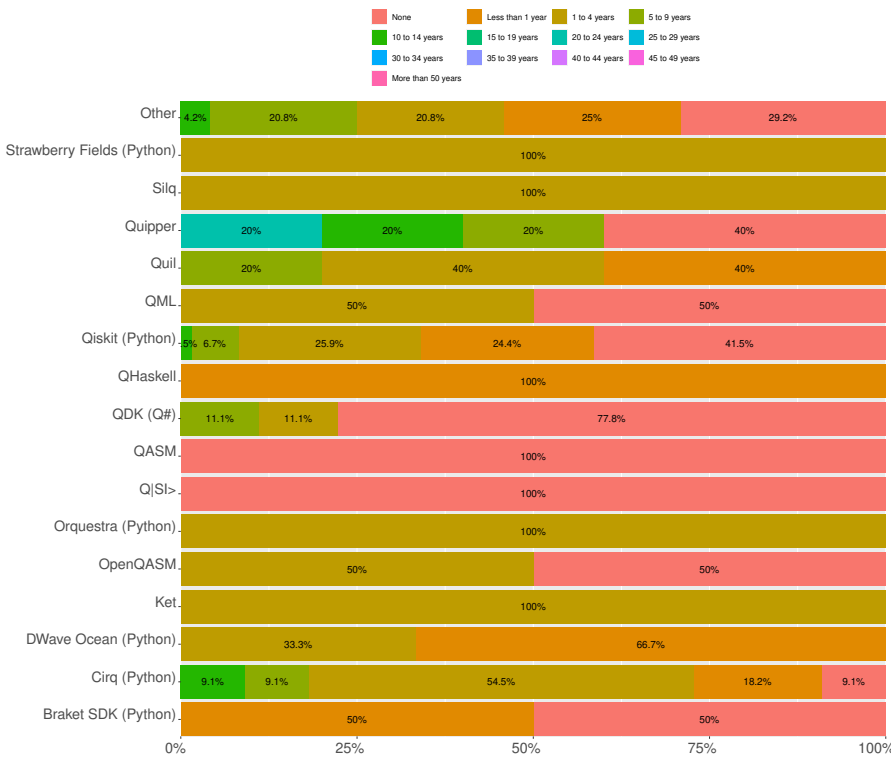

Figure 6.22: Relation between the primary quantum programming language chosen by the participants and their professional experience by percentage.

Summary RQ7: 42.8% of the participants answered that there are too many quantum programming languages. In contrast, 18.3% answered no, and 38.9% did not answer this question for different reasons like they did not know or did not have the knowledge to answer it.

6.8 Summary

In this chapter, we presented the results and answers to each proposed research question in this thesis. Regarding the RQ1, which aims to identify who is using quantum programming languages, the results showed that most participants are male (almost 90.0%); 58.2% have between 25 and 44 years old; 63.0% have master's or doctoral degrees; 86.2% have more than five years of experience using classical programming languages. About RQ2, to identify how the quantum programming languages are being used, the results showed that 42.8% of the participants use quantum programming languages for research, 34.6% use them because they like to learn new quantum languages, 16.4% use them for work, and 6.3% use them for other tasks. The study showed that Qiskit (Python) is the most used quantum programming language, used by more than 85.0% of the participants that responded to the survey, and Python is the most used classical language to build quantum programs, answering RQ3.

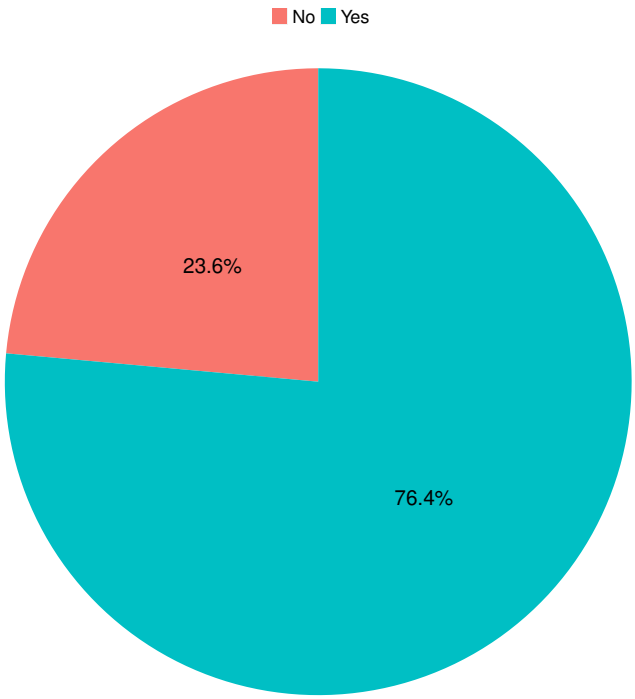

Figure 6.23: Identify if the participants perform tests in their quantum programs by percentage.

Participants mentioned several reasons for choosing a quantum programming language and responded to the RQ4. The most mentioned ones were open-source, based on an existing programming language, available documentation and examples, an active community, and available features. Cirq with 25.5% was the most chosen quantum programming language by the participants to work in the near future, answering RQ5.

Regarding RQ6, the main reasons to propose a new language are the need for a more mature, standardized, high-level quantum programming language with new features (e.g., handling error correction in real devices). Regarding RQ7, most participants answered that there are too many quantum programming languages (42.8%), 38.9% did not know, and 18.3% answered that there are not too many quantum programming languages.

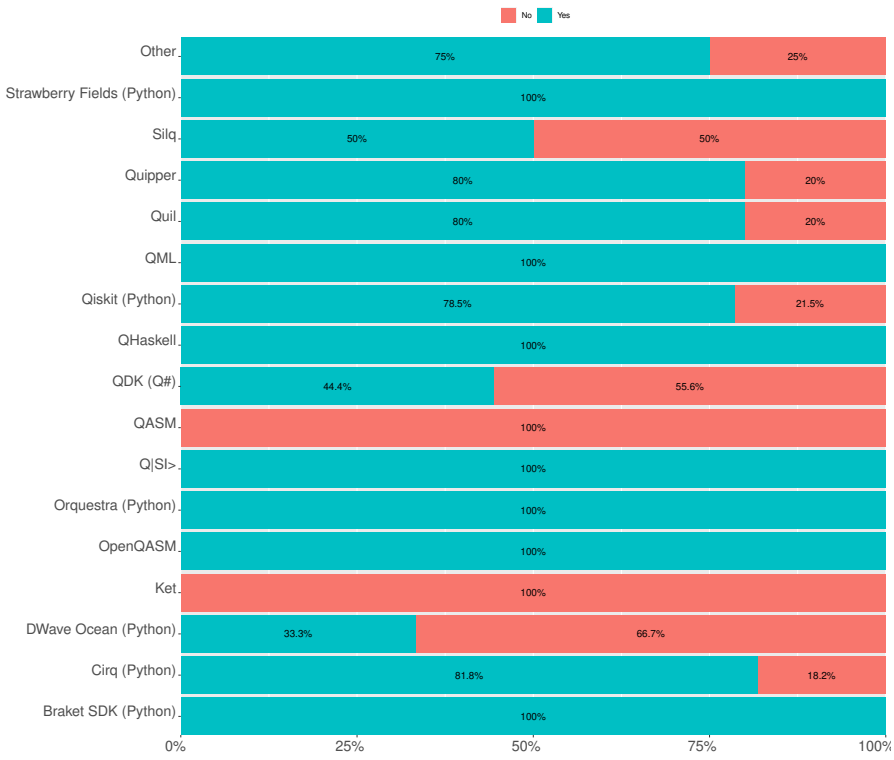

Figure 6.24: Relation between the primary quantum programming language chosen by the participants and if they test their quantum programs by percentage.

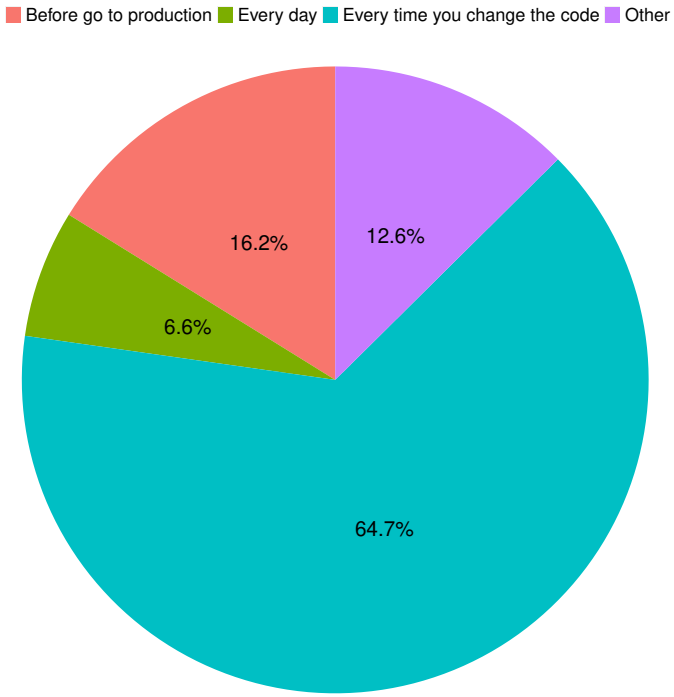

Figure 6.25: The frequency that the participant tests their quantum programs by percentage.

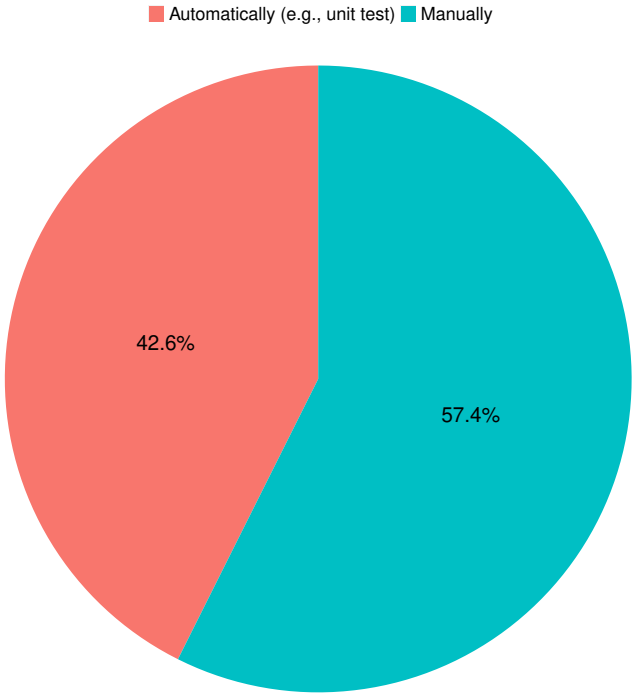

Figure 6.26: Identify if the participants use automatic or manual tests by percentage.

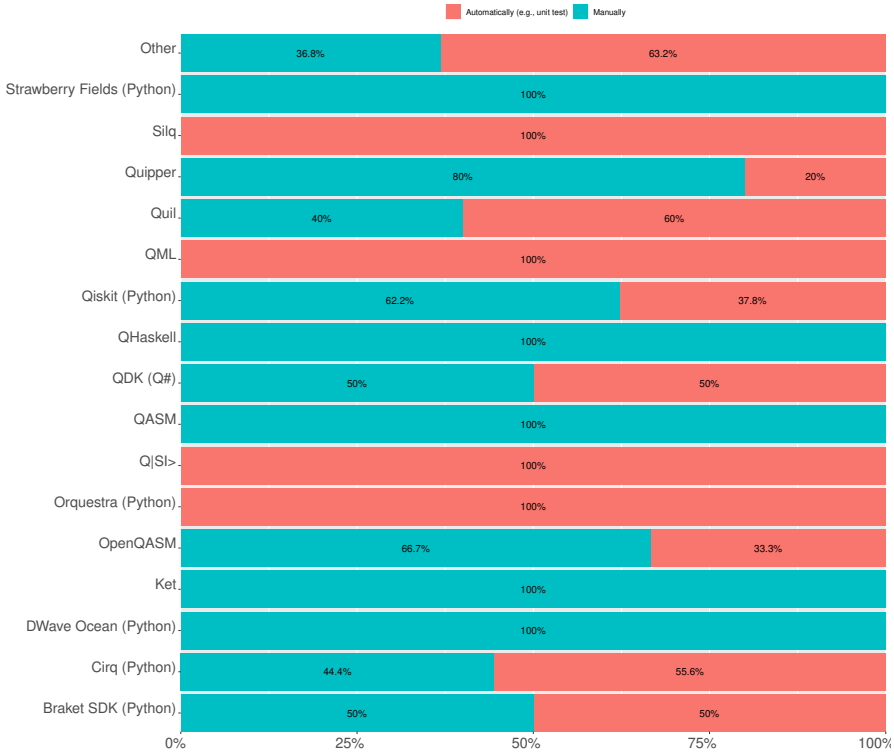

Figure 6.27: Relation between the primary quantum programming language chosen by the participants and how they test their quantum programs by percentage.

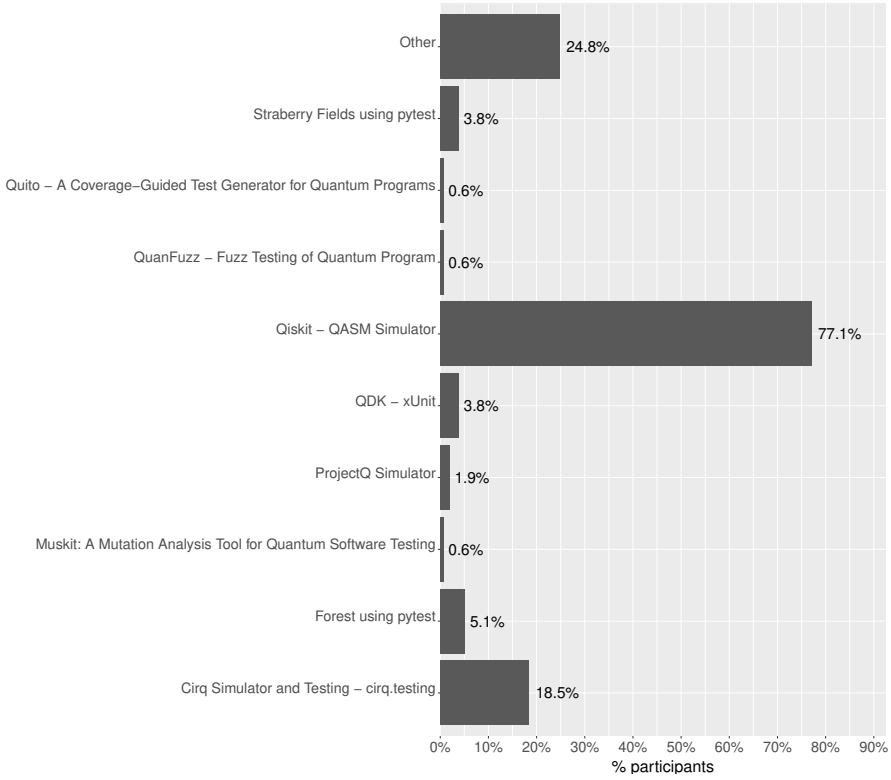

Figure 6.28: Show what the most used tools to test quantum programs are by percentage.

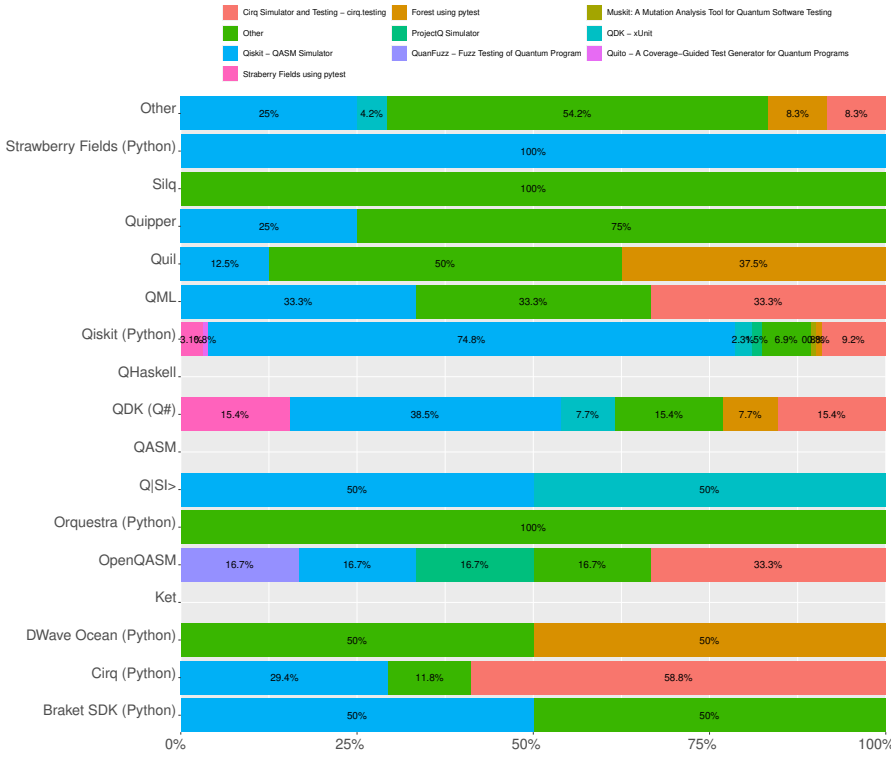

Figure 6.29: Relation between the primary quantum programming language chosen by the participants and the tool they use for testing by percentage.

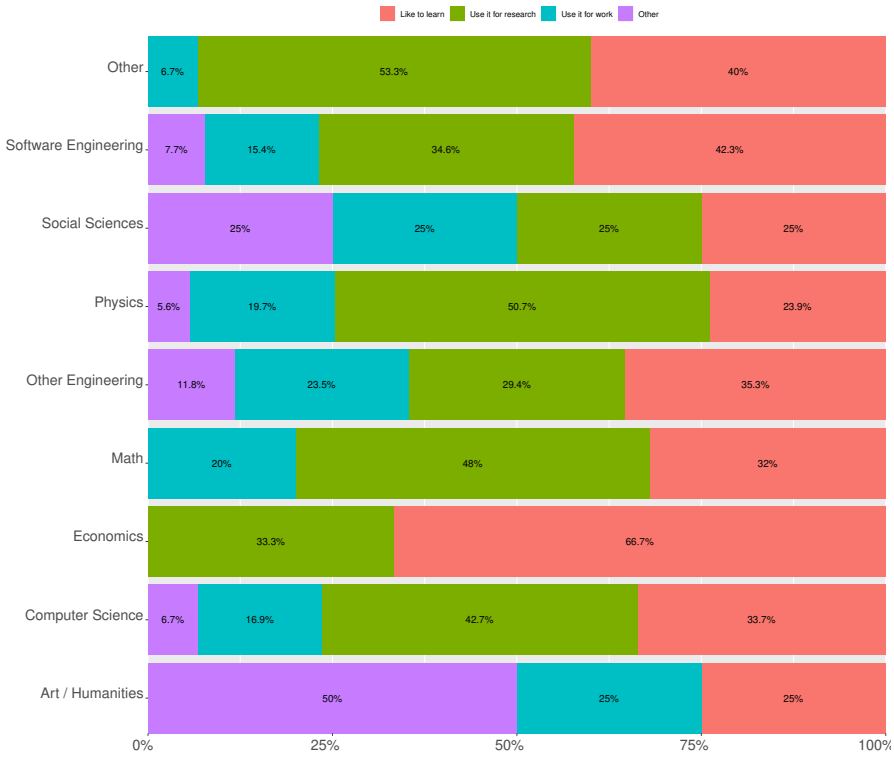

Figure 6.30: Relation between the work field of the participants and the reason they use the quantum programming languages by percentage.

Figure 6.31: Relation between the current job of the participants and the reason they are using the quantum programming language by percentage.

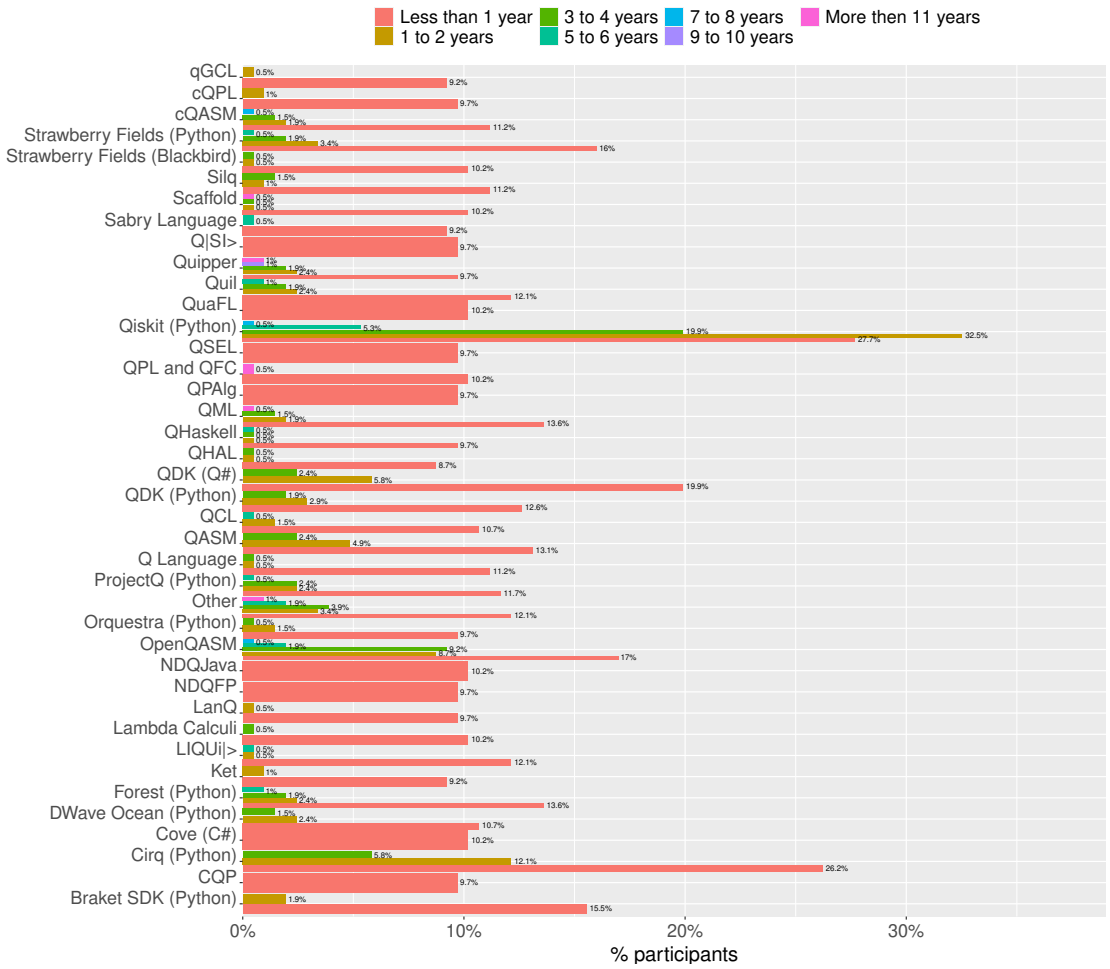

Figure 6.32: Show which quantum programming languages the participants use and how long by the percentage. (See Table 6.1 for absolute numbers.)

Quantum Programming Language	Less than 1 year	1 to 2 years	3 to 4 years	5 to 6 years	7 to 8 years	9 to 10 years	More then 11 years	Total
Braket SDK (Python)	32	4	0	0	0	0	0	36
Cirq (Python)	54	25	12	0	0	0	0	91
Cove (C#)	21	0	0	0	0	0	0	21
cQASM	23	4	3	0	1	0	0	31
CQP	20	0	0	0	0	0	0	20
cQPL	20	2	0	0	0	0	0	22
DWave Ocean (Python)	22	5	3	0	0	0	0	30
Forest (Python)	28	5	4	2	0	0	0	39
Ket	19	2	0	0	0	0	0	21
λ_q	21	0	1	0	0	0	0	22
LanQ	20	1	0	0	0	0	0	21
$LIQUi\|\rangle$	25	1	0	1	0	0	0	27
NDQFP	20	0	0	0	0	0	0	20
NDQJava	21	0	0	0	0	0	0	21
OpenQASM	35	18	19	4	1	0	0	77
Orquestra (Python)	20	3	1	0	0	0	0	24
ProjectQ (Python)	24	5	5	1	0	0	0	35
Q Language	23	1	1	0	0	0	0	25
$Q\|SI\rangle$	20	0	0	0	0	0	0	20
QASM	27	10	5	0	0	0	0	42
QCL	22	3	0	1	0	0	0	26
QDK (Python)	26	6	4	0	0	0	0	36
QDK (Q#)	41	12	5	0	0	0	0	58
qGCL	19	1	0	0	0	0	0	20
QHAL	18	1	1	0	0	0	0	20
QHaskell	20	1	1	1	0	0	0	23
Qiskit (Python)	57	67	41	11	1	0	0	177
QML	28	4	3	0	0	0	1	36
QPAlg	20	0	0	0	0	0	0	20
QPL and QFC	21	0	0	0	0	0	1	22
QSEL	20	0	0	0	0	0	0	20
QuaFL	21	0	0	0	0	0	0	21
Quil	25	5	4	2	0	0	0	36
Quipper	20	5	4	0	0	2	2	33
Sabry Language	19	0	0	1	0	0	0	20
Scaffold	21	1	1	0	0	0	1	24
Silq	23	2	3	0	0	0	0	28
Strawberry Fields (Blackbird)	21	1	1	0	0	0	0	23
Strawberry Fields (Python)	33	7	4	1	0	0	0	45
Other	25	7	8	4	0	0	2	46

Table 6.1: Number of participants per quantum programming language by usage time.

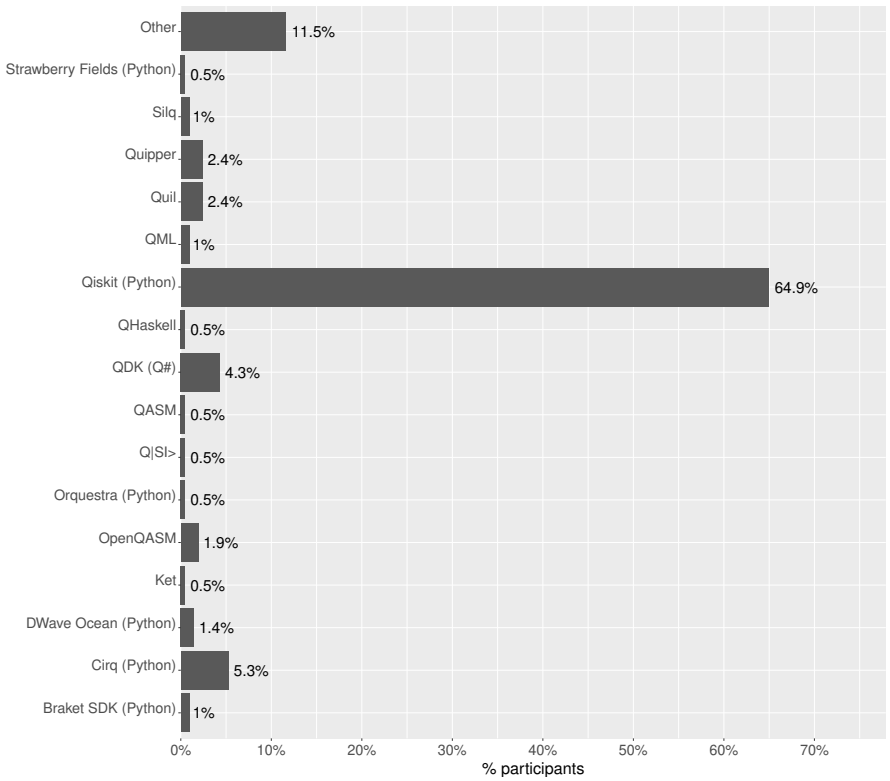

Figure 6.33: The primary quantum programming language used by the participants by percentage.

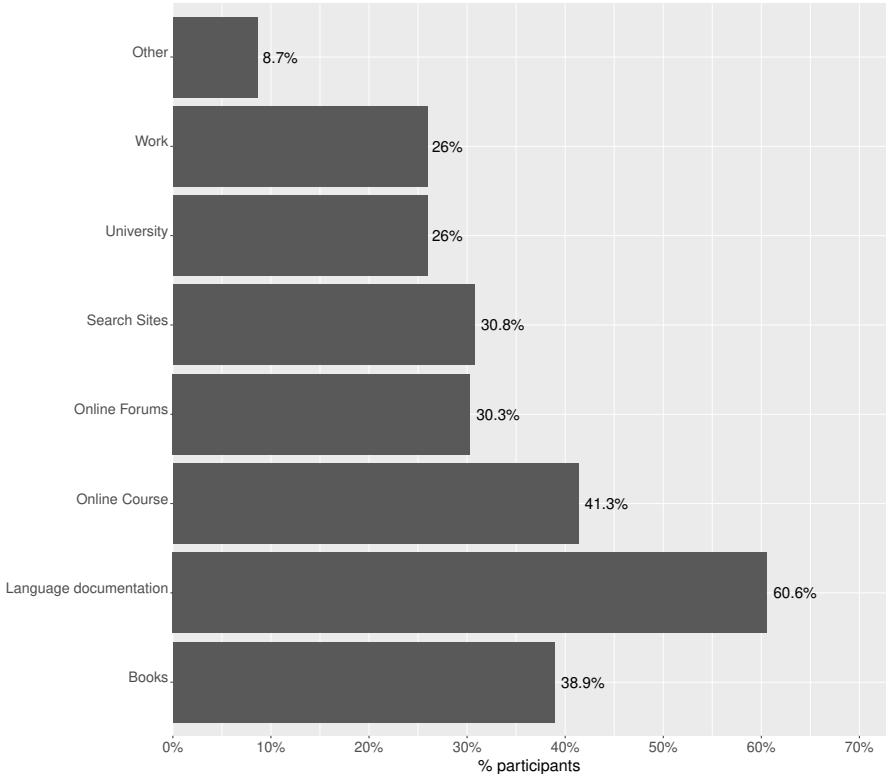

Figure 6.34: How the participants learned quantum programming languages by percentage.

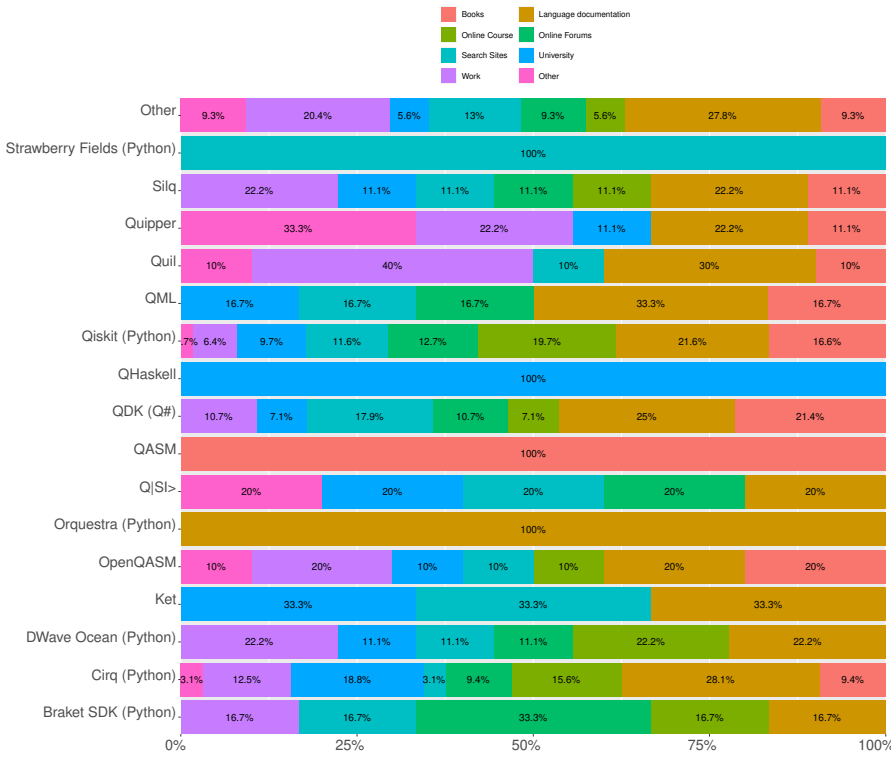

Figure 6.35: Relation between the primary quantum programming language chosen by the participants and how they learned the language by percentage.

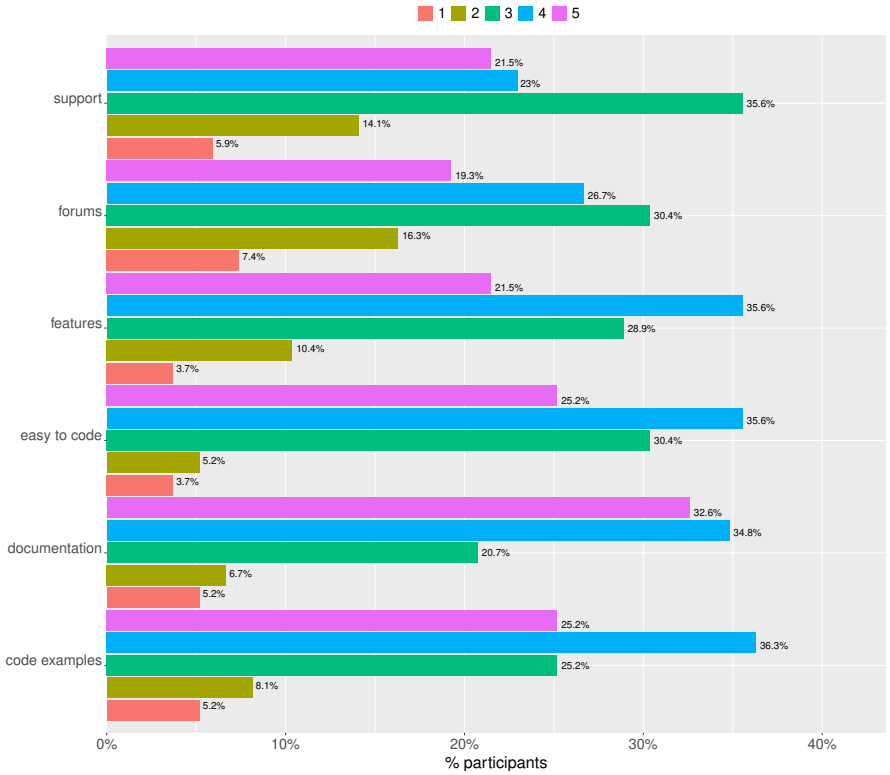

Figure 6.36: The rate in terms of ease (e.g., support, forums, features, easy to code, documentation, code examples) of Qiskit (participant's primary quantum programming language most chosen) by percentage.

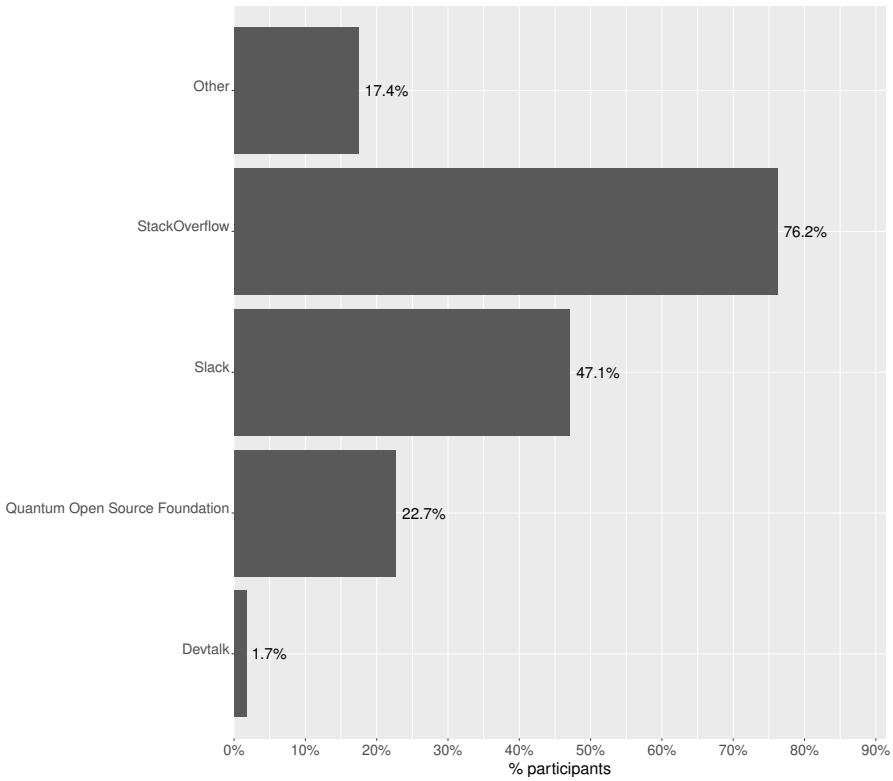

Figure 6.37: The most used forums to ask questions on quantum computing by percentage.

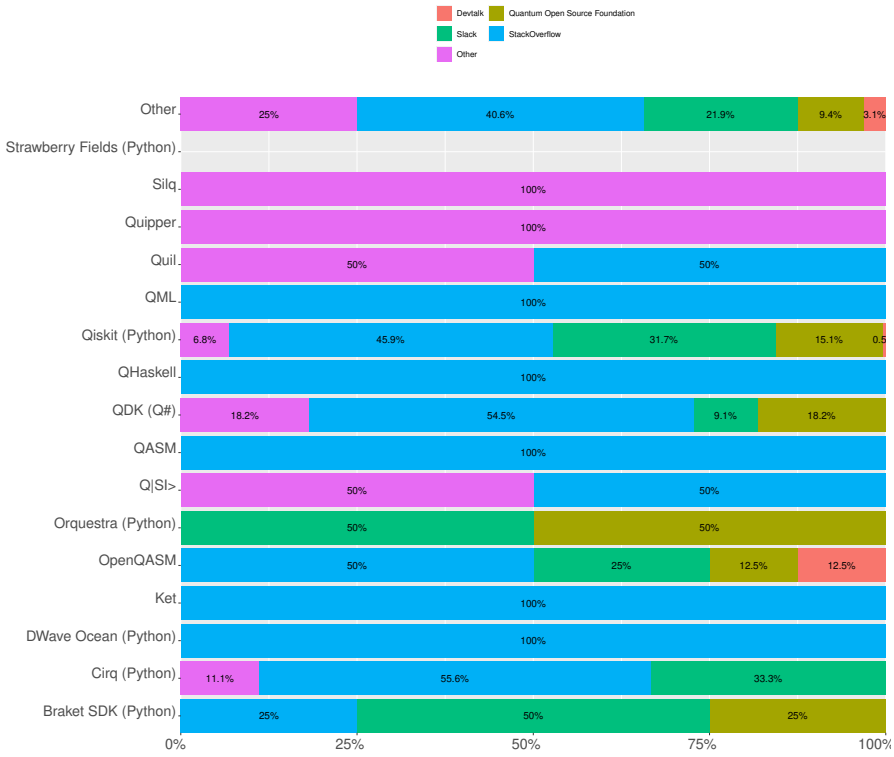

Figure 6.38: Relation between the primary quantum programming language chosen by the participants and the forum used by percentage.

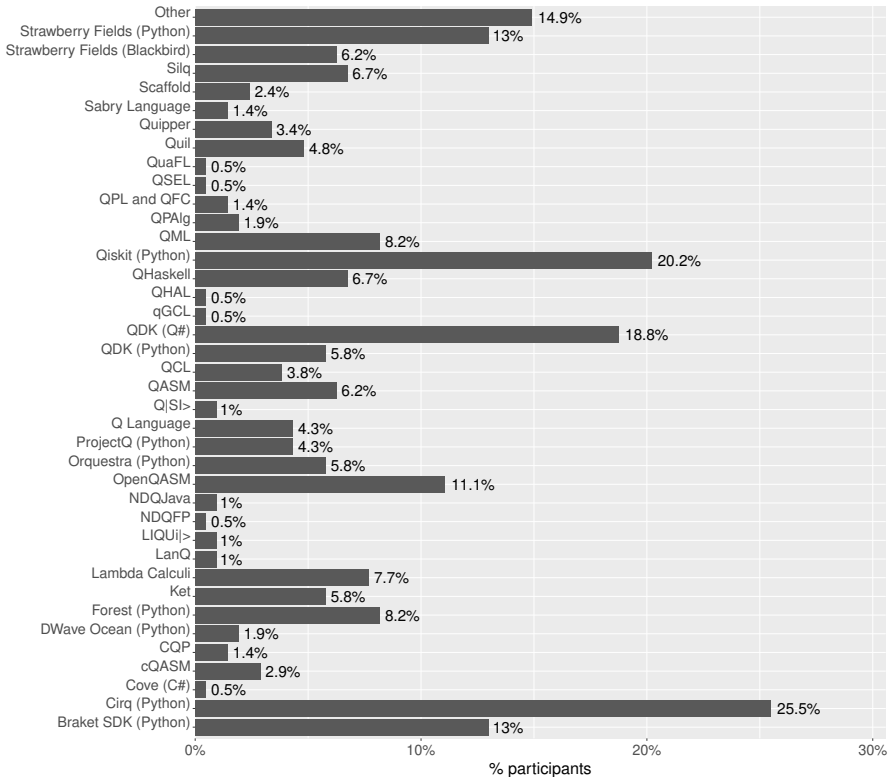

Figure 6.39: Which is the most like quantum programming language to be used in the future by percentage.

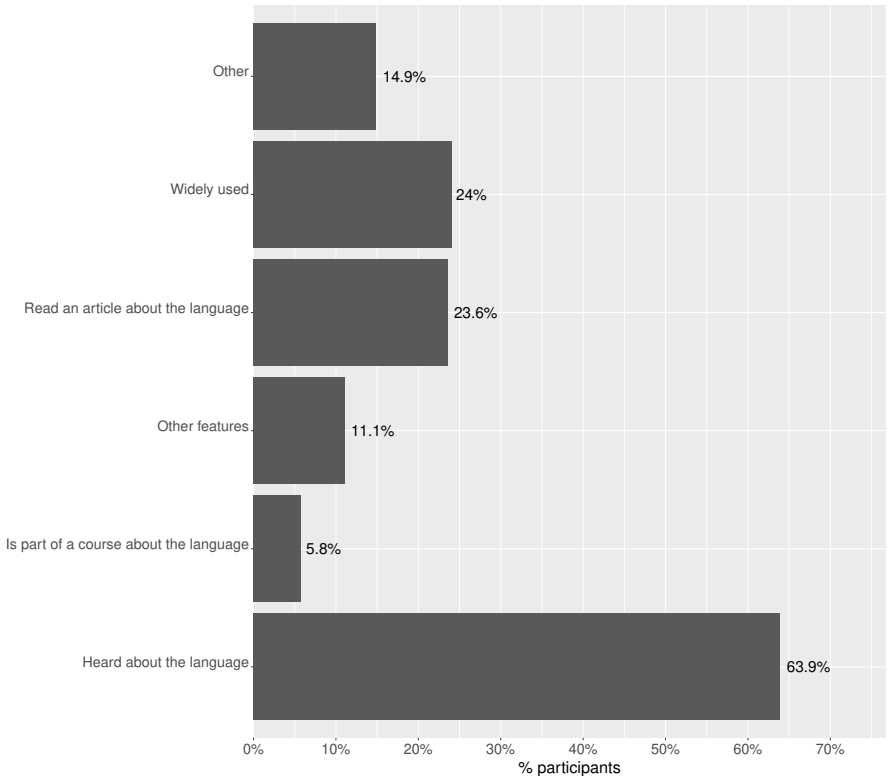

Figure 6.40: Reason why the participant wants to work with the quantum programming language by percentage.

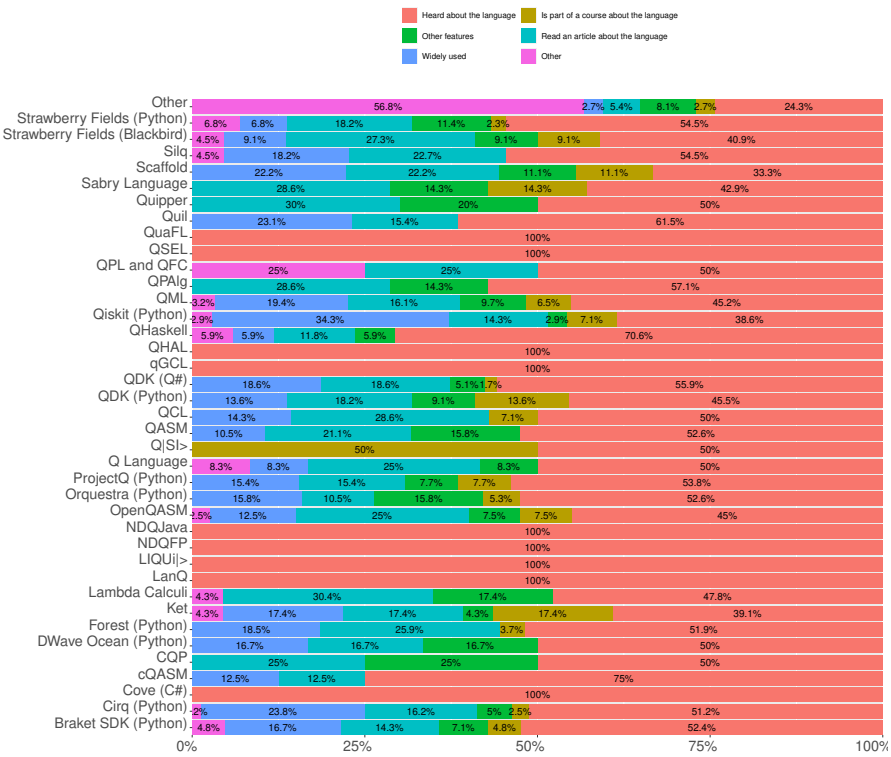

Figure 6.41: Relation between the primary quantum programming language the participants would like to work or try in the near future and why by percentage.

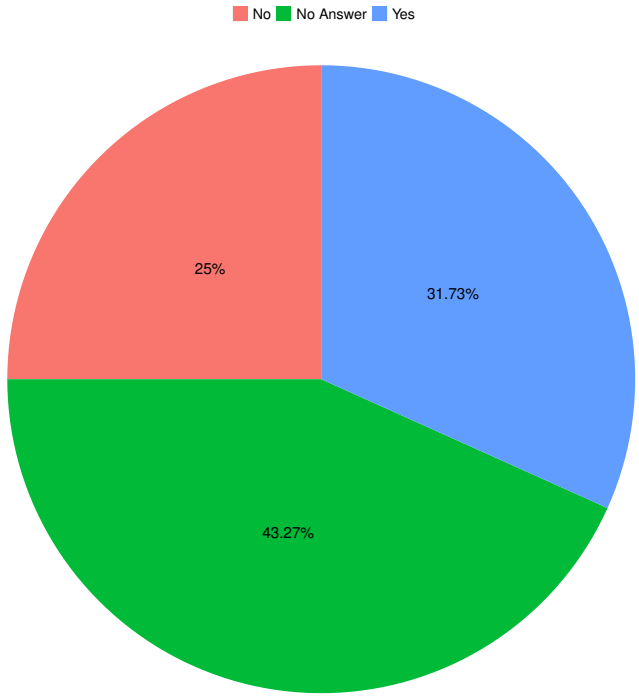

Figure 6.42: Participant's opinion if they think developing another quantum programming language by percentage is necessary.

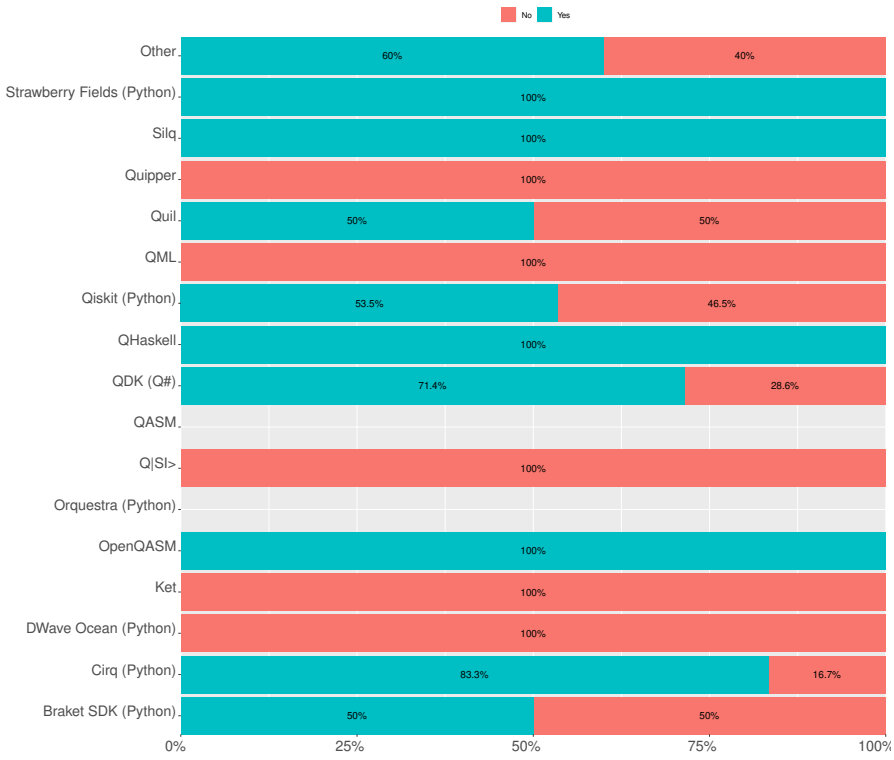

Figure 6.43: Relation between the primary quantum programming language chosen by the participants and their opinion regarding if another quantum programming language is needed by percentage.

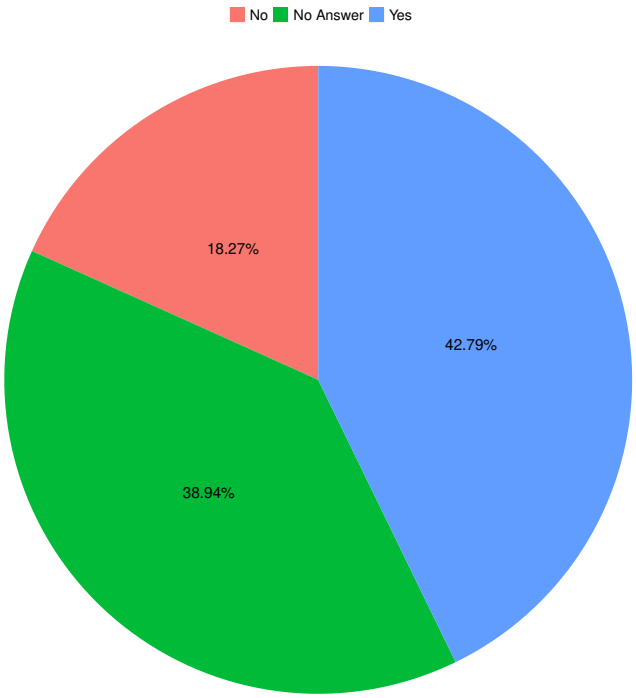

Figure 6.44: Participant's opinion about the existence of several quantum programming languages by percentage.

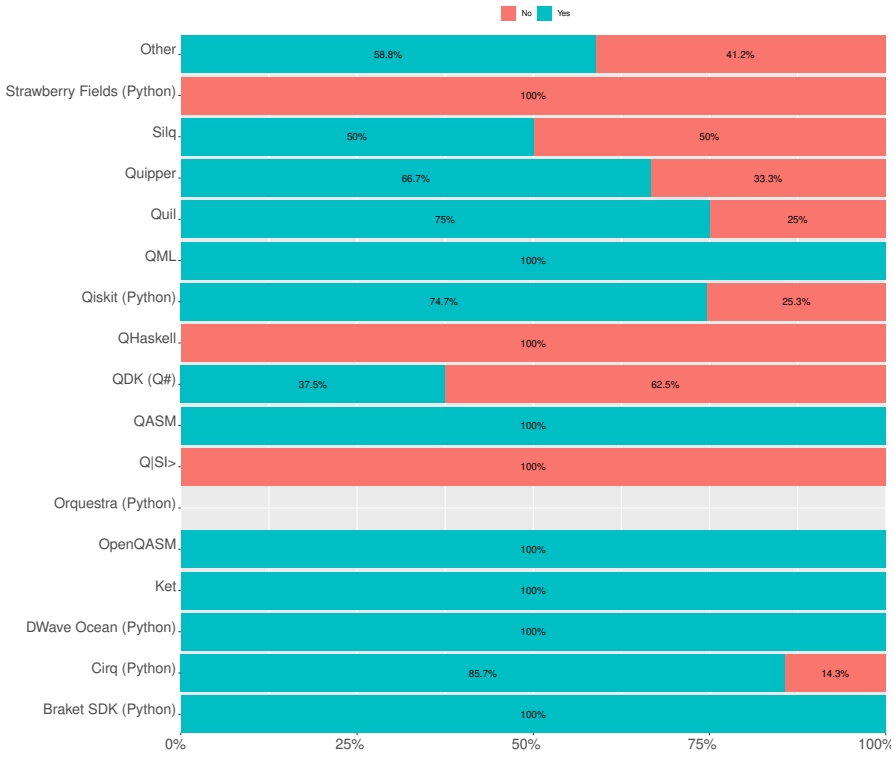

Figure 6.45: Relation between the primary quantum programming language chosen by the participants and their opinion regarding if there are too many quantum programming languages by percentage.

Chapter 7

Implications for New/Existing Quantum Programming Languages

The purpose of this section is to describe the implications our study might have in developing new quantum programming languages or updating existing ones.

7.1 Basic Characteristics

Quantum programming languages might have some basic features that are widely adopted, such as:

- **Generality**. Languages can be used for many problems.
- **Completeness**. Possibility to develop any quantum algorithm using the language.
- **Extensibility**. New features can be easily added to the languages, as new features will be needed with the evolution of quantum computers.
- **Expressivity**. Provide high-level structures.
- **Easy and standardized syntax**. Use the familiar syntax of programming languages, like C and Python.
- **Open source**. Developers prefer when a single vendor does not wield too much control over a language. Open source assures them that the code they write today cannot be made obsolete tomorrow by changes to the tools, language, or license.
- **Portability**. Possibility to use the language on multiple platforms, meaning that a quantum program might be easy to transfer from which they are developed to another hardware.
- **Classical Data Types**. Provide different classical data types (int, real, boolean, string).
- **Quantum Data Types**. Provide qubit registers to work with quantum data types.
- **Quantum Functions**. Functions to handle quantum registers.
- **Quantum Memory Management**. Provide memory management for quantum variables.

7.2 Document, examples, and community support

The purpose of documentation is to explain to a human reader how a program works so that it can be successfully adapted, either to meet the changing requirements of its users, to improve it in light of increasing knowledge, or to remove latent errors and oversights. When developing a new language, the authors must provide many tutorials, course materials, and resources so that developers can learn how to use the language to develop their algorithms. This documentation must be available so that developers can consult them whenever necessary. Code examples also benefit developers, especially when learning a new language. To encourage the use of the language, it must have an extensive and active community, allowing developers to exchange information with each other, and helping them to resolve doubts and issues encountered in their codes.

The quantum programming languages might also have corporate sponsorship so that the language supports its development by a company and that there are always developers working to evolve and support the quantum programming language. Big technology companies can put their weight behind a language by providing enough resources and a certain trust. Other companies and developers must be assured that the language they base their code on in a couple of years is still relevant.

Language documentation and community support are not just necessary for quantum programming languages, as stated by Hope [15], this is already a well-known case for classical programming languages, being a success factor for its wide use.

7.3 Run in real quantum computerss

The fact that quantum computing is an innovative area with much potential in the future is one of the big reasons why many developers are willing to learn new programming languages. That is why quantum programs might have the possibility to run on a real quantum computer instead of being used only in simulators so that developers are motivated to learn and use the quantum programming language to develop their programs. Furthermore, the companies might give the developers free access to their quantum computers to test their quantum programs; thus, as many developers use the quantum programming language as possible.

7.4 Build on top of classical programming language

A language can solve many problems and offer many features, but if it is not easy to use and learn, it is hard to use. If new languages support familiar programming styles that developers are used to, chances are developers will more easily use them because they tend to adopt languages with familiar syntax. We can see in Figures 6.10 and 6.32 that

the quantum programming languages most used by developers are based on the classical languages most used by them.

A quantum programming language based on a well-known classical language makes it easy for developers to build their code and not limit them only to the features of the quantum programming language. However, they can also use all of the features of the classical language and its libraries, making it possible to start learning and developing quickly and easily.

7.5 Features and tools

A quantum programming language, to be successful, might provide as many features and tools as possible for the developers to build a quantum program. For example:

- IDEs tailed for quantum programs, having a set of common development tools in a single graphical interface, such as a source code editor, local compilation automation, and tools for testing and debugging.
- Simulators.
- Libraries for quantum programming (e.g., standard quantum programs, quantum machine learning, and others).

An example of one functionality is the Qiskit draw function, which translates the source code into an image of the quantum circuit. This feature makes a quantum physics person who does not understand programming to be able to interact with the developer, and both understand the same code as seen in different ways.

7.6 Consult developer's needs

A last and essential implication is to consult the developers of quantum programs to understand their needs, what are the main features they need, what are the main problems encountered in the languages they currently use, and what can be developed to improve the language. The language users might always be consulted before creating or extending a quantum programming language because they know what they need to write better algorithms and what is missing in the language they are currently using.

7.7 Summary

In this chapter, we made some recommendations for further development of quantum programming languages, such as: quantum programming languages might have some basic features that are widely adopted (e.g., generality, completeness, extensibility, and

expressivity); when developing a new language, the authors might provide many tutorials, course materials, and resources; an extensive and active community; run on a real quantum computer instead of being used only in simulators; build on top of a classical programming language; provide as many features and tools as possible for the developers to build a quantum program; consult developer's needs.

Chapter 8

Conclusions and Future Work

Quantum programming language is an evolving area, and the number of languages developed is growing and expected to evolve with the development of real quantum computers. In this work, an exploratory study was conducted on the usage of quantum programming languages.

8.1 Conclusions

First, extensive research was carried out about the state of the art of quantum programming languages. They were classified according to their paradigm, and an overview was elaborated. A study on the history of quantum programming languages was made to trace their roots through the ages. We hope this list of languages gives credit to the authors and proves valuable to the growing quantum programming languages community. Secondly, a survey was created in this study and answered by 251 participants to help answer the research questions proposed on the usage of quantum programming languages. The results showed the profile of the participants who are using the languages, how they are being used, which language is most used, which language tends to be used in the near future, and the opinion of the participants about the number of quantum programming languages available, and the need to create new languages. The results reported that quantum languages are mainly used for learning and research, and that Python, a classical programming language, is the most widely used language to develop quantum programs. Finally, implications for the future development of a quantum programming language were given that may be useful for those who will develop new languages or update existing ones.

8.2 Future Work

For future work, we plan to extend the study with additional quantum programming languages that were not aware of when the survey was conducted. New questions, which were only identified after analyzing the answers of the survey, could augment our results'

value, e.g.,: what type of quantum programs and/or algorithms are developers developing with quantum programming languages, what kind of projects are developers working on. These and other questions might further shed light on the day-to-day usage of quantum programming languages and the developement of quantum programs.

Appendix A

Appendix

A.1 Survey questions

This appendix describes, by group, the questions that were asked in the survey, the reason for each question, and the type/domain of each answer.

#	Question	Reason	Answer Type	Possible Answers
		Group 1		
1	Have you ever used any Quantum Programming Language?	Identify if the participant worked with quantum programming languages and can answer the rest of the survey	Multiple Choice	Yes; No
		Group 2		
2	What is your age?	This question was chosen to identify the participants of this survey demographically.	Dropdown	Under 18 years old; 18-24 years old; 25-34 years old; 35-44 years old; 45-54 years old; 55-64 years old; 65 years or older; Prefer not to say
3	Where do you live? (Country)	Identify where the participants are geographically concentrated.	Dropdown	Brazil; Portugal; Spain; etc
4	Which of the following describe you?	Identify the gender of the participants.	Multiple Choice	Man; Woman; Non-binary, genderqueer, or gender non-conforming; Prefer not to say; Other
		Group 3		
5	How many years have you been coding?	Assess the experience and education of the participants in terms of coding.	Dropdown	Less than 1 year; 1 to 4 years; 5 to 9 years; 10 to 14 years; 15 to 19 years; 20 to 24 years; 25 to 29 years; 30 to 34 years; 35 to 39 years; 40 to 44 years; 45 to 49 years; More than 50 years
6	How many years have you coded professionally (as a part of your work)?	Assess the professional experience of the participants.	Dropdown	None; Less than 1 year; 1 to 4 years; 5 to 9 years; 10 to 14 years; 15 to 19 years; 20 to 24 years; 25 to 29 years; 30 to 34 years; 35 to 39 years; 40 to 44 years; 45 to 49 years; More than 50 years
7	How did you learn to code? Select all that apply.	Assess the education of the participants.	Checkboxes	Books / Physical media; Coding Bootcamp; Colleague; Friend or family member; Online Courses or Certification; Online Forum; Other online resources (videos, blogs, etc.); School; Other

8	What are the most used programming, scripting, and markup languages you have used? Select all that apply.	Identify the languages that the participant has used.	Checkboxes	Assembly; Bash C; Classic Visual Basic; COBOL C++; C#; Delphi/Object Pascal; Fortran; F#; Go; Groovy; Haskell; Java; JavaScrpit; Julia; Lisp; Matlab; ML; Objective-C; Pascal; Perl; pGCL; PHP; PowerShell; Prolog; Python; Ruby; SQL; Standard ML; Swift; Visual Basic; Visual C++; Other
9	What is your level of knowledge in Quantum Physics?	Assess level of education in terms of knowledge in quantum physics.	Multiple Choice	0 (no knowledge); 1 (novice); 2; 3; 4; 5 (expert)
10	Where did you learn Quantum Physics?	Assess the education of the participants in terms of learning quantum physics.	Checkboxes	Books; Online Course; Search Sites; University; Work; Other
11	Which of the following best describes the highest level of education that you have completed?	Identify the formal education of the participants.	Multiple Choices	Primary/elementary school; Secondary school (e.g., American high school, German Realschule or Gymnasium, etc.); Some college/university study without earning a degree; Associate degree (A.A., A.S., etc.); Bachelor's degree (B.A., B.S., B.Eng., etc.); Master's degree (M.A., M.S., M.Eng., MBA, etc.); Professional degree (JD, MD, etc.); Other doctoral degrees (Ph.D., Ed.D., etc.); Other
12	If you have completed a major, what is the subject?	Assess the work field of the participants.	Checkboxes	Art / Humanities; Computer Science; Economics; Software Engineering; Math; Other Engineering; Physics; Social Sciences; Other
13	Which of the following describes your current job? Please select all that apply.	Identify the roles of the participants.	Checkboxes	Academic researcher; Architect; Business Analyst; CIO / CEO / CTO; DBA (Database Administrator); Data Analyst / Data Engineer/ Data Scientist; Developer Advocate; Developer / Programmer / Software Engineer; DevOps Engineer / Infrastructure Developer; Instructor / Teacher / Tutor; Marketing Manager; Product Manager; Project Manager; Scientist / Researcher; Student; Systems Analyst; Team Lead; Technical Support; Technical Writer; Tester / QA Engineer; UX / UI Designer; Other
		Group 4		
14	Where and how did you learn Quantum Programming Languages?	Assess the education of the participants in terms of learning quantum programming languages.	Checkboxes	Books; Language documentation; University; Online Course; Online Forums; Search Sites; Work; Other
15	How many years have you been coding using Quantum Programming Languages in total?	Assess the experience and education of the participants in using quantum programming languages.	Dropdown	Less than 1 year; 1 to 4 years; 5 to 9 years; 10 to 14 years; 15 to 19 years; 20 to 24 years; 25 to 29 years; 30 to 34 years; 35 to 39 years; 40 to 44 years; 45 to 49 years; More than 50 years
16	How many years have you coded professionally using Quantum Programming Languages (as a part of your work)?	Assess the professional experience of the participants regarding quantum programming languages.	Dropdown	None; Less than 1 year; 1 to 4 years; 5 to 9 years; 10 to 14 years; 15 to 19 years; 20 to 24 years; 25 to 29 years; 30 to 34 years; 35 to 39 years; 40 to 44 years; 45 to 49 years; More than 50 years

17	What Quantum Programming Languages have you been using, and for how long?	Assess which quantum programming languages the participants use and how long.	Multiple Choice Grid	Rows (Blackbird; Braket SDK; Cirq; Cove; cQASM; CQP (Communication Quantum Processes); cQPL; Forest; Ket; LanQ; $LIQUi\|\rangle$; NDQFP; NDQJava; Ocean Software; OpenQASM; Orquestra; ProjectQ; Q Language; QASM (Quantum Macro Assembler); QCL (Quantum Computation Language); QDK (Quantum Development Kit); QHAL; Qiskit; qGCL; QHaskell; QML; QPAlg (Quantum Process Algebra); QPL and QFC; QSEL; QuaFL (DSL for quantum programming); Quil; Quipper; Q#; $Q\|SI\rangle$; Sabry's Language; Scaffold; Silq; Strawberry Fields; λ_q (Lambda Calculi); Other) and Columns (Less than 1 year; 1 to 2 years; 3 to 4 years; 5 to 6 years; 7 to 8 years; 9 to 10 years; More than 11 years)
18	Is there any other Quantum Programming Language not listed that you have been using?	Identify other quantum programming languages not listed that the participant used.	Open Text	-
19	Which of the following is your primary Quantum Programming Languages?	Identify the most used quantum programming language by the participants.	Dropdown	Blackbird; Braket SDK; Cirq; Cove; cQASM; CQP (Communication Quantum Processes); cQPL; Forest; Ket; LanQ; $LIQUi\|\rangle$; NDQFP; NDQJava; Ocean Software; OpenQASM; Orquestra; ProjectQ; Q Language; QASM (Quantum Macro Assembler); QCL (Quantum Computation Language); QDK (Quantum Development Kit); QHAL; Qiskit; qGCL; QHaskell; QML; QPAlg (Quantum Process Algebra); QPL and QFC; QSEL; QuaFL (DSL for quantum programming); Quil; Quipper; Q#; $Q\|SI\rangle$; Sabry's Language; Scaffold; Silq; Strawberry Fields; λ_q (Lambda Calculi); Other
20	In terms of ease, rate your primary Quantum Programming Language.	Rate the main characteristics of the participant's favorite quantum programming languages.	Multiple Choice Grid	Rows (Features / functionalities of the language; Documentation available; Code examples; Several forums; Support (e.g., GitHub issues); Easy to code) and Columns (1; 2; 3; 4; 5)
21	Is there anything else you like the most in your primary Quantum Programming Language?	Assess the main characteristic that the participants like in their primary quantum programming language.	Open Text	-
22	Is there anything else you do not like in your primary Quantum Programming Language?	Assess the main characteristic that the participants do not like in their primary quantum programming language.	Open Text	-
23	Which forums, e.g., to ask for help, search for examples, do you use? (if any)	Evaluate the most used forums to ask questions on quantum computing.	Checkboxes	Devtalk; Quantum Open Source Foundation; Slack; StackOverflow; Other

24	Which Quantum Programming Languages would you like to work or try in the near future?	Identify which is the most like quantum programming language to be used in the future.	Checkboxes	Blackbird; Braket SDK; Cirq; Cove; cQASM; CQP (Communication Quantum Processes); cQPL; Forest; Ket; LanQ; $LIQUi\|\rangle$; NDQFP; NDQJava; Ocean Software; OpenQASM; Orquestra; ProjectQ; Q Language; QASM (Quantum Macro Assembler); QCL (Quantum Computation Language); QDK (Quantum Development Kit); QHAL; Qiskit; qGCL; QHaskell; QML; QPAlg (Quantum Process Algebra); QPL and QFC; QSEL; QuaFL (DSL for quantum programming); Quil; Quipper; Q#; $Q\|SI\rangle$; Sabry's Language; Scaffold; Silq; Strawberry Fields; λ_q (Lambda Calculi); Other
25	Why would you like to work or try those languages?	The reason why the participant wants to work with the quantum programming language.	Checkboxes	Heard about the language; Is part of a course about the language; Read an article about the language; Widely used; Other features; Other
26	What challenges did you run into when choosing a Quantum Programming Language?	Assess the challenges the participant faces when choosing a quantum programming language.	Open Text	-
27	In your opinion, what makes learning Quantum Programming Languages important?	Assess why the participants want to learn quantum programming languages.	Open Text	-
		Group 5		
28	How do you use Quantum Programming Languages?	Assess how the participants use quantum programming languages.	Multiple Choice	Use it for work; Use it for research; Like to learn; Other
29	What type of tools do you think are necessary or missing to develop better and faster Quantum Programs?	Evaluate what tools are missing in the quantum programming languages.	Open Text	-
30	Do you test your Quantum Programs?	Identify if the participants perform tests.	Multiple Choice	Yes and No
31	How often do you test your Quantum Programs?	Evaluate the frequency that the participant tests their quantum programs.	Multiple Choice	Before go to production; Every day; Every time you change the code; Other
32	How do you test your Quantum Programs?	Identity if the participants use automatic or manual tests	Multiple Choice	Automatically (e.g., unit test); Manually

33	What tools do you use to test your Quantum Programs?	Identify what the most used tools to test quantum programs are.	Checkboxes	Cirq Simulator and Testing - cirq.testing (https://quantumai.google/cirq); Forest using pytest (https://github.com/rigetti/forest-software); MTQC - Mutation Testing for Quantum Computing (https://javpelle.github.io/MTQC/); Muskit: A Mutation Analysis Tool for Quantum Software Testing (https://ieeexplore.ieee.org/document/9678563); ProjectQ Simulator (https://arxiv.org/abs/1612.08091); QDiff - Differential Testing of Quantum Software Stacks (https://ieeexplore.ieee.org/abstract/document/9678792); QDK - xUnit (https://azure.microsoft.com/en-us/resources/development-kit/quantum-computing/); Qiskit - QASM Simulator (https://qiskit.org/); QuanFuzz - Fuzz Testing of Quantum Program (https://arxiv.org/abs/1810.10310); Quito - A Coverage-Guided Test Generator for Quantum Programs (https://ieeexplore.ieee.org/abstract/document/9678798); Strawberry Fields using pytest (https://strawberryfields.ai/); Other
		Group 6		
34	In your opinion, do you think there are too many or too few Quantum Programming Languages? Why?	Assess the opinion of the participants on why exists or not several quantum programming languages.	Open Text	-
35	In your opinion, do you think we would need yet another Quantum Programming Language in the near future? Why?	Assess if the participant thinks that it is necessary the development of another quantum programming language	Open Text	-

Table A.1: Survey proposed questions.

A.2 Social networks contacted for the survey

This appendix shows the information about the social networks contacted for the survey, such as name, type, link, members and any observations.

Name	Type	Link	# Members	Observation
Exploring Quantum Computing	Facebook	https://www.facebook.com/groups/525270931156832	2,900	
Quantum Computing Now	Facebook	https://www.facebook.com/groups/328231110942652	6,900	

Quantum Information and Quantum Computer Scientists of the World Unite	Facebook	https://www.facebook.com/groups/qinfo.scientists.unit	15,900	
Quantum AI	Facebook	https://www.facebook.com/groups/quantumai/?multi_permalinks=1398423033952553	5,200	
Quantum Computing	Facebook	https://www.facebook.com/groups/896233200461905/	25,200	
Quantum Open-Source Foundation	Linkedin	https://www.linkedin.com/company/qosf/	2,971	Does not allow posts from group members
Quantum Computing and Quantum Information	Linkedin	https://www.linkedin.com/groups/1416467/	10,159	Did not accepted group membership
Quantum Information Science	Linkedin	https://www.linkedin.com/groups/1172457/	3,147	Did not accepted group membership
Quantum Computing	Linkedin	https://www.linkedin.com/groups/3748642/	10,378	
European Quantum Computing Applications Community	Linkedin	https://www.linkedin.com/groups/9015002/	2,413	
Quantum Computing Technology	Linkedin	https://www.linkedin.com/groups/4139130/	2,352	
Quantum Computing and AI Professionals	Linkedin	https://www.linkedin.com/groups/12083423/	1,049	
Quantum programming	Linkedin	https://www.linkedin.com/groups/8979014/	13	Did not accepted group membership
Quantum Computing and Programming	Linkedin	https://www.linkedin.com/groups/7468626/	14	Did not accepted group membership
IBM Quantum Computing	Linkedin	https://www.linkedin.com/groups/12376868/	3,381	
Q# Community	Discord	https://discord.qsharp.community/	200	
@quantum_comput	Twitter	https://twitter.com/quantum_comput	3,008	
Quantum Open-Source Foundation	Slack	https://qosf.slack.com/	2,546	
Strawberry Fields Community	Slack	https://u.strawberryfields.ai/slack	1,638	
Quantum Computing Slack Community	Slack	https://quantum-computing.slack.com	519	
myQLM	Slack	https://myqlmworkspace.slack.com	73	
Quantum Foundations	Mailing List	quantum-foundations@maillist.ox.ac.uk	-	Did not accepted the e-mail
Quantum Announcements	Mailing List	quantum-announcements@cs.ox.ac.uk	-	
Quantum Computing Institute	Mailing List	qci-external@elist.ornl.gov	-	Did not accepted the e-mail
Quantum Computing StackExchange	Forums and Communities	https://quantumcomputing.stackexchange.com/	-	Message rejected by the forum administrator
Reddit Quantum Computing	Forums and Communities	https://www.reddit.com/r/QuantumComputing/	33,600	
Reddit Quantum	Forums and Communities	https://www.reddit.com/r/quantum/	39,100	
Researches e-mails	E-mails		155	
Developers e-mails	E-mails	GitHub quantum program repositories	1,242	

Table A.2: Social networks contact for the survey. Information was obtained in April 2022.

Bibliography

[1] Chris Bernhardt. *Quantum computing for everyone*. Mit Press, 2019.

[2] Sarah Moore. *How Much Power Does Quantum Computing Need?* https://www.azoquantum.com/Article.aspx?ArticleID=136. [Online; accessed December-2021].

[3] Princy A. J. *Quantum Computing: A Big Wave of Revolution in the Computing Arena*. https://www.researchdive.com/blog/what-is-quantum-computing-is-it-a-replacement-to-traditional-computing. [Online; accessed December-2021].

[4] Maciej Adamiak. *Machine learning — what I've learned when experimenting with quantum computing*. https://blog.softwaremill.com/machine-learning-what-ive-learned-when-experimenting-with-quantum-computing-2a76da1d6fee. [Online; accessed December-2021].

[5] Daphne Leprince-Ringuet. *A quantum computer just solved a decades-old problem three million times faster than a classical computer*. https://www.zdnet.com/article/a-quantum-computer-just-solved-a-decades-old-problem-three-million-times-faster-than-a-classical-computer/. [Online; accessed December-2021].

[6] ICT Reverse. *What are Quantum Computers and Why are They Important?* https://ictreverse.com/what-are-quantum-computers-and-why-are-they-important/. [Online; accessed December-2021].

[7] Bruce J MacLennan. *Principles of programming languages: design, evaluation, and implementation*. Holt, Rinehart & Winston, 1986.

[8] Ravi Sethi. *Programming languages concepts and constructs*. Addison Wesley Longman Publishing Co., Inc., 1996.

[9] David Thomas and Andrew Hunt. *The Pragmatic Programmer: your journey to mastery*. Addison-Wesley Professional, 2019.

[10] Alfred V. Aho. *Compilers: principles, techniques and tools*. Addison Wesley, 2006.

[11] HOPL. *Online Historical Encyclopaedia of Programming Languages*. https://hopl.info/. [Online; accessed March-2022]. 2020.

[12] GitHub. *Developer feedback helps steer GitHub Public Policy commitments*. https://octoverse.github.com/#developer-feedback-helps-steer-github-public-policy-commitments. [Online; accessed March-2022]. 2021.

[13] Vasyl Lagutin. *Why are there so many programming language*. https://www.freecodecamp.org/news/why-are-there-so-many-programming-languages/. [Online; accessed March-2022]. 2021.

[14] Matt Sherman. *Why are there so many programming language*. https://stackoverflow.blog/2015/07/29/why-are-there-so-many-programming-languages/. [Online; accessed March-2022]. 2015.

[15] Computer Hope. *Why are there so many programming language*. https://www.computerhope.com/issues/ch000569.htm. [Online; accessed March-2022]. 2020.

[16] Michael L. Scott. *Programming language Pragmatics*. http://www.cs.yorku.ca/~billk/cse3301_S06/lectures/cse3301_S06_july17_6slides.pdf. [Online; accessed March-2022]. 2017.

[17] Peter Selinger. "Towards a Quantum Programming Language". In: *Mathematical. Structures in Comp. Sci.* 14.4 (Aug. 2004), 527–586. ISSN: 0960-1295. DOI: 10.1017/S0960129504004256. URL: https://doi.org/10.1017/S0960129504004256.

[18] Peter Selinger. "A brief survey of quantum programming languages". In: *In Proceedings of the 7th International Symposium on Functional and Logic Programming*. Springer, 2004, pp. 1–6.

[19] E Knill. "Conventions for quantum pseudocode". In: (June 1996). DOI: 10.2172/366453. URL: https://www.osti.gov/biblio/366453.

[20] Jean-Yves Girard. *Between logic and quantic: a tract*. Tech. rep. MATHEMATICAL STRUCTURES IN COMPUTER SCIENCE, 2003.

[21] Samson Abramsky and Bob Coecke. "Physical Traces: Quantum vs. Classical Information Processing". In: *CoRR* cs.CG/0207057 (2002). URL: https://arxiv.org/abs/cs/0207057.

[22] Abbas Edalat. "An Extension of Gleason's Theorem for Quantum Computation". In: *International Journal of Theoretical Physics* 43.7/8 (2004), 1827–1840. ISSN: 0020-7748. DOI: 10.1023/b:ijtp.0000048823.93080.7e. URL: http://dx.doi.org/10.1023/B:IJTP.0000048823.93080.7e.

[23] B. Coecke and K. Martin. "A Partial Order on Classical and Quantum States". In: *New Structures for Physics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 593–683. ISBN: 978-3-642-12821-9. DOI: 10.1007/978-3-642-12821-9_10. URL: https://doi.org/10.1007/978-3-642-12821-9_10.

[24] Dominique Unruh. "Quantum programming languages". In: *Inform., Forsch. Entwickl.* 21 (Oct. 2006), pp. 55–63. DOI: 10.1007/s00450-006-0012-y.

[25] Simon J Gay. "Quantum programming languages: Survey and bibliography". In: *Mathematical Structures in Computer Science* 16.4 (2006), pp. 581–600.

[26] David Deutsch. "Quantum theory, the Church–Turing principle and the universal quantum computer". In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 400.1818 (1985), pp. 97–117.

[27] Bernhard Ömer. “A Procedural Formalism for Quantum Computing”. In: (Dec. 1998).

[28] S. Bettelli, Luciano Serafini, and Tommaso Calarco. “Toward an architecture for quantum programming”. In: *The European Physical Journal D* 25 (Mar. 2001). DOI: 10.1140/epjd/e2003-00242-2.

[29] J. W. Sanders and P. Zuliani. “Quantum Programming”. In: *Mathematics of Program Construction*. Ed. by Roland Backhouse and José Nuno Oliveira. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 80–99. ISBN: 978-3-540-45025-2.

[30] Philip Maymin. “The lambda-q calculus can efficiently simulate quantum computers”. In: *arXiv preprint quant-ph/9702057* (1997).

[31] André Van Tonder. “A lambda calculus for quantum computation”. In: *SIAM Journal on Computing* 33.5 (2004), pp. 1109–1135.

[32] T. Altenkirch and J. Grattage. “A functional quantum programming language”. In: *20th Annual IEEE Symposium on Logic in Computer Science (LICS’ 05)*. 2005, pp. 249–258. DOI: 10.1109/LICS.2005.1.

[33] Simon J Gay and Rajagopal Nagarajan. “Communicating quantum processes”. In: *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming languages*. 2005, pp. 145–157.

[34] Philippe Jorrand and Marie Lalire. “Toward a quantum process algebra”. In: *Proceedings of the 1st Conference on Computing Frontiers*. 2004, pp. 111–119.

[35] Donald A. Sofge. “A Survey of Quantum Programming Languages: History, Methods, and Tools”. In: *Second International Conference on Quantum, Nano and Micro Technologies (ICQNM 2008)*. 2008, pp. 66–71. DOI: 10.1109/ICQNM.2008.15.

[36] Jean-Yves Girard. “Linear logic”. In: *Theoretical computer science* 50.1 (1987), pp. 1–101.

[37] Devon Rojas. “The Modern State of Quantum Programming Language”. In: (2019).

[38] Sunita Garhwal, Maryam Ghorani, and Amir Ahmad. “Quantum programming language: A systematic review of research topic and top cited languages”. In: *Archives of Computational Methods in Engineering* 28.2 (2021), pp. 289–310.

[39] Manuel De Stefano, Fabiano Pecorelli, Dario Di Nucci, Fabio Palomba, and Andrea De Lucia. “Software Engineering for Quantum Programming: How Far Are We?” In: *arXiv preprint arXiv:2203.16969* (2022).

[40] QWA team member Alba Cervera-Lierta and researcher at Quantic group. *Quantum Computing languages landscape*. https://medium.com/@quantum_wa/quantum-computing-languages-landscape-1bc6dedb2a35. [Online; accessed Dezember-2021]. 2018.

[41] Evandro Chagas Ribeiro Da Rosa and Rafael De Santiago. “Ket Quantum Programming”. In: *J. Emerg. Technol. Comput. Syst.* 18.1 (Oct. 2021). ISSN: 1550-4832. DOI: 10.1145/3474224. URL: https://doi.org/10.1145/3474224.

[42] Evandro Chagas Ribeiro da Rosa. *Ket Quantum Programming Git*. https://github.com/quantum-ket/ket. [Online; accessed March-2022].

[43] Rafael de Santiago Evandro Chagas Ribeiro da Rosa. *Ket Quantum Programming*. https://quantumket.org/. [Online; accessed March-2022].

[44] Riverlane. *Quantum HAL Specifications*. https://riverlane.github.io/QHAL_internal/v0.2.0/. [Online; accessed Dezember-2021].

[45] Riverlane. *QHAL - Quantum Hardware Abstraction Layer*. https://github.com/riverlane/QHAL. [Online; accessed Dezember-2021].

[46] Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Martin Vechev. "Silq: A High-Level Quantum Language with Safe Uncomputation and Intuitive Semantics". In: *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2020. London, UK: Association for Computing Machinery, 2020, 286–300. ISBN: 9781450376136. DOI: 10.1145/3385412.3386007. URL: https://doi.org/10.1145/3385412.3386007.

[47] Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Martin Vechev. *Silq*. https://github.com/eth-sri/silq. [Online; accessed Dezember-2021].

[48] Amazon. *Amazon Braket Documentation*. https://docs.aws.amazon.com/braket/?id=docs_gateway. [Online; accessed Dezember-2021]. 2020.

[49] Amazon. *Amazon Braket GitHub*. https://github.com/aws/amazon-braket-sdk-python. [Online; accessed Dezember-2021]. 2020.

[50] Nathan Killoran, Josh Izaac, Nicolás Quesada, Ville Bergholm, Matthew Amy, and Christian Weedbrook. "Strawberry fields: A software platform for photonic quantum computing". In: *Quantum* 3 (2019), p. 129.

[51] Nader Khammassi, Gian G Guerreschi, Imran Ashraf, Justin W Hogaboam, Carmen G Almudever, and Koen Bertels. "cqasm v1. 0: Towards a common quantum assembly language". In: *arXiv preprint arXiv:1805.09607* (2018).

[52] Quantum Inspire. *cQASM: A Quantum Programming Language*. https://www.quantum-inspire.com/kbase/cqasm/. [Online; accessed March-2022].

[53] D-Wave. *D-Wave Ocean Software Documentation*. https://docs.ocean.dwavesys.com/en/stable/. [Online; accessed Dezember-2021].

[54] D-Wave. *Ocean is D-Wave's suite of tools for solving hard problems with quantum computers.* https://github.com/dwavesystems/dwave-ocean-sdk. [Online; accessed Dezember-2021].

[55] Google. *Cirq - An open source framework for programming quantum computers*. https://quantumai.google/cirq. [Online; accessed March-2022].

[56] Google. *Cirq*. https://github.com/quantumlib/Cirq. [Online; accessed March-2022].

[57] Dave Bacon. *Quantum Super Entangled Language (QSEL)*. https://github.com/dabacon/qsel. [Online; accessed March-2022].

[58] A Zapata Computing Product. *Orquestra*. https://www.orquestra.io/. [Online; accessed Dezember-2021].

[59] Zapata. *Zapata Computing Git*. https://github.com/zapatacomputing. [Online; accessed Dezember-2021].

[60] Rigetti. *Welcome to Quantum Cloud Services*. https://docs.rigetti.com/qcs/. [Online; accessed Dezember-2021].

[61] Rigetti. *Projects Developed using Forest*. https://github.com/rigetti/forest-software. [Online; accessed Dezember-2021].

[62] Robert S Smith, Michael J Curtis, and William J Zeng. "A practical quantum instruction set architecture". In: *arXiv preprint arXiv:1608.03355* (2016).

[63] Microsoft. *Quantum Development Kit*. https://github.com/microsoft/Quantum. [Online; accessed November-2021]. 2017.

[64] Microsoft. *Azure Quantum documentation*. https://docs.microsoft.com/en-us/azure/quantum/?view=qsharp-preview. [Online; accessed Dezember-2021]. 2021.

[65] Krysta M Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. "Q#: Enabling scalable quantum computing and development with a high-level domain-specific language". In: *arXiv preprint arXiv:1803.00652* (2018).

[66] Microsoft. *Azure Quantum documentation*. https://docs.microsoft.com/pt-br/azure/quantum/. [Online; accessed 26-October-2021]. 2021.

[67] IBM. *Qiskit An open-source SDK for working with quantum computers at the level of pulses, circuits, and algorithms.* https://github.com/Qiskit. [Online; accessed November-2021]. 2017.

[68] Shusen Liu, Xin Wang, Li Zhou, Ji Guan, Yinan Li, Yang He, Runyao Duan, and Mingsheng Ying. *$Q|SI\rangle$: A Quantum Programming Environment*. 2017. arXiv: 1710.09500 [quant-ph].

[69] Andrew W Cross, Lev S Bishop, John A Smolin, and Jay M Gambetta. "Open quantum assembly language". In: *arXiv preprint arXiv:1707.03429* (2017).

[70] Scott Pakin. "A quantum macro assembler". In: *2016 IEEE High Performance Extreme Computing Conference (HPEC)*. 2016, pp. 1–8. DOI: 10.1109/HPEC.2016.7761637.

[71] Scott Pakin. *QMASM: A Quantum Macro Assembler*. https://github.com/lanl/qmasm. [Online; accessed Dezember-2021].

[72] Damian S Steiger, Thomas Häner, and Matthias Troyer. "ProjectQ: an open source software framework for quantum computing". In: *Quantum* 2 (2018), p. 49.

[73] Dave Wecker and Krysta M Svore. "$LIQUi|\rangle$: A software design architecture and domain-specific language for quantum computing". In: *arXiv preprint arXiv:1402.4467* (2014).

[74] Microsoft. *The Language-Integrated Quantum Operations ($LIQUi|\rangle$) simulator*. https://github.com/StationQ/Liquid. [Online; accessed March-2022].

[75] Microsoft. *$LIQUi|\rangle$ The Language Integrated Quantum Operations Simulator*. http://stationq.github.io/Liquid/. [Online; accessed March-2022].

[76] Microsoft. *Language-Integrated Quantum Operations: $LIQUi|\rangle$*. https://www.microsoft.com/en-us/research/project/language-integrated-quantum-operations-liqui/. [Online; accessed March-2022].

[77] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. "Quipper: A Scalable Quantum Programming Language". In: *arXiv e-prints*, arXiv:1304.3390 (Apr. 2013), arXiv:1304.3390. arXiv: 1304.3390 [cs.PL].

[78] Andrei Lapets, Marcus P da Silva, Mike Thome, Aaron Adler, Jacob Beal, and Martin Rötteler. "QuaFL: A typed DSL for quantum programming". In: *Proceedings of the 1st annual workshop on Functional programming concepts in domain-specific languages*. 2013, pp. 19–26.

[79] Ali Javadi Abhari, Arvin Faruque, Mohammad Javad Dousti, Lukas Svec, Oana Catu, Amlan Chakrabati, Chen-Fu Chiang, Seth Vanderwilt, John Black, Fred Chong, et al. "Scaffold: Quantum Programming Language". In: (2012).

[80] Matt Purkeypile. "Cove: A practical quantum computer programming framework". In: *arXiv preprint arXiv:0911.2423* (2009).

[81] JiaFu Xu and FangMin Song. "Quantum programming languages: A tentative study". In: *Science in China Series F: Information Sciences* 51.6 (2008), pp. 623–637.

[82] Hynek Mlnarik. "Operational semantics and type soundness of quantum programming language LanQ". In: *arXiv preprint arXiv:0708.0890* (2007).

[83] Juliana Kaizer Vizzotto and Antonio Carlos da Rocha Costa. "Towards quantum haskell via quantum arrows". In: *Workshop-Escola de Computação e Informação Quântica*. Vol. 52. Citeseer. 2006.

[84] Jiafu Xu and Fanming Song. "Quantum programming languages". In: *Frontiers of Computer Science in China* 2.2 (2008), pp. 161–166.

[85] Wolfgang Mauerer. "Semantics and simulation of communication in quantum programming". In: *arXiv e-prints*, quant-ph/0511145 (Nov. 2005), quant–ph/0511145. arXiv: quant-ph/0511145 [quant-ph].

[86] Philippe Jorrand and Marie Lalire. "From quantum physics to programming languages: a process algebraic approach". In: *International Workshop on Unconventional Programming Paradigms*. Springer. 2004, pp. 1–16.

[87] S. Bettelli, T. Calarco, and L. Serafini. "Toward an architecture for quantum programming". In: *The European Physical Journal D - Atomic, Molecular and Optical Physics* 25.2 (2003), 181–200. ISSN: 1434-6079. DOI: 10.1140/epjd/e2003-00242-2. URL: http://dx.doi.org/10.1140/epjd/e2003-00242-2.

[88] Amr Sabry. "Modeling Quantum Computing in Haskell". In: *Proceedings of the 2003 ACM SIGPLAN Haskell Workshop* (Jan. 2003). DOI: 10.1145/871895.871900.

[89] J. W. Sanders and P. Zuliani. "Quantum Programming". In: *In Mathematics of Program Construction*. Springer-Verlag, 1999, pp. 80–99.

[90] Bernhard Ömer. *QCL – A Programming Language for Quantum Computers*. http://tph.tuwien.ac.at/~oemer/qcl.html. [Online; accessed October-2021]. 2014.

[91] Bernhard Oemer. *QCL (Quantum Computing Language)*. https://github.com/aviggiano/qcl. [Online; accessed November-2021]. 2018.

[92] Philip Maymin. "The lambda-q calculus can efficiently simulate quantum computers". In: *arXiv e-prints*, quant-ph/9702057 (Feb. 1997), quant–ph/9702057. arXiv: quant-ph/9702057 [quant-ph].

[93] Peter Selinger and Benoît Valiron. "A Lambda Calculus for Quantum Computation with Classical Control". In: *Typed Lambda Calculi and Applications*. Ed. by Paweł Urzyczyn. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 354–368. ISBN: 978-3-540-32014-2.

[94] Wolfgang Mauerer. "Semantics and simulation of communication in quantum programming". In: *arXiv preprint quant-ph/0511145* (2005).

[95] SoGoSurvey. *SoGoSurvey Homepage*. https://www.sogosurvey.com. [Online; accessed 12-October-2021]. 2021.

[96] Google. *Google Forms Homepage*. https://www.google.com/forms/about/. [Online; accessed 12-October-2021]. 2021.

[97] Survio. *Survio Homepage*. https://www.survio.com. [Online; accessed 12-October-2021]. 2021.

[98] MindMiners. *MindMiners Homepage*. https://mindminers.com/. [Online; accessed 12-October-2021]. 2021.

[99] TypeForm. *TypeForm Homepage*. https://www.typeform.com/. [Online; accessed 12-October-2021]. 2021.

[100] SurveyMonkey. *SurveyMonkey Homepage*. https://https://www.surveymonkey.com/. [Online; accessed 12-October-2021]. 2021.

[101] Ugur Kuter. "Survey Methods: Questionnaires and Interviews". In: (Oct. 2021).

[102] Serene Dalati and Jorge Gómez. "Surveys and Questionnaires". In: Mar. 2018, pp. 175–186. ISBN: 978-3-319-74172-7. DOI: 10.1007/978-3-319-74173-4_10.

[103] Pramod Regmi, Elizabeth Waithaka, Anjana Paudyal, Padam Simkhada, and Edwin Van Teijlingen. “Guide to the design and application of online questionnaire surveys”. In: *Nepal Journal of Epidemiology* 6 (May 2017), p. 640. DOI: 10.3126/nje.v6i4.17258.

[104] Stack Overflow. *Stack Overflow Annual Developer Survey*. https://insights.stackoverflow.com/survey. [Online; accessed November-2021]. 2021.

[105] JetBrains. *The State of Developer Ecosystem 2021*. https://www.jetbrains.com/lp/devecosystem-2021/. [Online; accessed November-2021]. 2021.

[106] ComputerScience.org. *ComputerScience.org website*. https://www.computerscience.org/. [Online; accessed November-2021]. 2021.

[107] Increment.com. *Six questions on programming languages*. https://increment.com/programming-languages/six-questions-on-programming-languages/. [Online; accessed November-2021]. 2018.

[108] PyGitHub. *PyGitHub repository*. https://github.com/PyGithub/PyGithub. [Online; accessed June-2022].

[109] Claes Wohlin, Per Runeson, Martin Hst, Magnus C. Ohlsson, Bjrn Regnell, and Anders Wessln. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012. ISBN: 3642290434.

[110] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2022. URL: https://www.R-project.org/.

[111] Manuel De Stefano. “An Empirical Study on the Current Adoption of Quantum Programming”. In: *2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. 2022, pp. 310–312. DOI: 10.1109/ICSE-Companion55297.2022.9793820.