Available online at www.sciencedirect.com
ScienceDirect
Science of Computer Programming 00 (2024) 1–53
Science of Computer Programming

An Exploratory Study on the Usage of Quantum Programming Languages

Felipe Ferreira[a], José Campos[a,b,*]

[a] *LASIGE, Faculdade de Ciências, Universidade de Lisboa, Lisboa, Portugal*
[b] *Faculty of Engineering, University of Porto, Porto, Portugal*

Abstract

As in the classical computing realm, quantum programming languages in quantum computing allow one to instruct a quantum computer to perform certain tasks. In the last 25 years, many imperative, functional, and multi-paradigm quantum programming languages with different features and goals have been developed. However, to the best of our knowledge, no study has investigated who uses quantum languages, how practitioners learn a quantum language, how experience are practitioners with quantum languages, what is the most used quantum languages, in which context practitioners use quantum languages, what are the challenges faced by quantum practitioners while using quantum languages, are program written with quantum languages tested, and what are quantum practitioners' perspectives on the variety of quantum languages and the potential need for new languages. In this paper, we first conduct a systematic survey to find and collect all quantum languages proposed in the literature and/or by organizations. Secondly, we identify and describe 37 quantum languages. Thirdly, we survey 251 quantum practitioners to answer several research questions about their quantum language usage. Fourthly, we conclude that (i) 58.2% of all practitioners are 25–44 years old, 63.0% have a master's or doctoral degree, and 86.2% have more than five years of experience using classical languages. (ii) 60.6% of practitioners learn quantum languages from the official documentation. (iii) Only 16.3% of practitioners have more than five years of experience with quantum languages. (iv) Qiskit (Python) is the most used quantum language, followed by Cirq (Python) and QDK (Q#). (v) 42.8% use quantum languages for research. (vi) Lack of documentation and usage examples are practitioners' most challenging issues. Practitioners prefer open-source quantum languages with an easy-to-learn syntax (e.g., based on an existing classical language), available documentation and examples, and an active community. (vii) 76.4% of all participants test their quantum programs, and 42.6% test them automatically. (viii) A standard quantum language, perhaps high-level language, for quantum computation could accelerate the development of quantum programs. Finally, we present a set of suggestions for developers and researchers on the development of new quantum languages or enhancement of existing ones.

Keywords: Quantum computing, Quantum programming languages, Survey

1. Introduction

Quantum computing leverages the principles of quantum mechanics, like superposition and entanglement, to solve complex problems practically intractable in classical computing [1]. For instance, quantum computing has the potential to solve computational problems in research areas such as cryptography [2], computational physics [3], and machine learning [4] that were previously unattainable due to computational limitations [5]. Unlike classical computers that use bits to represent 0s and 1s, quantum computers use qubits (the fundamental unit of quantum

*Corresponding author
Email addresses: fc55895@alunos.fc.ul.pt (Felipe Ferreira), jcmc@fe.up.pt (José Campos)

information), which can exist in a superposition of states, allowing them to represent multiple values simultaneously and consequently perform specific calculations much faster than classical counterparts [6, 7].

To harness the full potential of quantum computing, there arises a distinct need for quantum programming languages. These languages provide a bridge between theoretical quantum concepts and practical quantum computing implementations, enabling researchers and developers to express quantum algorithms in a way that aligns with the unique principles of quantum mechanics. Some quantum languages adopt the form of dedicated programming languages, e.g., OpenQASM [8] or Q# [9], analogous to their classical counterparts. In contrast, others manifest as libraries developed on top of established classical programming languages such as Python, e.g., the Cirq library [10] developed by Google and the Qiskit library [11] developed by IBM.

In this paper, **"*quantum language* is referred to, with no distinction, both as a quantum equivalence of a programming language and as a library to write quantum programs supported by some well-known classical programming language"**, as suggested by Cervera-Lierta [12].

Over the last 25 years, at least 37 quantum languages have been proposed (see Figure 1 in Section 2). It is likely that many others are under development, given the rapid development of the quantum computing area. This number of languages calls attention further to investigate the usage of quantum languages across different dimensions. Such investigation could provide a comprehensive view of quantum languages' impact, potential, challenges, and opportunities. This knowledge can improve language design, user experience, educational resources, and the broader quantum technology ecosystem.

Hence, this paper sheds light on the usage of 37 quantum languages and puts forward several research questions (Section 3.1 describes in detail the set of RQs) which aim

- To help understand who uses quantum languages and the relationships between demographic factors, programming experience, education, and quantum physics knowledge. (Section 3.1.1)
- To provide a comprehensive perspective on participants' preferences, challenges, and motivations related to quantum languages. By examining these aspects, researchers can gain insights into the broader trends and dynamics within the quantum programming community, ultimately informing the development of better tools, resources, and strategies for learners and practitioners. (Section 3.1.2)
- To delve into the practices, preferences, and needs of testing programs written with quantum languages. By exploring these aspects, researchers can gain insights into the challenges programmers face, the tools they find valuable or lacking, and the broader strategies for enhancing the reliability and correctness of quantum programming. (Section 3.1.3)
- To explore users' perspectives on the variety of quantum languages and the potential need for new languages. Understanding these viewpoints can shed light on the perceived gaps, preferences, and motivations related to language proliferation in the quantum computing domain. (Section 3.1.4)

To answer all our research questions, we first conducted a systematic survey to find and collect all quantum languages that have been proposed, see Section 2.1. Secondly, we analyzed each language in detail and identified the main functionalities and characteristics (e.g., year of inception, type, and whether it is active). Section 2.1.4 describes the outcome of the systematic survey, Figure 1 and Table 1 list all quantum languages found, and Sections 2.2 to 2.4 briefly describe each quantum language. Thirdly, we surveyed 251 quantum developers / researchers familiar with quantum languages to assess their usage, experience, and opinion on the languages. Section 3 describes the survey in great detail, and Section 4 answers all our research questions using the data gathered from the survey. Finally, we provided a detailed set of suggestions for the further development / enhancement of quantum languages (Section 5).

2. Quantum programming languages

Quantum programming languages are specialized systems of syntax and rules designed to express instructions for quantum computers. These languages enable programmers to develop algorithms that harness the unique properties

of quantum mechanics, such as superposition and entanglement, to solve complex problems efficiently. Quantum languages provide an interface for writing code that manipulates quantum bits (qubits) and orchestrates quantum operations. They abstract the underlying complexities of quantum physics, making it easier for programmers to develop quantum software without delving into the intricacies of quantum physics. These languages are a crucial tool for realizing the potential of quantum computing and expanding its applications across various fields.

In this section, we describe the systematic survey we conducted to gather all quantum languages that have been proposed, and then we briefly present all quantum languages.

2.1. Systematic survey of quantum languages

A systematic literature review aims to identify, evaluate, and discuss all existing research pertinent to a specific domain of interest. The execution of a systematic literature review necessitates a meticulous and impartial search strategy, which guarantees the thoroughness of the search process for evaluation purposes [13].

In this context, we therefore used an evidence-based systematic methodology. We identified 37 quantum languages proposed either by researchers (e.g., on research papers) and/or individuals in open-source software repositories. The methodology provides a systematic selection and evaluation process with a rigorous and repeatable evidence-based studies selection process.

2.1.1. Data sources

We considered seven data sources for retrieving research documents, thesis, reports, and version control systems that describe or propose a quantum language.

- ACM digital library[1], IEEE Xplore[2], and Springer Link[3], which have been recommended by Kitchenham and Charters [13] and Petersen et al. [14] and used in many other systematic mapping studies (e.g., [15, 16]).
- Two recent open-access journals on quantum computing: IEEE Transactions on Quantum Engineering[4] and Quantum[5].
- arXiv[6], given that there is a trend in quantum computing to first disseminate knowledge on arXiv, even if those have not yet been peer-reviewed.
- GitHub[7], given that some quantum languages might exist in a version control system without a formal research document describing it.

2.1.2. Search

In order to conduct a thorough and efficient search for relevant artifacts, the use of appropriate search terms is essential. Kitchenham and Charters [13] have suggested population, intervention, comparison, and outcome viewpoints in this regard. Several systematic literature reviews and mapping studies have broadly used these viewpoints [17, 18, 19, 16]. In our context, the relevant terms for population and intervention are:

- **Population:** quantum
- **Intervention:** programming, language

To maintain the consistency of search on the seven data sources, given that different data sources have different interfaces for advanced search, the search was carried out by applying the following generic *search string*:

```
((quantum) AND (programming OR language))
```

The period for the search was defined from April 1980 (when Paul Benioff described the first quantum mechanical model of a computer [20]) to December 2021 (when this study was conducted) in order to try to ensure that *all* quantum languages were included.

[1] `http://www.dl.acm.org`
[2] `http://www.ieeexplore.ieee.org`
[3] `http://link.springer.com`
[4] `https://tqe.ieee.org`
[5] `https://quantum-journal.org`
[6] `https://arxiv.org`
[7] `https://github.com`

2.1.3. Procedure

In order to perform the search for quantum languages, we first defined the following inclusion criteria. An artifact is considered (and therefore included in our study) iff it matches all criteria.

- Artifacts (papers, reports, repositories, etc.) that present or propose a quantum language.
- Artifacts (papers, reports, software repositories, etc.) that are written in English.

To ensure an unbiased annotation process, the two authors independently evaluated the artifacts using two grades matching the inclusion criteria (met or not met). They discussed their evaluation afterward to reach a consensus. The decision on whether a research document was included was taken based on title, keywords, abstract, and optionally partial reading (e.g., introduction and conclusions) or full reading it to dismiss any question. The decision on whether a software repository was included was taken based on its description and any markdown file (e.g., README.md).

Then, for the elected artifacts, we collected the following metadata:

- Year of inception.
- Whether it is open-source (yes or no).
- Whether it is active (yes or no). A language is considered active iff
  - there is a comment or commit on its version control system in the last two years, in case there is a version control system.
  - or, there is any update on its webpage in the last two years, if any.
- Author's name.
- Whether it was firstly proposed by academics or industrials.
- Whether it is an imperative, functional, or multi-paradigm language.

2.1.4. Search results

Overall, we found 37 artifacts, each describing a quantum language.

Table 1 lists the quantum languages along with the metadata collected. Note that the columns without a concrete answer (i.e., '—') mean we could not collect such metadata. In detail, the oldest quantum language is λ_q, a functional language proposed by Maymin [70] in 1997. The youngsters are Ket (an imperative language proposed by Da Rosa and De Santiago [21]) and QHAL (a multi-paradigm language proposed by Riverlane [24]), both in 2021. Both are available as open-source projects, but only Ket is currently active. In summary, although the first quantum language was proposed in 1996 (28 years ago at the time of writing this paper), 60% of all quantum languages (22 out of 37) were proposed between 2011 and 2021.

- 26 languages out of 37 (70%) are available on GitHub as open-source projects, and 16 are currently active (43%).
- 24 languages were proposed by academics (65%) and 13 by industrials (35%).
- 15 languages are characterized as multi-paradigm (41%), 14 as imperative (38%), and 8 as functional (22%).

Additionally, Figure 1 reports the evolution and root of quantum languages over time. As we can see, most quantum languages were built on top of the Python language (12, 32%), followed by Haskell (7, 19%), and C++ (5, 14%). Two popular examples in Python are Cirq [10] proposed by Google and Qiskit [11] proposed by IBM. According to the TIOBE Index for 2021, Python is the most used classical language. This could explain why most quantum languages are based on Python.

In the following subsections, we group quantum languages by their type (imperative, functional, and multi-paradigm) as suggested by Sofge [71], and briefly describe each language.

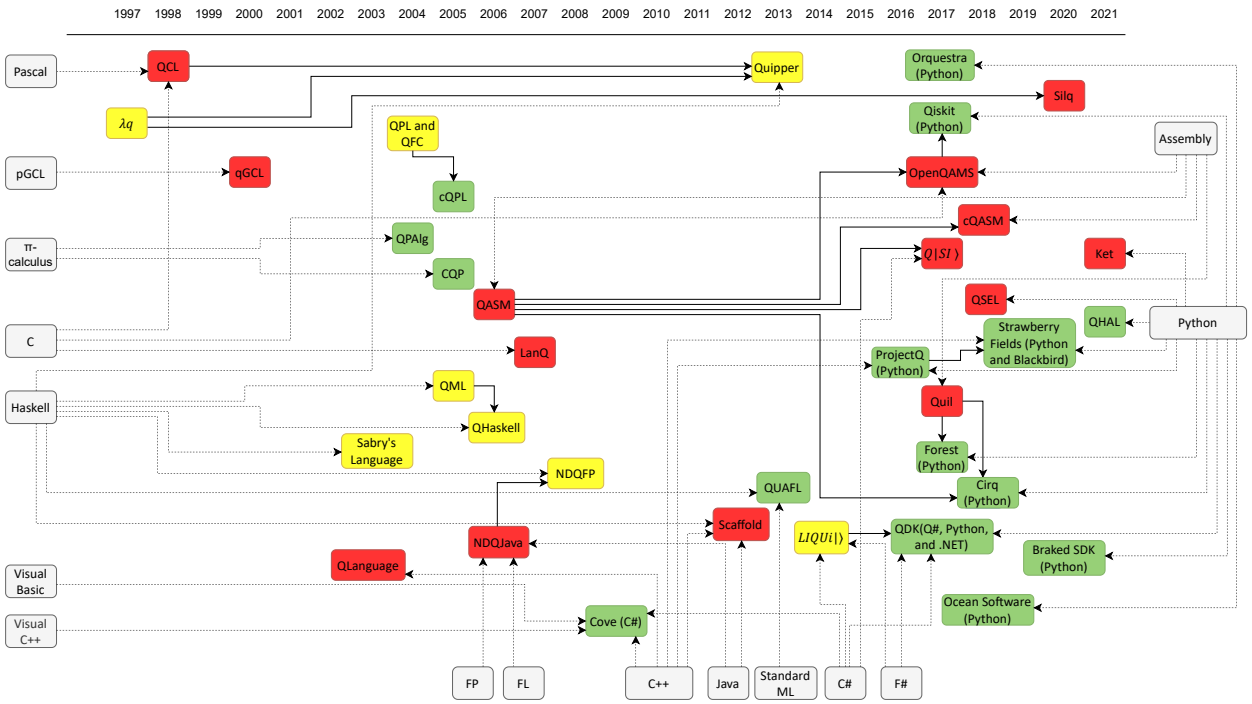

Figure 1: Evolution of quantum programming languages.
Background colors represent the type of a programming language, i.e., classical languages are colored with gray, functional languages with yellow, imperative languages with red, and multi-paradigm languages with green.
Dot-based-arrows represent quantum languages that evolve to new quantum languages and full-arrows represent quantum languages originated from classical languages. Note that some quantum languages have been built on top of classical languages, e.g., Qiskit (Python), and others are syntactically related, e.g., Q# is syntactically related to both C# and F# yet also has some significant differences.

2.2. *Imperative quantum programming languages*

Imperative programming languages are a category of programming languages that provide explicit, step-by-step instructions for a computer to follow. In imperative programming, the emphasis is on describing how a program should accomplish a task, often by specifying sequences of statements that modify the program's state. This programming style is rooted in giving the computer a set of commands to execute, similar to a recipe or a set of directions. Common examples of classical imperative languages include C/C++ and Java. In the following subsubsections, we briefly describe imperative quantum languages.

2.2.1. *QCL*

QCL (Quantum Computation Language) [67] was the first quantum 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 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.

2.2.2. *QASM*

QASM (Quantum Macro Assembler) [44] is a low-level quantum 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.

Name	Year	Open source	Active	Academic or Industrial	Author	Type
Ket [21, 22, 23]	2021	Yes	Yes	Academic	Rosa et al.	Imperative
QHAL [24]	2021	Yes	No	Industrial	Riverlane	Multi-paradigm
Silq [25, 26]	2020	Yes	Yes	Academic	Bichsel et al.	Imperative
Braket SDK (Python) [27, 28]	2020	Yes	Yes	Industrial	Amazon	Multi-paradigm
Strawberry Fields (Python and Blackbird) [29]	2019	Yes	Yes	Industrial	Killoran et al. (Xanadu)	Multi-paradigm
cQASM [30, 31]	2018	Yes	Yes	Academic	Khammassi et al.	Imperative
Ocean Software (Python) [32, 33]	2018	Yes	Yes	Industrial	D-Wave	Multi-paradigm
Cirq (Python) [10, 34]	2018	Yes	Yes	Industrial	Google AI Quantum Team	Multi-paradigm
QSEL [35]	2018	Yes	No	Academic	Bacon	Imperative
Orquestra (Python) [36]	2017	Yes	Yes	Industrial	Zapata	Multi-paradigm
Forest (Python) [37]	2017	Yes	Yes	Industrial	Rigetti	Multi-paradigm
Quil [38]	2017	Yes	Yes	Industrial	Smith et al.	Imperative
QDK (Q#, Python and .NET Languages) [39, 9, 40, 41]	2017	Yes	Yes	Industrial	Microsoft	Multi-paradigm
Qiskit (Python) [11, 42]	2017	Yes	Yes	Industrial	IBM	Multi-paradigm
$Q\|SI\rangle$ [43]	2017	-	-	Academic	Duan et al.	Imperative
OpenQASM [8]	2017	Yes	Yes	Industrial	Bishop et al.	Imperative
QASM [44, 45]	2016	Yes	Yes	Academic	Pakin	Imperative
ProjectQ (Python) [46]	2016	Yes	Yes	Academic	Hañer et al.	Multi-paradigm
$LIQUi\|\rangle$ [47, 48, 49, 50]	2014	Yes	No	Industrial	Wecker et al.	Functional
Quipper [51]	2013	Yes	No	Academic	Green et al.	Functional
QUAFL [52]	2013	-	-	Academic	Lapets et al.	Multi-paradigm
Scaffold [53]	2012	Yes	Yes	Industrial	Abhari et al.	Imperative
Cove (C#) [54]	2009	Yes	No	Academic	Purkeypile	Multi-paradigm
NDQFP [55]	2008	No	No	Academic	Xu et al.	Functional
LanQ [56]	2007	Yes	No	Academic	Mlnarík	Imperative
QHaskell [57]	2006	-	-	Academic	Vizzotto et al.	Functional
NDQJava [58]	2006	No	No	Academic	Xu et al.	Imperative
cQPL [59]	2005	Yes	No	Academic	Mauerer	Multi-paradigm
QML [60]	2005	Yes	No	Academic	Altenkirch et al.	Functional
CQP [61]	2005	-	-	Academic	Gay et al.	Multi-paradigm
QPAlg [62]	2004	-	-	Academic	Jorrand et al.	Multi-paradigm
QPL and QFC [63]	2004	-	-	Academic	Selinger	Functional
Q Language [64]	2003	Yes	No	Academic	Bettelli et al.	Imperative
Sabry's Language [65]	2003	-	-	Academic	Sabry	Functional
qGCL [66]	2000	-	-	Academic	Sanders et al.	Imperative
QCL [67, 68, 69]	1998	Yes	No	Academic	Ömer et al.	Imperative
λ_q [70]	1996	-	-	Academic	Maymim et al.	Functional

Table 1: Quantum programming languages, ordered descending by year, that have been proposed by others.

2.2.3. Silq

Silq [25] is a high-level quantum 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.

2.2.4. Q

Bettelli presents an extension of the C++ language as a model for a high-level quantum language [72]. 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.

2.2.5. qGCL

The language for quantum programs specification, qGCL [73] (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.

2.2.6. LanQ

Mlnarik [56] introduced in 2007 a high-level quantum 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.

2.2.7. $Q|SI\rangle$

$Q|SI\rangle$ [43] 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.

2.2.8. OpenQASM

OpenQASM [8] (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.

2.2.9. Scaffold

Scaffold [53] 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.

2.2.10. cQAMS

The cQASM [30, 31] (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.

2.2.11. Quil

The Quil [38] 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 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.

2.2.12. QSEL

QSEL [35] (Quantum Super Entangled Language) is a quantum 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.

2.2.13. Ket

Ket [22, 23, 21] is a high-level classical-quantum 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.

2.2.14. NDQJava

NDQJava [58] is a quantum 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.

2.3. Functional quantum programming languages

Functional programming languages are a category of programming languages that focus on describing what needs to be accomplished rather than specifying how to achieve it. In contrast to imperative programming, which emphasizes giving step-by-step instructions, functional programming allows developers to state the desired outcome and let the underlying system figure out the best way to achieve it. Examples of classical functional languages include Haskell and Prolog.

2.3.1. QPL and QFC

QFC and QPL are two functional quantum languages introduced by Selinger [63] 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.

2.3.2. QML

Altenkirch [60] developed a functional Haskell-like quantum 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.

2.3.3. Sabry's

Sabry [65] 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.

2.3.4. 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 [74] made the first effective effort to create a functional quantum 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 [75], 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.

2.3.5. Quipper

Quipper [51] is a functional quantum 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.

2.3.6. NDQFP

NDQFP [55] is a modular functional quantum 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.

2.3.7. $LIQUi|\rangle$

$LIQUi|\rangle$ [47, 48, 49, 50] is a quantum 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.

2.3.8. QHaskell

QHaskell [57] is a functional quantum 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.

2.4. 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 paradigms in the same language, in which they can freely mix the paradigms, making it possible to develop programs more effectively and efficiently. The following subsubsections briefly describe quantum languages that are multi-paradigm.

2.4.1. QDK (Q#, Python, and .NET)

QDK (Quantum Development Kit) [9], 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 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# [40, 41] is a multi-paradigm quantum 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.

2.4.2. cQPL

Mauerer's [76] 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.

2.4.3. QPAlg

QPAlg (Quantum Process Algebra) was created by Jorrand and Lalire [62] to describe the interactions between quantum and classical processes using an algebraic process approach. 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.

2.4.4. CQP

CQP [61] (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.

2.4.5. QualFL

QualFL, created by Lapets et al., is a type of domain-specific quantum 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.

2.4.6. QHAL

QHAL [24] 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.

2.4.7. QISKIT (Python)

Qiskit [11], 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.

2.4.8. Cirq (Python)

Cirq [10, 34] 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.

2.4.9. Braket SDK (Python)

Amazon Braket [27, 28] 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.

2.4.10. Strawberry Fields (Blackbird and Python)

Strawberry field [29] 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 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 [29] 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).

2.4.11. Forest (Python)

Forest [37] 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.

2.4.12. DWave Ocean (Python)

Ocean software development kit (SDK) [32, 33] 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.

2.4.13. Orquestra (Python)

Orquestra [36] 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.

2.4.14. Cove (C#)

Cove [54] 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.

2.4.15. ProjectQ (Python)

ProjectQ [46] 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.

3. Study

In this section, we described our set of research questions and the survey conducted to collect data that would allow us to answer each research question.

3.1. Research questions

We formulated 15 research questions and organized them in four groups. The following subsection describe in detail each group of RQs using the Goal Question Metric approach proposed by Basili et al. [77].

3.1.1. Group I – Who uses quantum languages

Firstly, our goal is to help understand who uses quantum languages and the relationships between demographic factors, programming experience, education, and quantum physics knowledge. By

- Knowing **who uses a quantum language** could help researchers, developers, and organizations understand users' demographics, backgrounds, expertise, and the diversity and profiles of those users within the quantum programming community. Such knowledge can guide the development of learning resources, tutorials, and documentation tailored to the specific needs of different user groups.
- One can map the evolving quantum computing ecosystem by understanding **where quantum languages are used geographically and in various industries**. This insight can be valuable for collaborations, partnerships, and investment decisions related to quantum technology.

- Educators and platform providers can design more effective learning pathways by understanding **how people learn and adopt quantum languages**. This could include creating structured curricula, interactive coding environments, and hands-on projects that cater to different learning styles and levels of expertise.

RQ1 What is the profile of individuals who use quantum languages?

RQ2 How do the learning pathways for classical and quantum languages overlap or diverge?

RQ3 How do individuals' experiences differ between classical languages and quantum languages?

3.1.2. Group II – Participants' preferences, challenges, and motivations

Secondly, our goal is to provide a comprehensive perspective on participants' preferences, challenges, and motivations related to quantum languages. By examining these aspects, researchers can gain insights into the broader trends and dynamics within the quantum programming community, ultimately informing the development of better tools, resources, and strategies for learners and practitioners. In summary, by

- Understanding **how quantum languages are used** in different contexts allows developers to refine and expand language features. For example, if a particular application domain is prevalent, the language could be enhanced with libraries or functionalities that cater to that domain's needs.
- Understanding **where quantum languages are used** can inform policy discussions and ethical considerations. For example, privacy concerns and regulations must be addressed if quantum computing applications are used in sensitive areas like cryptography.
- Analyzing the **challenges users face**, such as barriers to entry, complexity, or lack of resources, can guide the development of strategies to lower these barriers. This might involve creating more user-friendly interfaces, better documentation, or educational initiatives.

RQ4 Which classical and quantum languages are used by participants, and how long have they been used?

RQ5 What quantum language do participants primarily use and what specific features or attributes do they appreciate or find challenging in it?

RQ6 What relation exists between participants' primary quantum language, their major, their familiarity with quantum physics, and their personal/professional experiences?

RQ7 In what contexts do participants apply quantum languages?

RQ8 What quantum languages are participants interested in trying or using in the future, and why?

RQ9 What are the participants' perspectives on the importance of learning a quantum language?

RQ10 What are the main challenges participants face when selecting a quantum language?

3.1.3. Group III – Practices, preferences, and needs of testing programs written with quantum languages

Thirdly, our goal is to delve into the practices, preferences, and needs of testing programs written with quantum languages. By exploring these aspects, researchers can gain insights into the challenges programmers face, the tools they find valuable or lacking, and the broader strategies for enhancing the reliability and correctness of quantum programming. In summary, by

- Understanding the **needs and gaps in existing tools for writing quantum programs** can help researchers and developers create more efficient, user-friendly, and practical programming environments. This could lead to improved tools that make quantum programming more accessible, i.e., reduce the barriers for programmers who want to enter the quantum computing field.
- Identifying the **tools commonly used for verification and validation** can guide improvements to these tools. Developers can gain insights into the strengths and weaknesses of existing solutions and work on enhancing or creating new tools to address user needs.

RQ11 What are the perceived needs and gaps in tools for writing quantum programs?

RQ12 Are quantum programs tested, how often, and how?

RQ13 What tools do users employ for testing quantum programs?

3.1.4. Group IV – Participants' perspectives on the variety of quantum languages

Fourthly, our goal is to explore users' perspectives on the variety of quantum languages and the potential need for new languages. Understanding these viewpoints can shed light on the perceived gaps, preferences, and motivations related to language proliferation in the quantum computing domain.

RQ14 How do users perceive the diversity of quantum languages?

RQ15 What factors influence participants' opinions about the necessity of introducing new quantum languages?

3.1.5. Metrics

In all RQs but RQ9, RQ10, and RQ11, we measured the number and percentage of participants that selected each of the options available in each survey's question. For example, in RQ1 we measured the number/percentage of participants that are under 18 years old, 18-24 years old, 25-34 years old, . . . , 65 years or older; the number/percentage of participants per country, and the number/percentage of participants per education level. In RQ12 we measured the number/percentage of participants that either test or not their quantum programs, and how. Whenever possible and relevant, we combined measurements from multiple RQs to show relations and insight details. In RQ9, RQ10, and RQ11 (which analyze open questions) we did not measure any variable and therefore only report participants' opinions.

3.2. Survey structure

Following the guidelines proposed by Kuter and Yilmaz [78], Dalati and Gómez [79], Regmi et al. [80] on how to elaborate questions for questionnaires and how to conduct a survey with human developers, we formulated a total of 35 questions organized in six sections:

- **Section 1** briefly describes the questionnaire, its scope, the confidentiality agreement, the estimated duration, and assesses if the participant has ever used any quantum language.
- **Section 2** asks demographic questions, e.g., age, location.
- **Section 3** asks questions regarding participants's education and experience.
- **Section 4** asks questions regarding the usage of quantum languages.
- **Section 5** asks questions regarding the usage of tools to write programs with quantum languages.
- **Section 6** asks questions regarding participants' perspectives on the variety of quantum languages and the potential need for new languages.

Table A.2 describes, by section, the questions that were asked in the survey, the reason for each question, and the type/domain of each answer. Additionally, it also provides a mapping between the questions asked in the survey and the set of research questions presented in Section 1.

3.3. Survey platform

Although there are several survey platforms one could use to conduct an online survey (e.g., SoGoSurvey [81], Google Forms [82], Survio [83], MindMiners [84], Typeform [85], and SurveyMonkey [86]), only Google Forms [82] is free, has no limitation on the number of questions / answers, or the number of collaborators to create/edit the survey. Thus, we elected Google Forms for the task.

3.4. Survey participants

The survey was mainly published on social networks and mailing lists about *quantum computing*. Table B.3 lists the online platforms where we successfully and unsuccessfully[8] published the survey. In summary, we published it on five channels on Facebook, five on LinkedIn, four on Slack, two on Reddit, one on Discord and Twitter, and one

[8] Not all social networks allow members to post new messages, for example.

mailing list. The survey was also emailed to authors of papers on quantum languages and other researchers in quantum computing[9]. The total number of users registered on all platforms is 174,058.

Note that one might be registered on a platform / social network about *quantum computing* but have never used a *quantum language*. Thus, the first question of our survey explicitly asks "Have you ever used any quantum language?" to roll out participants unfamiliar with quantum languages.

3.5. How was the survey conducted

This survey was conducted in two phases. The **first phase** was done with only two participants (not authors of this paper). This phase served as a pre-test for the survey during which the participants validated the survey's questions. This aimed to ensure that the questions were clear and ordered accordingly to the topics. Each participant rated each question from 1 (not clear) to 5 (must have). All participants rated all questions with 5. Additionally, this phase helped in calibrating the estimated time required for participants to answer the survey, ensuring that it is reasonable and appropriate. Then, in the **second phase**, the survey was published on social networks and conducted by 251 participants. The survey was open for 60 days (from March 31, 2022 to May 31, 2022), and one reminder was sent after ten days. After 60 days, the survey was closed, and the data was collected.

3.6. Data analysis

The data analysis of this survey was made after we collected all data from the question answers. Given our study is on quantum languages and their usage, we filtered out data / answers from the participants that have never used quantum languages. Then, we computed and reported raw numbers and percentage values for closed answers, and manually analyzed the open answers and share some of them in the results section.

Furthermore, to statistically compare the relation between categorical variables (e.g., quantum languages and participants' major) we used Fisher's exact test [87] with a 95% confidence level.

4. Results

In this section, we reported the findings of our study based on the information we gathered from the survey. We had a total of 251 responses, of which 208 (82.9%) answered that they have used quantum languages and, therefore, continued answering the questions of the survey. The remaining 43 (17.1%) answered that they have never used quantum languages and, therefore, did not answer any other question.

4.1. RQ1: What is the profile of individuals who use quantum languages?

Age: 39.9% of all participants are 25–34 years old, while 26.4% are 18–24 years old, and 18.3% are 35–44 years old. Regarding age outliers, 1.9% are 65 or older, and 2.9% are under 18 (see Figure 2).

Country: Most participants live in the United States of America (22.1%), followed by Spain, India, and Canada (5.8%), see Figure 3.

Gender: 88.9% of all participants identify as male, 6.2% as woman, 2.9% prefer not to say, and 1.9% are non-binary.

Education level: 27.9% of all participants have a doctoral degree, 35.1% have a master's degree, and 21.6% have a bachelor's degree (see Figure 4). Figure 5 reports the relation between participants' age and education level, for instance, more than 80.0% of the participants that are 25–64 years old and have a master's or doctorate degree.

Major: As expected, most majors are related to computer science (47.1%) or software engineering (13.8%), and physics (37.6%), see Figure 6.

Quantum physics knowledge: Given that to successfully develop a quantum algorithm / program, one might require some knowledge of quantum physics, we asked the participants to rate their knowledge of quantum physics from 0 (no knowledge) to 5 (expert). We found that all participants have some knowledge of quantum physics, with the experience level being balanced between 1 and 5. For instance, novices in quantum physics represent 19.2% of all participants, and experts represent 18.8%.

[9] In an attempt to find more participants, we also mined the email address of users that have contributed to GitHub projects related to quantum computing — repositories' description that match the keywords "quantum algorithms", or "quantum programs", or "quantum programming languages".

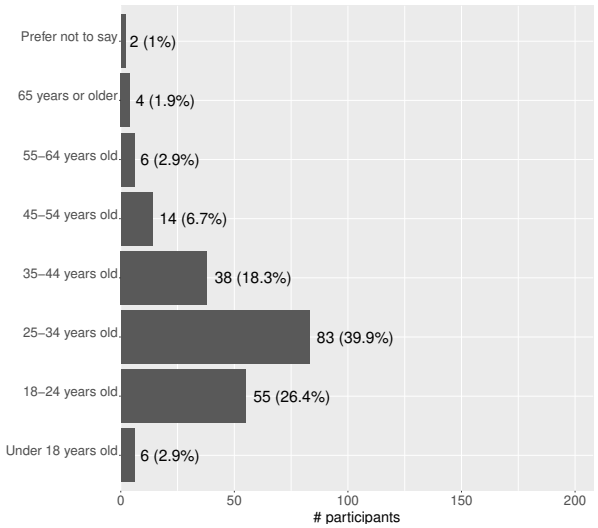

Figure 2: Distribution of the 208 participants' age grouped by category.
Figure discussed in RQ1 (Section 4.1).

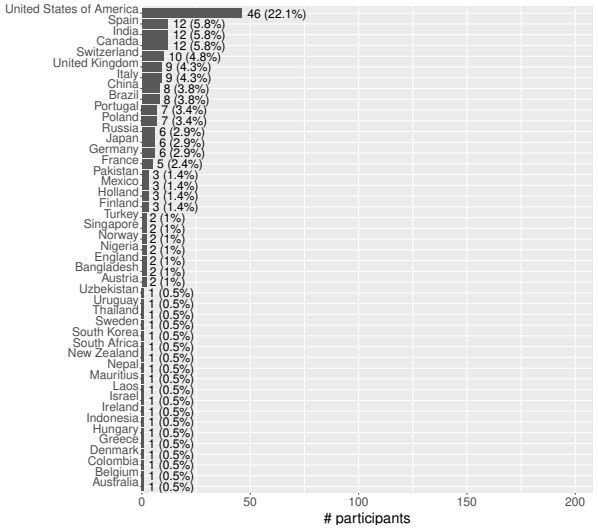

Figure 3: Distribution of the 208 participants' countries.
Figure discussed in RQ1 (Section 4.1).

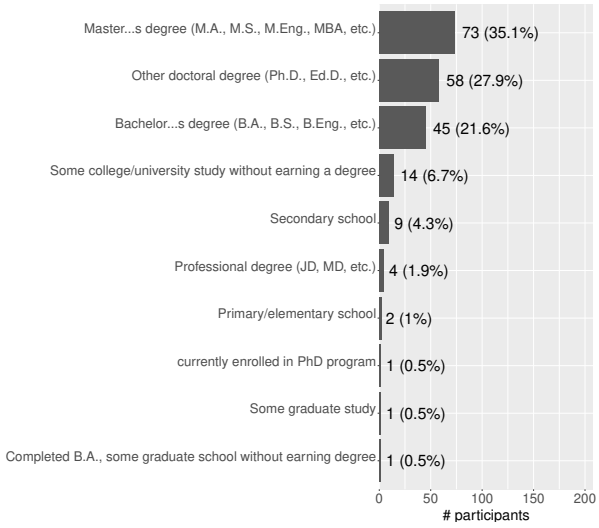

Figure 4: Formal education of the 208 participants'.
Figure discussed in RQ1 (Section 4.1).

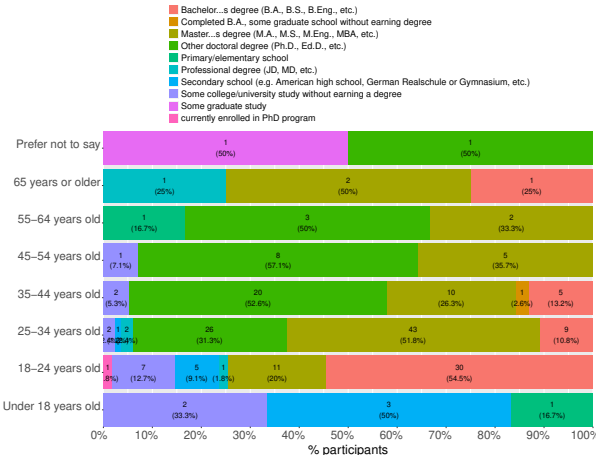

Figure 5: Age and education level of the 208 participants. According to Fisher's exact test (p-value 0.00000), we reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (age and education level). In other words, knowing the value of one variable helps to predict the value of the other variable.
This figure reports the intersection between Figures 2 and 4's data, and it is discussed in RQ1 (Section 4.1).

Learning quantum physics: Most participants learned quantum physics through books (69.6%) and / or at the university 64.7% (see Figure 8).

Current job: Most of the participants are developers / programmers / software engineers (37.5%), students (35.1%), or scientists/researchers (33.2%), see Figure 7. As participants were allowed to choose more than one option, we observed that most participants have more than one job (or more than one role in their organization), for example, academic researchers are also students or professors.

4.2. RQ2: How do the learning pathways for classical and quantum languages overlap or diverge?

The learning pathways for classical languages and quantum languages diverge and are quite different. On one hand, most participants have learned classical languages from books (66.8%), using an online forum (61.5%), and at

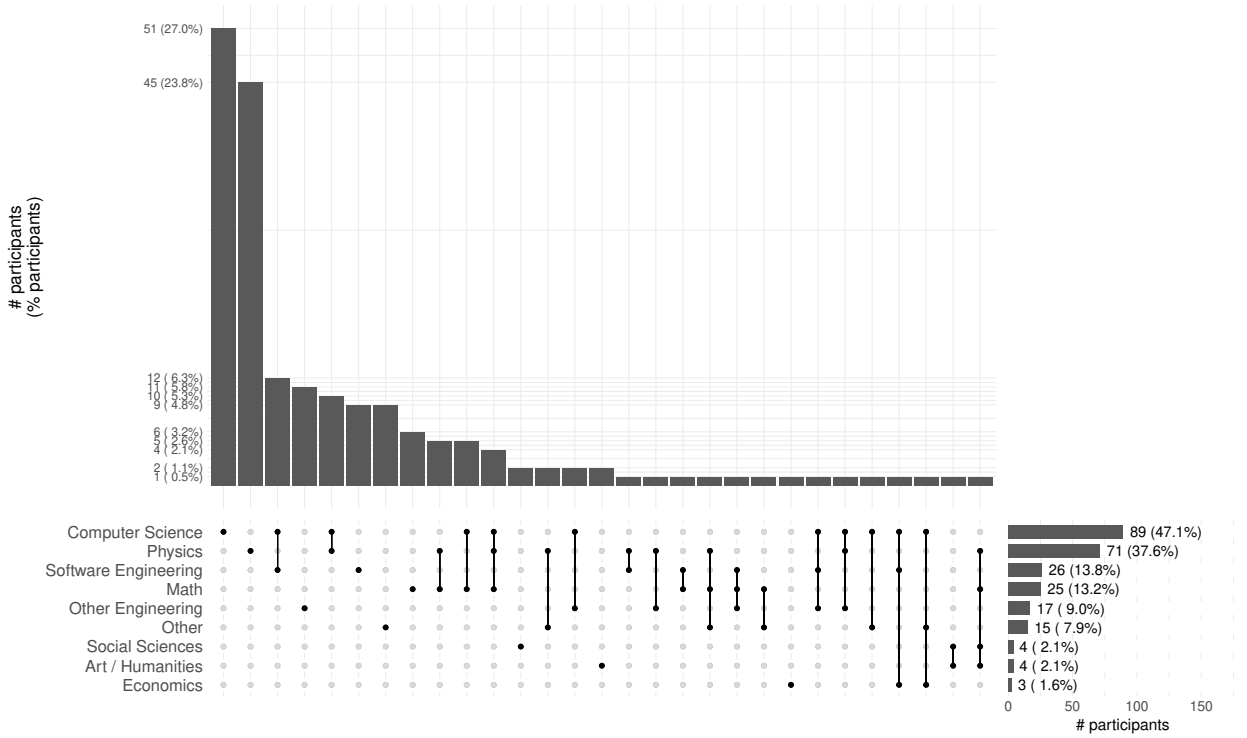

Figure 6: The major subject(s) of 189 participants. (Note that survey question number 12 (see Table A.2) was not a mandatory question and 19 participants did not answer it.)
Figure discussed in RQ1 (Section 4.1).
This figure shows the intersections of sets with an UpSet plot [88] using the UpSetR package [89]. Although the most common approach to visualizing sets is to use Venn Diagrams, it only scales up to four or five sets. UpSet plots, in contrast, show the intersections of sets as a matrix and are, therefore, well suited for quantitative data analysis with more than three sets. In an UpSet plot, each row of the matrix (bottom part) corresponds to a set, and the bar charts on the right show the size of the set. For instance, 89 participants did their major in 'Computer Science', followed by 'Physics' (71), and 'Software Engineering' (26). Each column of the matrix corresponds to a possible intersection (the filled-in and connected cells / dots show which set is part of an intersection), and the bar charts on top show the cardinality of the intersections. Recall that each participant's major might be described by just one label or multiple labels. For instance, 51 participants did their major **just** in 'Computer Science', and 12 participants did their major in 'Computer Science' **and** 'Software Engineering' (see the two filled-in and connected cells / dots in the third column).

school (60.6%), see Figure 9. On the other hand, 60.6% have learned quantum languages from official documentation (e.g., website, markdown files), 41.3% through online courses, and 38.9% from books (see Figure 10). It is worth noting that the importance of language documentation concerning learning is evident; the more documentation a programming language has, the more likely participants will use it.

4.3. RQ3: How do individuals' experiences differ between classical languages and quantum languages?

On one hand, 86.2% of all participants have more than five years of personal experience with classical languages, and 50.0% over ten years (see Figure 11). Similarly, 83.7% have over five years of professional experience with classical languages, and 28.8% over ten years (see Figure 12). On the other hand, 83.7% have less than five years of personal experience with quantum languages (see Figure 13), 38.0% have never used quantum languages professionally, and 50.5% have less than five years (see Figure 14). Only 16.3% of participants have more than five years of experience with quantum languages.

Thus, regarding personal/professional experience, the participants have much less experience in quantum languages than in classical ones. This is not, however, surprising. 60% of all quantum languages listed in Figure 1 were only proposed in the last 10 years (i.e., between 2011 and 2021). The most used quantum languages (discussed in **RQ4**) were only proposed four years ago, between 2017 and 2018.

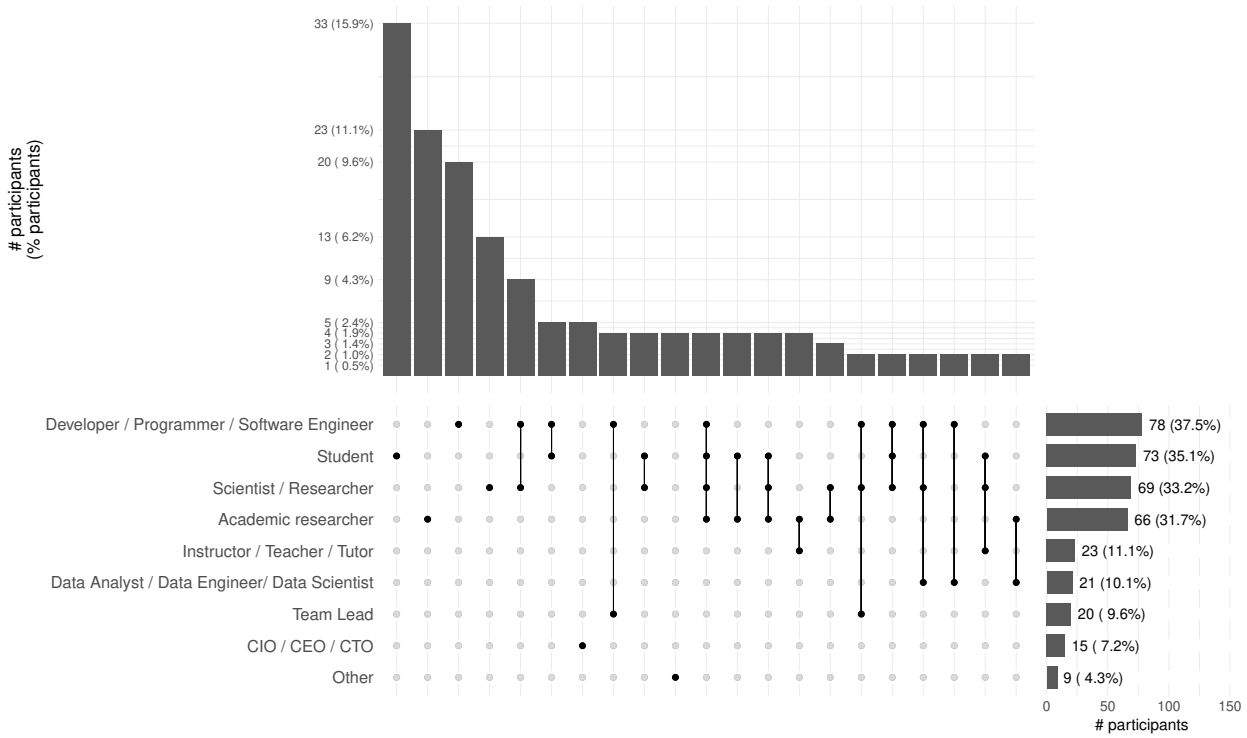

Figure 7: Current job of the 208 participants.
Figure discussed in RQ1 (Section 4.1).
(Please refer to Figure 6 for an explanation of the UpSet plot [88].)

4.4. RQ4: Which classical and quantum languages are used by participants, and how long have they been used?

Regarding classical languages, most participants use the Python programming language (92.3%). It is worth noting that this is the classical language on which most quantum languages have been built on top of, as shown in Figure 1. Other classical languages, e.g., C++ (60.1%) and C (55.3%), are also widely used. The least used classical languages are Groovy (1.4%), COBOL (1.9%), and Delphi/Object Pascal (2.9%), see Figure 15.

Regarding quantum languages, the vast majority of participants, 177 (85.1%), use Qiskit (Python), followed by Cirq (Python) (91, 43.8%), and OpenQASM (77, 37.0%). Some participants mentioned some other quantum languages that were not included in our study (see Section 4.16.1 for more details): Xanadu's Pennylane [90] was mentioned by 4 participants, QUTIP [91, 92] by 2, and FunQy [93], SQIR [94], Quingo [95], and Perceval [96] once. Regarding how long, most participants have been mainly using Qiskit (Python) for up to 2 years and several other languages for less than a year, e.g., QDK (Q#), and Cirq (Python), see Figure 16.

4.5. RQ5: What quantum language do participants primarily use and what specific features or attributes do they appreciate or find challenging in it?

Primary quantum language: most participants use Qiskit (Python) (64.9%) as their primary quantum language, followed by Cirq (Python) (5.3%) and QDK (Q#) (4.3%), see Figure 17. Several reasons could explain why Qiskit (Python) is the most used primary quantum language. We conjecture two reasons. (i) Qiskit (Python) is built on top of the most used classical language, Python. This would allow one with basic knowledge of Python to quickly start developing a quantum program and use an extensive and comprehensive set of other frameworks and libraries from the Python ecosystem in their quantum programs. (ii) The large number of tutorials, course materials, and resources for learning Qiskit (Python) that are available online could also explain its popularity.

Ease rating: Several reasons could lead a person to choose which programming language he or she wants to use, e.g., easy-to-learn syntax, online documentation, examples, and support through online forums. We asked participants

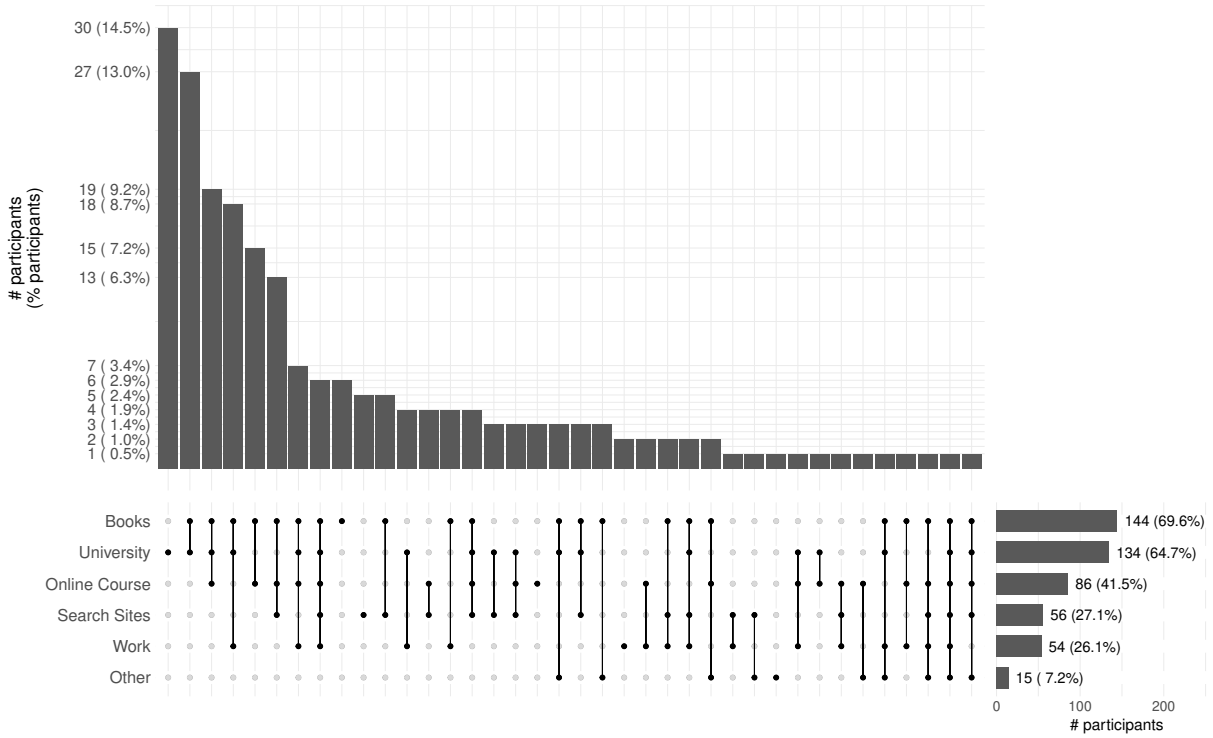

Figure 8: Education of 207 participants regarding learning quantum physics. (Note that survey question number 10 (see Table A.2) was not a mandatory question and one participant did not answer it.)
Figure discussed in RQ1 (Section 4.1).
(Please refer to Figure 6 for an explanation of the UpSet plot [88].)

to rate their primary language using a scale of 1 (difficult) to 5 (easy) in terms of features, available documentation, code examples, easy-to-code, and support. Figure 18 reports the results for the most used language, Qiskit (Python). Overall, Qiskit (Python) was rated as a ≥ 3 by 88.0% of the participants and as ≥ 4 by 56.2%. Most of the participants that use Qiskit (Python) as their primary quantum language rated its forums and support as a 3, 35.6% and 30.4%, respectively. And they rated features and easy-to-code, documentation, and code examples as a 4 out of 5 (35.6%, 34.8%, 36.3% respectively). Despite the positive ratings, there is still room for improvement–5.2% rated Qiskit (Python)'s ease as 1.

Likes: Most participants who elected Qiskit (Python) as their primary quantum language reported that they like it because it is open-source, easy to understand, easy to code, Python-based, has many tutorials, continuous updates, operators available, and has a large and active community. Some participants also reported, as positive aspects, that IBM is leading and supporting its development, and the number of modules the language provides. One of the participants mentioned: "There are many modules to help us to run real quantum hardware".

Dislikes: Regarding what participants do not like or miss in Qiskit (Python), we can highlight the performance and consumption of RAM, over-complicated architecture, still a low-level language, and a few examples to run on current quantum hardware.

Forums/Communities: An extensive and active community might lead one to choose a particular language. Most participants use StackOverflow (76.2%), followed by Slack (47.1%), QOSF (22.7%), Other (17.4%), and Devtalk (1.7%) to interact with the quantum community. As we can see in Figure 19, we can find users of Qiskit (Python) on all forums (but more likely on StackOverflow), and StackOverflow is the only forum used by users of some languages, i.e., QML, QHaskell, QASM, Ket, and DWave Ocean (Python).

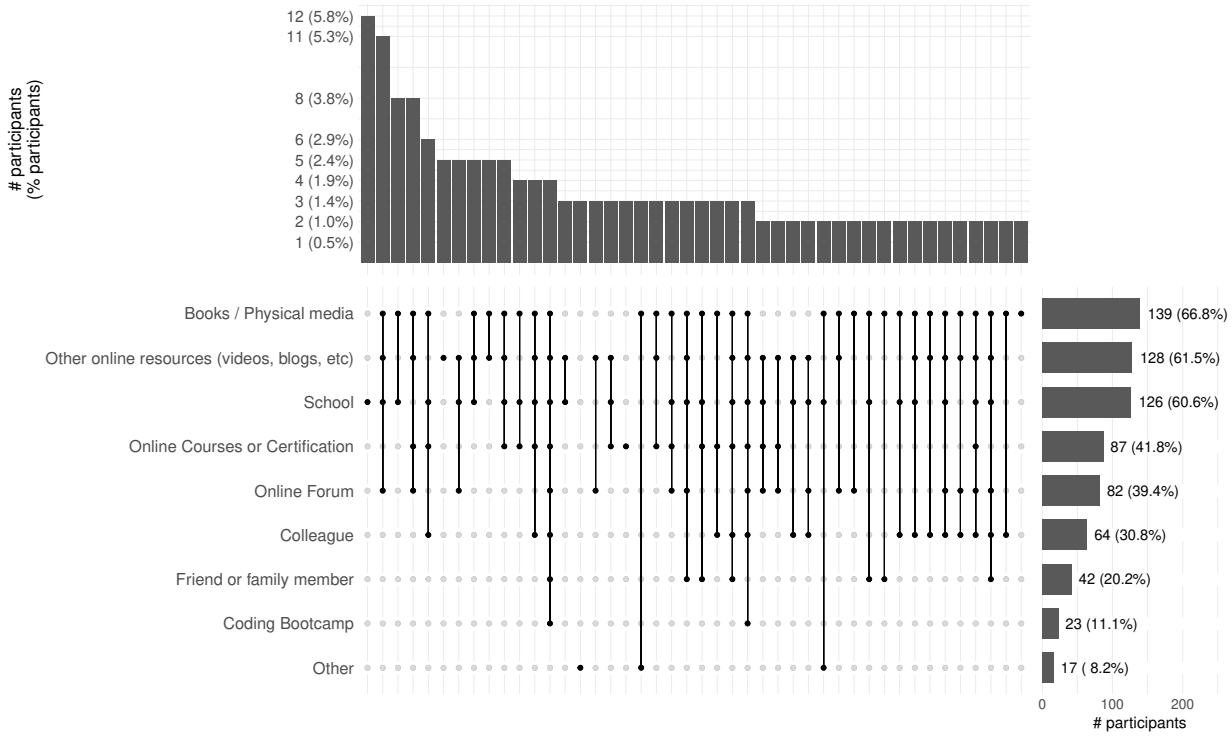

Figure 9: How the 208 participants learned classical languages.
Figure discussed in RQ2 (Section 4.2).
(Please refer to Figure 6 for an explanation of the UpSet plot [88].)

4.6. RQ6: What relation exists between participants' primary quantum language, their major, their familiarity with quantum physics, and their personal/professional experiences?

Figure 20 shows from which sources participants learned their primary quantum language. Overall, participants use a variety of sources to learn more about a specific quantum language. All participants reported documentation for the Orquestra language, which might indicate that the language provides good documentation or that there is no other source from which others could learn. QHaskell has been learned mainly at the University.

Figure 21 shows the relation between participants' primary quantum language and their major. Participants with specific majors use specific quantum languages, but not exclusively. For instance,

- Strawberry Fields (Python) is exclusively used by participants with a major in physics.
- QHaskell is exclusively used by participants with a major in software engineering.
- QASM and Ket are exclusively used by participants with a major in computer science.
- All languages but Strawberry Fields (Python), QHaskell, Orquestra (Python), and Braket SDK (Python) are used by participants who have a major in computer science.
- Participants with a major in physics use Strawberry Fields (Python), Silq, Quil, Qiskit (Python), QDK (Q#), $Q|SI\rangle$, Orquestra (Python), and Cirq (Python).

Results reported in Figure 22 suggest that some quantum languages might be suitable for novices in quantum physics (e.g., QML, QDK (Q#), QASM, DWave Ocean (Python), and Braket SDK (Python)), while other languages (e.g., Strawberry Fields (Python), Quipper, Quil, and Cirq (Python)) are mainly used by experts in quantum physics. Qiskit (Python) is used by participants with different knowledge levels in quantum physics, which may indicate that

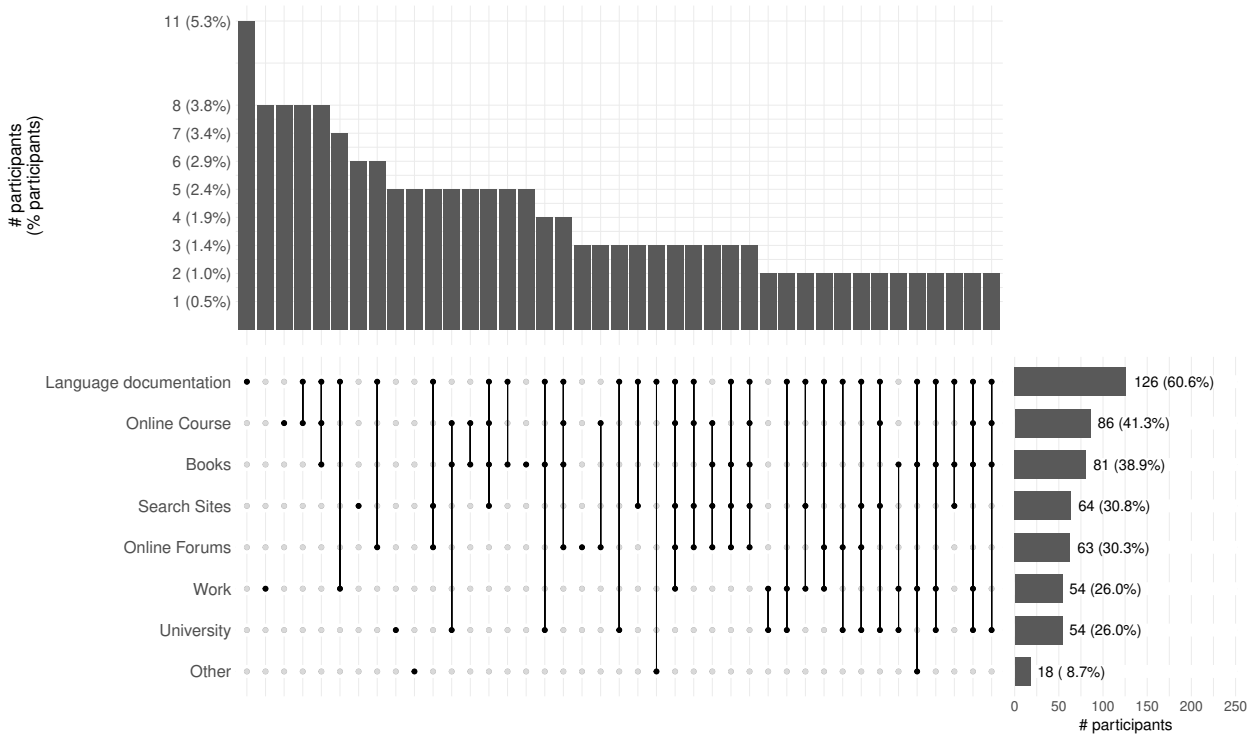

Figure 10: How the 208 participants learned quantum languages.
Figure discussed in RQ2 (Section 4.2).
(Please refer to Figure 6 for an explanation of the UpSet plot [88].)

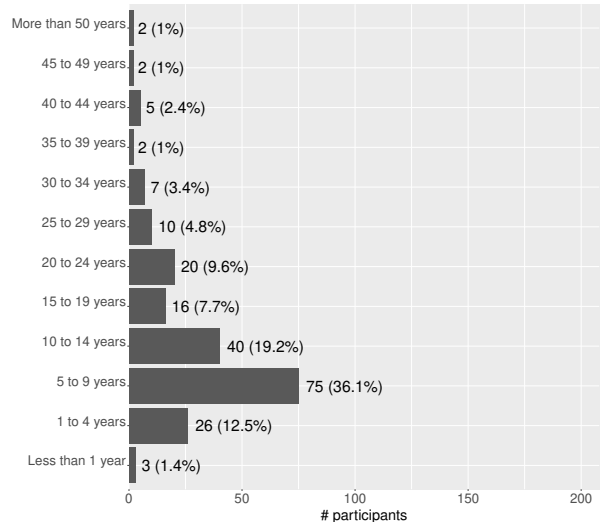

Figure 11: Personal experience of the 208 participants with classical languages.
Figure discussed in RQ3 (Section 4.3).

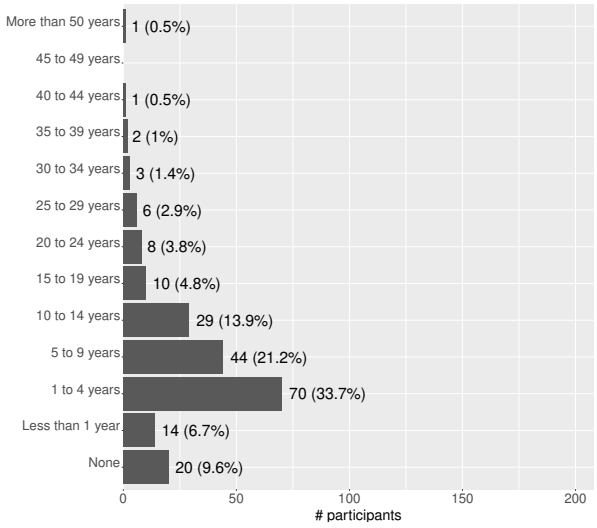

Figure 12: Professional experience of the 208 participants with classical languages.
Figure discussed in RQ3 (Section 4.3).

its features and functionalities are suitable for participants with different levels of knowledge. This could explain its popularity.

Figures 23 and 24 show participants' personal and professional experience with quantum languages. As discussed

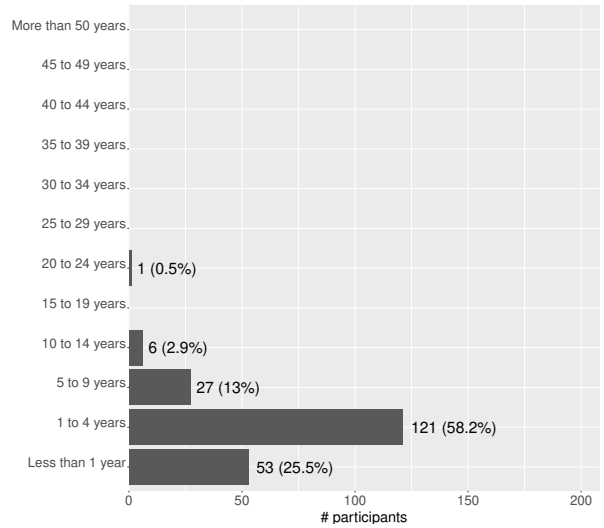

Figure 13: Personal experience of the 208 participants with quantum languages.
Figure discussed in RQ3 (Section 4.3).

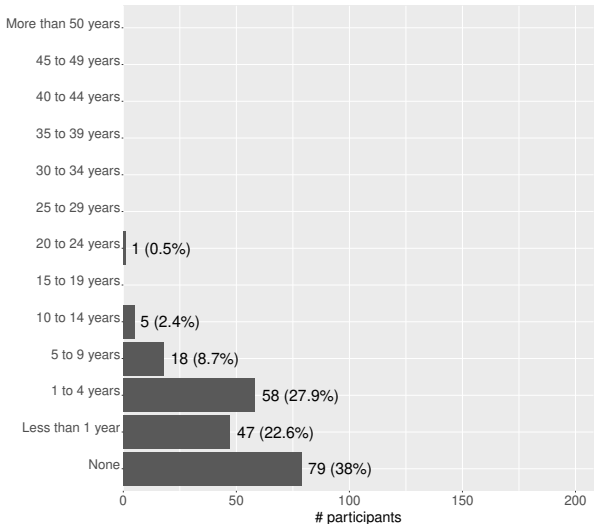

Figure 14: Professional experience of the 208 participants with quantum languages.
Figure discussed in RQ3 (Section 4.3).

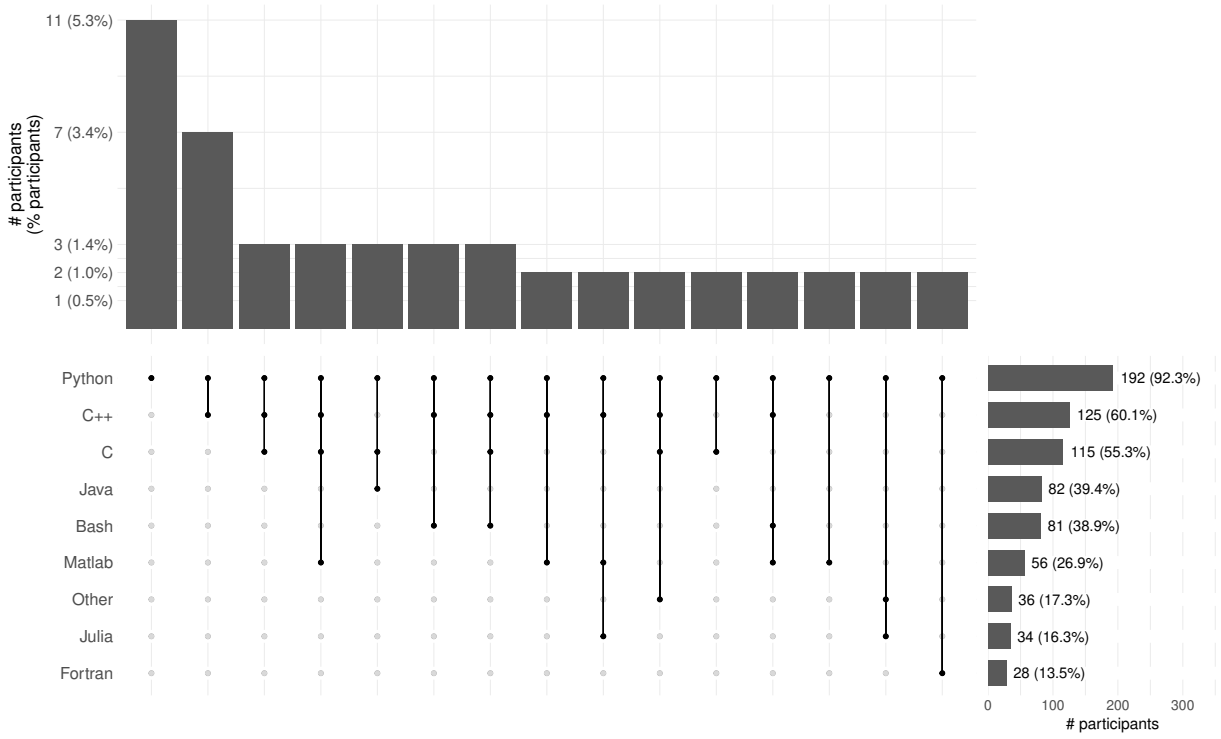

Figure 15: Classical languages used by the 208 participants.
Figure discussed in RQ4 (Section 4.4).
(Please refer to Figure 6 for an explanation of the UpSet plot [88].)

in **RQ4**, most participants have less than four years of personal experience or no professional experience, likely because most languages were created recently.

- Strawberry Fields (Python), Silq, QHaskell, $Q|SI\rangle$, Orquestra (Python), and Ket are exclusively used by partic-

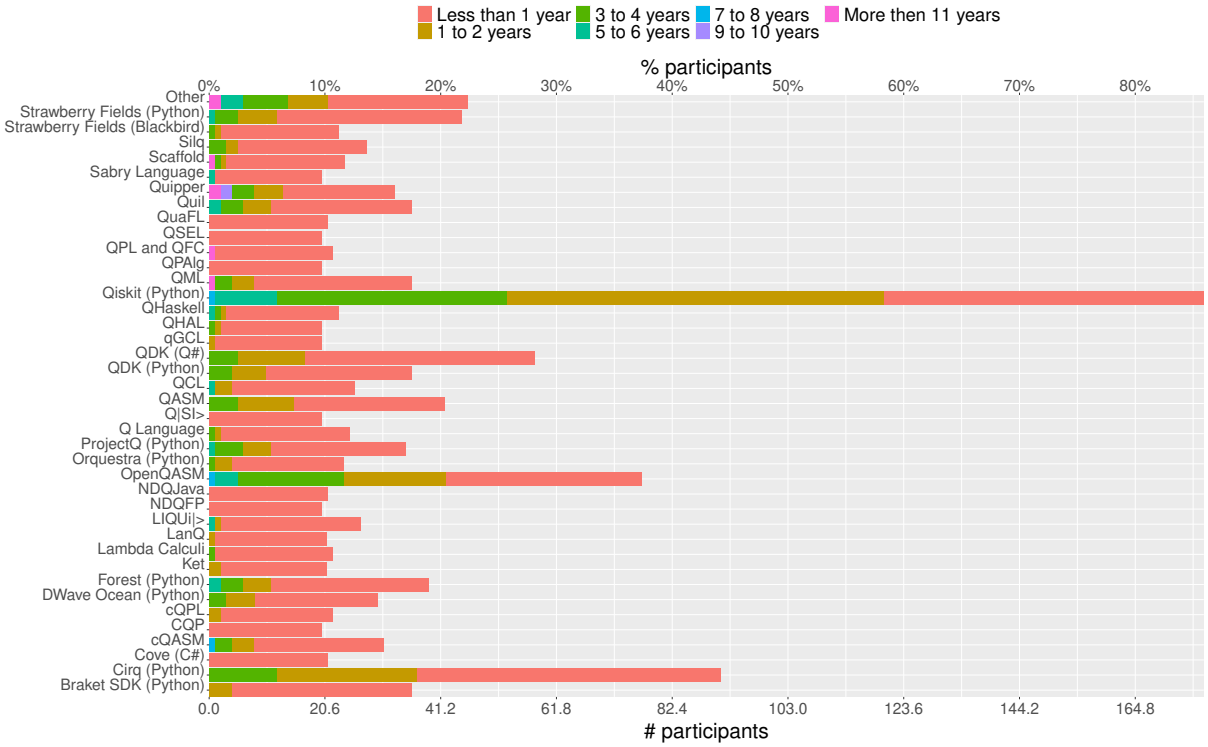

Figure 16: Quantum languages used by the 208 participants and for how long. (See Table C.4 for absolute numbers.)
Figure discussed in RQ4 (Section 4.4).

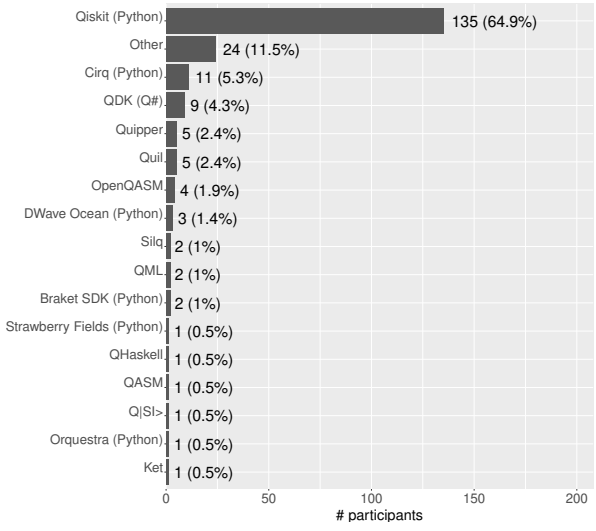

Figure 17: The primary quantum language used by the 208 participants.
Figure discussed in RQ5 (Section 4.5).

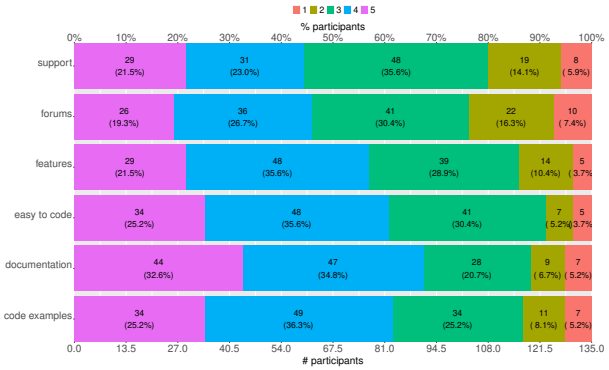

Figure 18: The rate in terms of ease (e.g., features, easy to code, documentation, code examples, support, forums) of Qiskit, the primary quantum language of 135 out of the 208 participants (see Figure 17).
Figure discussed in RQ5 (Section 4.5).

ipants with 1–4 years of *personal* experience with quantum languages.

- Quil, Qiskit (Python), DWave Ocean (Python), and Cirq (Python) are mainly used (however, not exclusively) by participants with 1–4 years of *personal* experience.

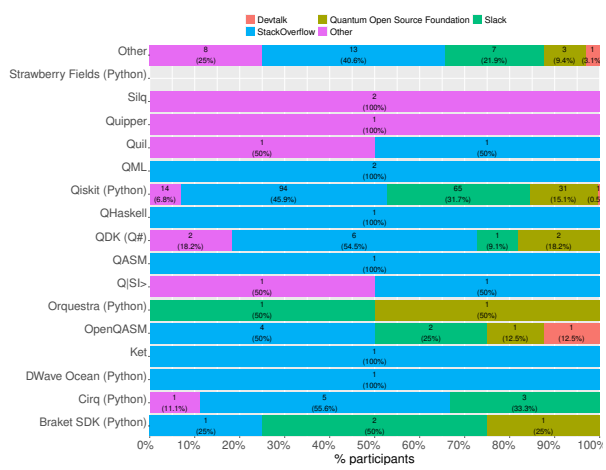

Figure 19: Participants' primary quantum language and the forums used by them. According to Fisher's exact test (p-value 0.05942), we do not reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (primary quantum language and forums used).
Figure discussed in RQ5 (Section 4.5).

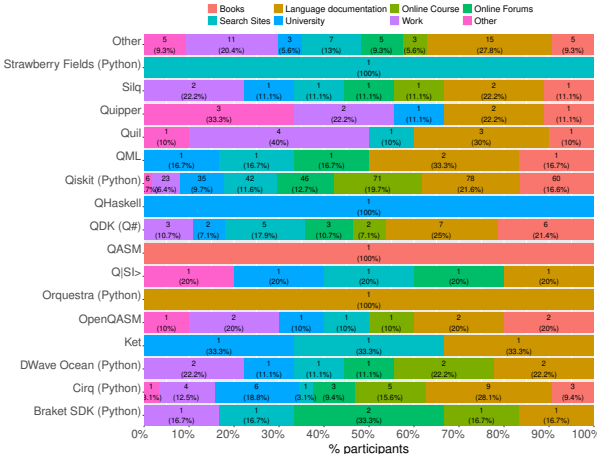

Figure 20: Participants' primary quantum language and how they learned it. According to Fisher's exact test (p-value 0.01074), we reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (primary quantum language and how participants learned it).
This figure reports the intersection between Figures 10 and 17's data, and it is discussed in RQ6 (Section 4.6).

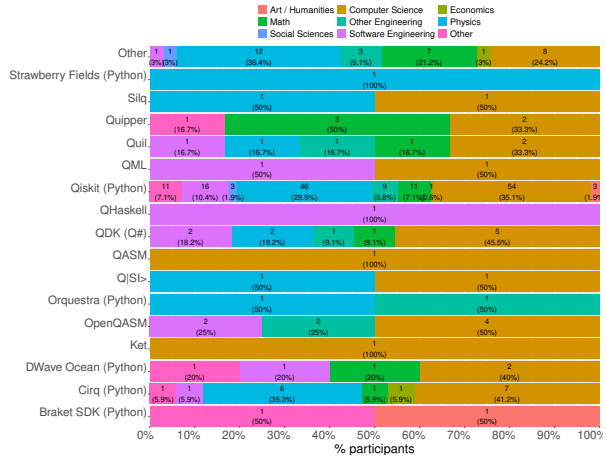

Figure 21: Participants' primary quantum language and their major. According to Fisher's exact test (p-value 0.27723), we do not reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (primary quantum language and participants' major).
This figure reports the intersection between Figures 6 and 17's data, and it is discussed in RQ6 (Section 4.6).

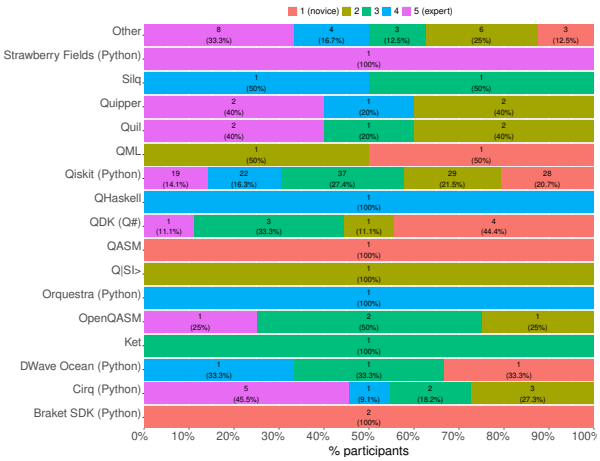

Figure 22: Participants' primary quantum language and their knowledge in quantum physics. According to Fisher's exact test (p-value 0.12501), we do not reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (primary quantum language and participant's knowledge in quantum physics).
Figure discussed in RQ6 (Section 4.6).

- Strawberry Fields (Python), Silq, Orquestra (Python), Ket are exclusively used by participants with 1–4 years of *professional* experience with quantum languages.
- QHaskell is exclusively used by participants with less than a year of *professional* experience.
- QASM and $Q|SI\rangle$ are not used by participants professionally.

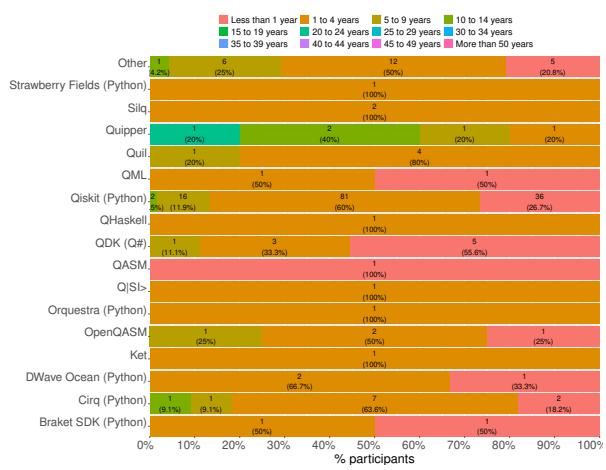

Figure 23: Participants' primary quantum language and their personal experience with quantum languages. According to Fisher's exact test (p-value 0.20924), we do not reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (primary quantum language and participants' personal experience).
This figure reports the intersection between Figures 13 and 17's data, and it is discussed in RQ6 (Section 4.6).

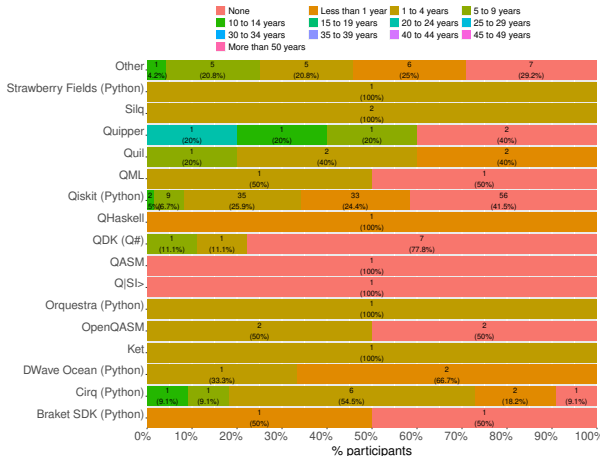

Figure 24: Participants' primary quantum language and their professional experience with quantum languages. According to Fisher's exact test (p-value 0.20924), we do not reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (primary quantum language and participant's professional experience with quantum languages).
This figure reports the intersection between Figures 14 and 17's data, and it is discussed in RQ6 (Section 4.6).

4.7. RQ7: In what contexts do participants apply quantum languages?

42.8% of all participants use quantum languages for research, while 34.6% use them because they like to learn new languages, 16.3% use them for work, and 6.2% for other purposes. The fact that most participants use quantum languages for research or to acquire new knowledge indicates that quantum computing is still a relatively new field, mainly conducted at labs and research institutes. This is in line with previous works. For instance, De Stefano [97] conducted an empirical study on the current adoption of quantum programming in open-source repositories, and he found out that the primary use of quantum languages (such as Qiskit, Cirq, and Q#) is in research and to assist teaching. The relation between the quantum languages used by the participants and what for they use them is shown in Figure 25. Some languages, such as Ket, QASM, Quipper, Silq, and Strawberry Fields, are mainly used for research, while Orquestra and Quil are mainly used for (industrial/engineering) work.

Participants with different profiles use quantum languages for different reasons. Figure 27 shows the relation between participants' major and for what they use quantum languages. Most participants who have a major in economics use quantum languages to augment their knowledge, while those who have a major in physics, math, or computer science use them for research. Interestingly, 50% of the participants who have a major in art/humanities use quantum languages for other reasons than to learn, research, or work.

Figure 28 shows the relation between participants' current job and why they use quantum languages. The participants working as Architect, Product Managers, Technical Support and Technical Writer use quantum languages for work, while those working as Academic Research, Instructor/Teacher/Tutors, Scientist/Researchers, and Students mainly use them for research. Tester/QA Engineer, UX/UI Designer, and DBA (Database Administrator) use quantum languages to learn new languages. Participants working as Marketing Managers use quantum languages for other reasons than learning, researching, or working.

4.8. RQ8: What quantum languages are participants interested in trying or using in the future, and why?

Participants are willing to try other quantum languages in the future: Cirq (25.5% of all participants), Qiskit (20.2%), and Q# (18.8%), see Figure 29. These languages are supported by the largest technology companies (Google, IBM, and Microsoft, respectively), and their development might be aligned, to some extent, with the development of real quantum computers. As most users would like to work in cutting-edge technology such as quantum computing

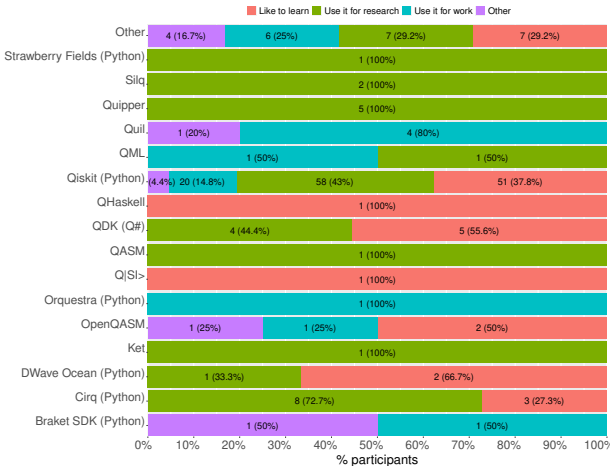

Figure 25: Participants' primary quantum language and for what they use it. According to Fisher's exact test (p-value 0.00185), we reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (primary quantum language and their usage).
Figure discussed in RQ7 (Section 4.7).

and using real quantum devices instead of simulators, the language that better aligns with these expectations and with an easy syntax has significant changes to impose itself over the others.

Most of the participants would like to try a new quantum language out of curiosity, i.e., just because they "heard about the language" (63.9%), or because it is widely used (24.0%), or read an article about it (23.6%), see Figure 30. Figure 26 shows the relation between quantum languages and why the participants would like to try them. Note that 34.3% of the participants who have not yet used Qiskit (Python) know it is one of the most widely used.

4.9. RQ9: What are the participants' perspectives on the importance of learning a quantum language?

The fact that quantum computing is an innovative area with much potential is a significant factor for the participants to work with a quantum language. In summary, we observed that the most common opinions on the importance of learning a quantum language reported by participants are:

- "It is an important emerging field."
- "Development of quantum algorithms."
- "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."

4.10. RQ10: What are the main challenges participants face when selecting a quantum language?

The main challenges participants encounter are the need for more documentation, usage examples, and the lack of communities to ask questions about the languages. One participant mentioned "Lack of documentation and examples" while another mentioned "The lack of documentation and videos, tutorials, etc. showing the basics of these languages.". Others reported that quantum languages are so different from each other that it is very difficult to compare them. One participant mentioned "too many choices, and they are all different." while another, "There are so many frameworks. It is difficult to try and find the best one.".

Participants also reported that most languages lack tools for abstractions and high-level reusability. In addition to the need for deep knowledge about qubit, measurement, and quantum physics in general, this was reported as a challenge to successfully develop a quantum algorithm/program.

In summary, the main challenges encountered by the participants are:

- "Lack of online documentation, videos, tutorials, and examples."

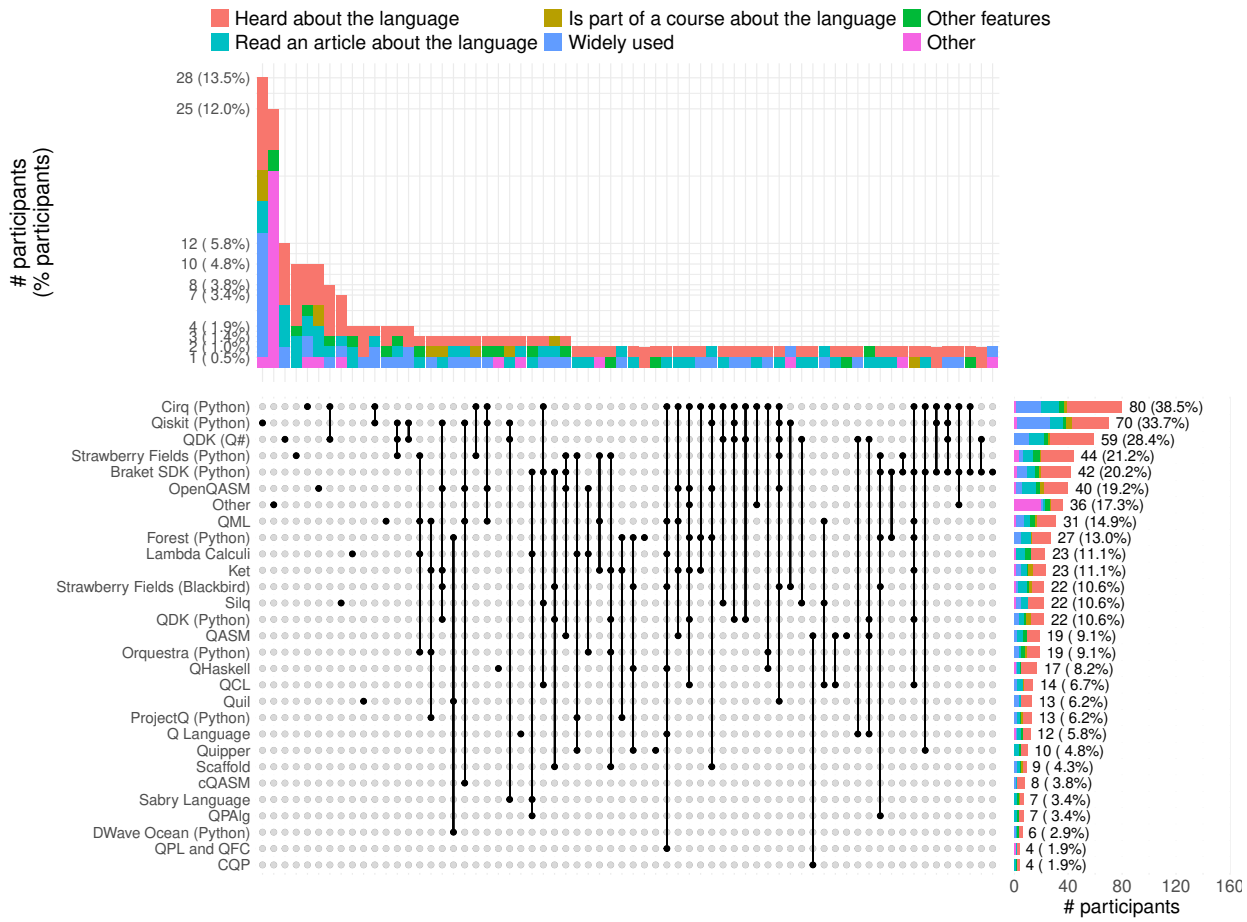

Figure 26: Quantum languages that the 208 participants would like to try and why. According to Fisher's exact test (p-value 0.00039), we reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (quantum languages and reasons why participants would like to try them).
This figure reports the intersection between Figures 29 and 30's data, and it is discussed in RQ8 (Section 4.8).
(Please refer to Figure 6 for an explanation of the UpSet plot [88].)

- "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."

4.11. RQ11: What are the perceived needs and gaps in tools for writing quantum programs?

We asked the participants what tools they thought were necessary or missing to use quantum languages better and develop better quantum programs. Most participants reported that a tailored quantum integrated development environment and tools for testing and debugging quantum programs still need to be included. As mentioned in the

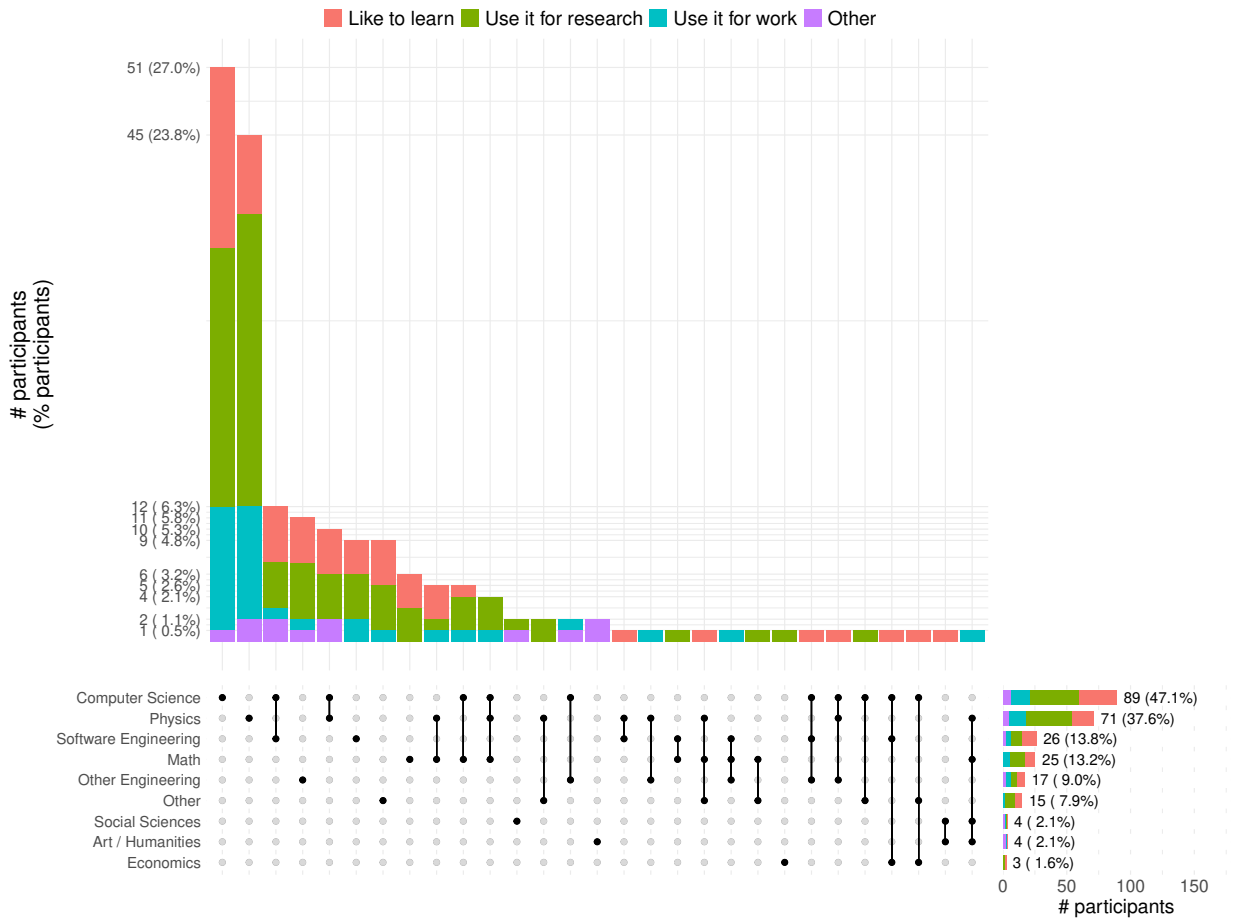

Figure 27: Participants' major and the reason for what they use quantum languages. According to Fisher's exact test (p-value 0.40102), we do not reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (participants' major and the reason for what they use quantum languages).
Figure discussed in RQ7 (Section 4.7).
(Please refer to Figure 6 for an explanation of the UpSet plot [88].)

previous RQ, some participants indicated that a high-level language is necessary to facilitate the effective development of quantum programs. The participants also raised some other needs, such as:

- "Quantum integrated development environments, with debugging tools tailored for quantum and with tools to visualize circuit."
- "Tools for error correction."
- "Better device level optimizers, simulators, and device access."
- "Meaningful standardised benchmark suites."
- "Easy interoperability between quantum libraries and/or frameworks."

4.12. RQ12: Are quantum programs tested, how often, and how?

76.4% of all participants test their quantum programs, while 23.6% do not. Figure 31 reports that participants who write quantum programs using, e.g., Braked SDK, QHaskell, Orquestra, *always* test their programs, while participants who use, e.g., QASM and Ket, *never* perform any kind of testing procedure.

Of the participants who test their quantum algorithms / programs, the majority (64.7%) perform testing every time the source code is modified, 16.2% only before releasing the program to production, 6.6% every day, and 12.6% answered other.

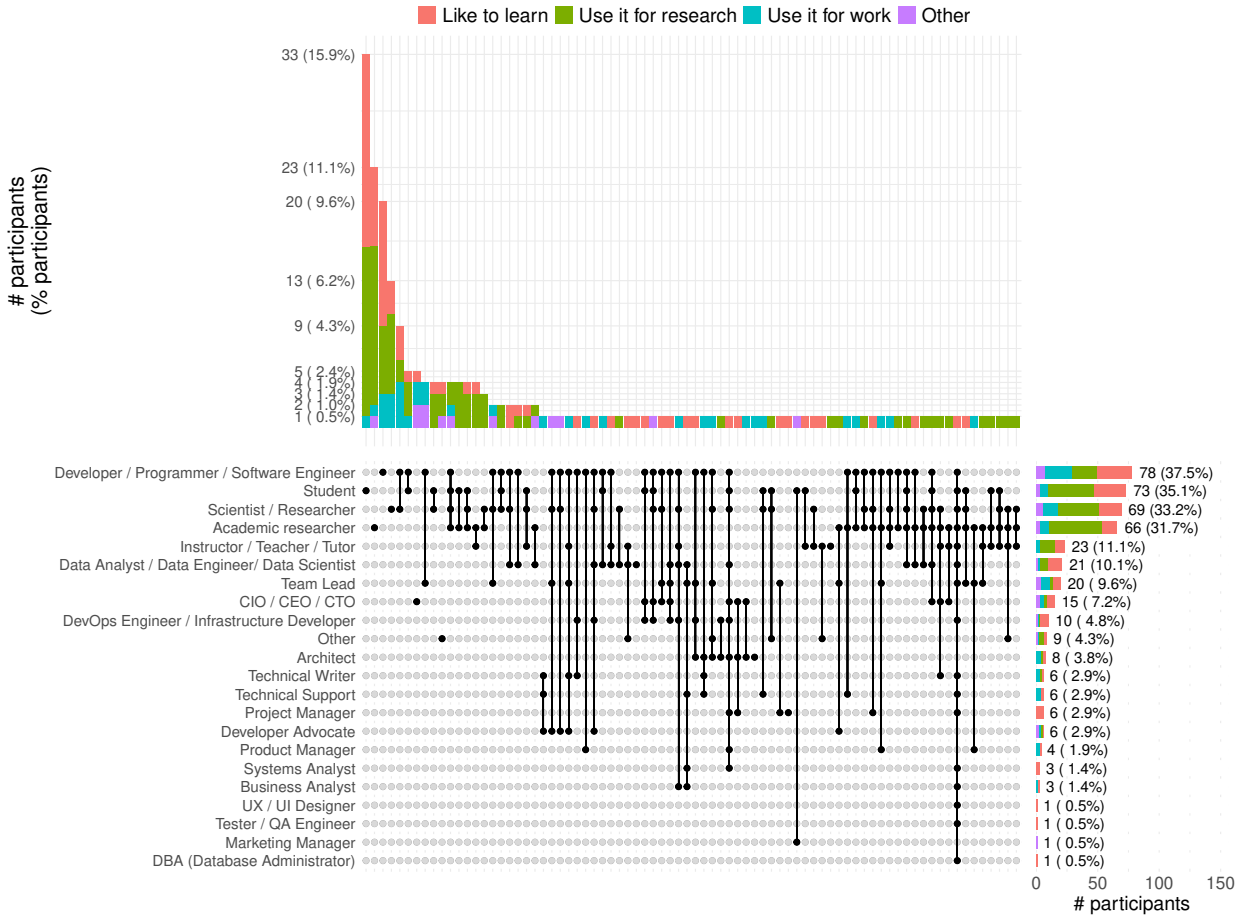

Figure 28: Participants' current job and the reason for what they use quantum languages. According to Fisher's exact test (p-value 0.00000), we reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (participants' current job and reasons for what they use quantum languages).
Figure discussed in RQ7 (Section 4.7).
(Please refer to Figure 6 for an explanation of the UpSet plot [88].)

57.4% of all participants test their programs manually, while 42.6% test them automatically (e.g., through the execution of unit tests), see Figure 32. This might indicate a need for testing tools for quantum programs or a lack of dissemination of existing testing tools.

Programs written with Silq, QML, $Q|SI\rangle$, and Orquestra (Python), are tested automatically, while programs written with Strawberry Fields (Python), QHaskell, QASM, Ket, and DWave Ocean (Python) are tested manually. Programs written with the most used language, i.e., Qiskit (Python), are mainly tested automatically.

4.13. RQ13: What tools do users employ for testing quantum programs?

The tool most used by participants to test their programs is Qiskit - QASM Simulator, used by 77.1% of participants, followed by Cirq Simulator and Testing - cirq.testing (18.5%), see Figure 33. Given that Qiskit (Python) is the most used language used by participants, the testing tool provided by language is also the most used one. Worth noting that participants are starting to use novel tools to automatically generate tests for quantum programs (e.g., Quito [98]), to fuzz (e.g., QuanFuzz [99]), and mutation testing (e.g., Muskit [100]).

Figure 34 shows the relation between the quantum language and the tool used to ease testing. Although programs written with Qiskit (Python) are mainly manually tested (Figure 32), 74.8% of the participants use the Qiskit - QASM

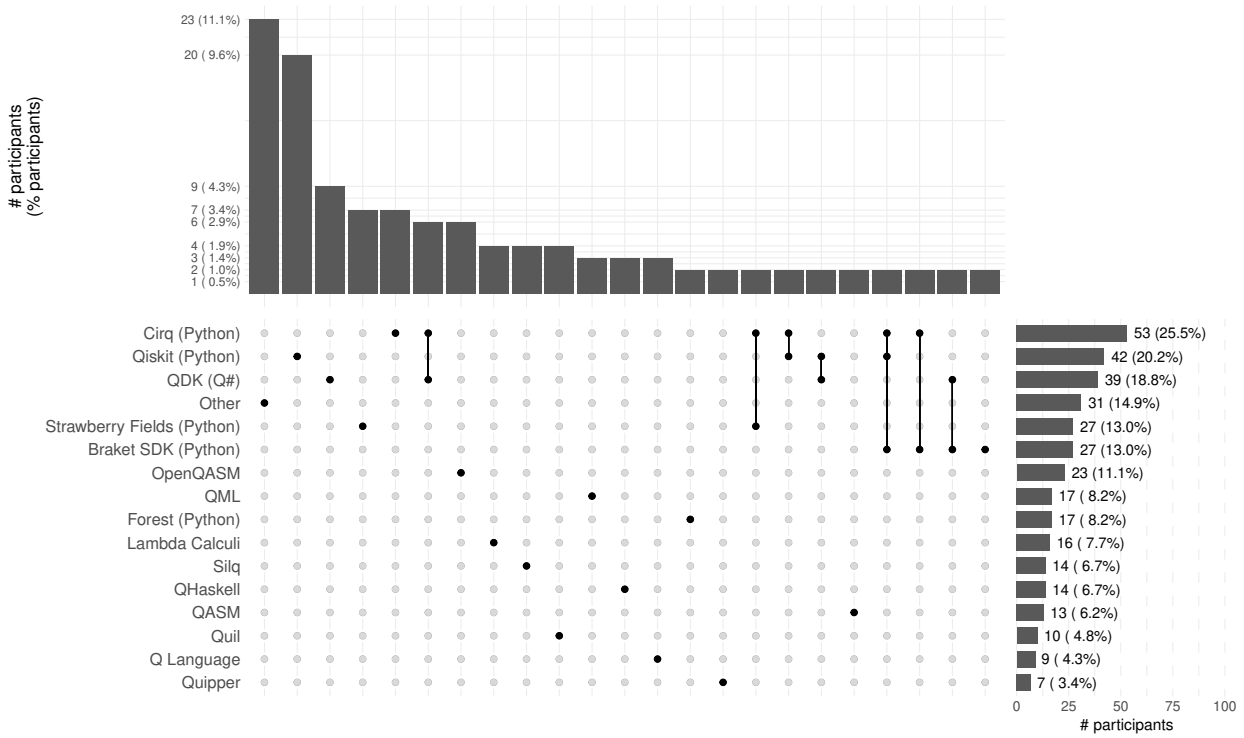

Figure 29: Quantum languages that the 208 participants would like to try in the future.
Figure discussed in RQ8 (Section 4.8).
(Please refer to Figure 6 for an explanation of the UpSet plot [88].)

Simulator as a tool to ease the testing procedure. No tool is used to test programs written with QHaskell, QASM, and Ket.

4.14. RQ14: How do users perceive the diversity of quantum languages?

42.8% of all participants reported that too many quantum languages might have been proposed, 18.3% disagreed, 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 participant mentioned that, "Too many - there's a lot of overlap between the languages, most could be done if a single language was settled on.". Participants also shared other opinions, for example:

- "There are many languages with overlapping features that are not standardized."
- "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."
- "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 35 reports the relation between quantum languages and participants' opinion regarding the number of quantum languages out there. Most participants that use Quipper, Quil, QML, Qiskit (Python), QASM, OpenQASM, Ket, DWave Ocean (Python), Cirq (Python), and Braket SDK (Python) reported that too many quantum languages might have been proposed. Participants that use Strawberry Fields (Python), QHaskell, QDK (Q#), and $Q|SI\rangle$, disagree.

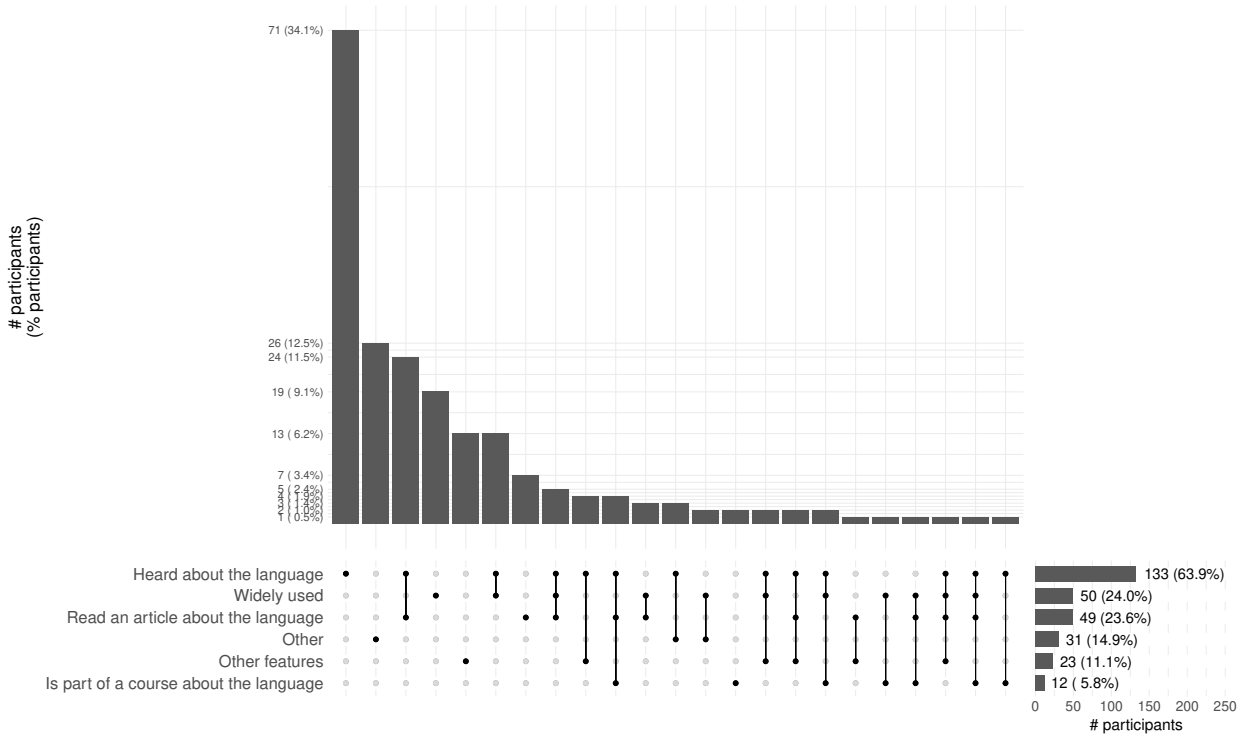

Figure 30: Why do the 208 participants like to try other quantum languages.
Figure discussed in RQ8 (Section 4.8).
(Please refer to Figure 6 for an explanation of the UpSet plot [88].)

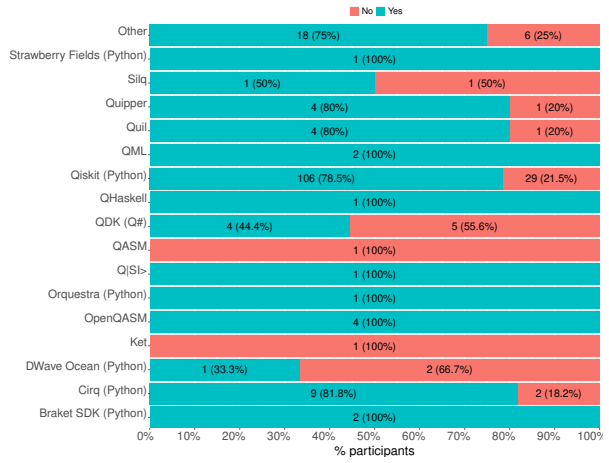

Figure 31: Whether testing is performed by quantum language. According to Fisher's exact test (p-value 0.23619), we do not reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (quantum languages and whether participants perform testing).
Figure discussed in RQ12 (Section 4.12).

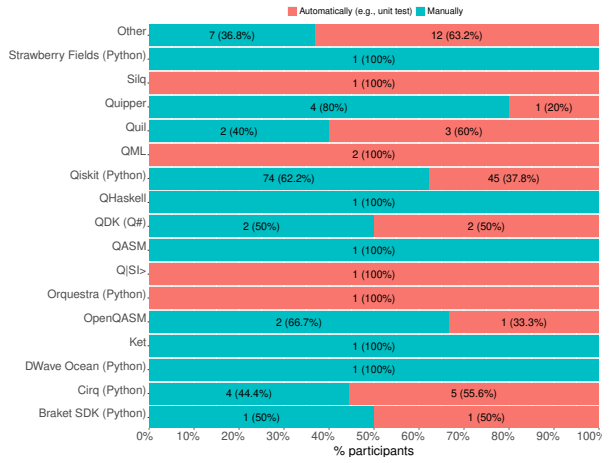

Figure 32: How is testing performed by quantum language. According to Fisher's exact test (p-value 0.28170), we do not reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (quantum languages and how participants perform testing).
Figure discussed in RQ12 (Section 4.12) and RQ13 (Section 4.13).

There are several reasons to justify the number of existing quantum languages and the development of new ones. For example, several languages could have been created for an application area, and others to take advantage of some specific quantum computing properties. Another reason might be that a recent and innovative area with great growth

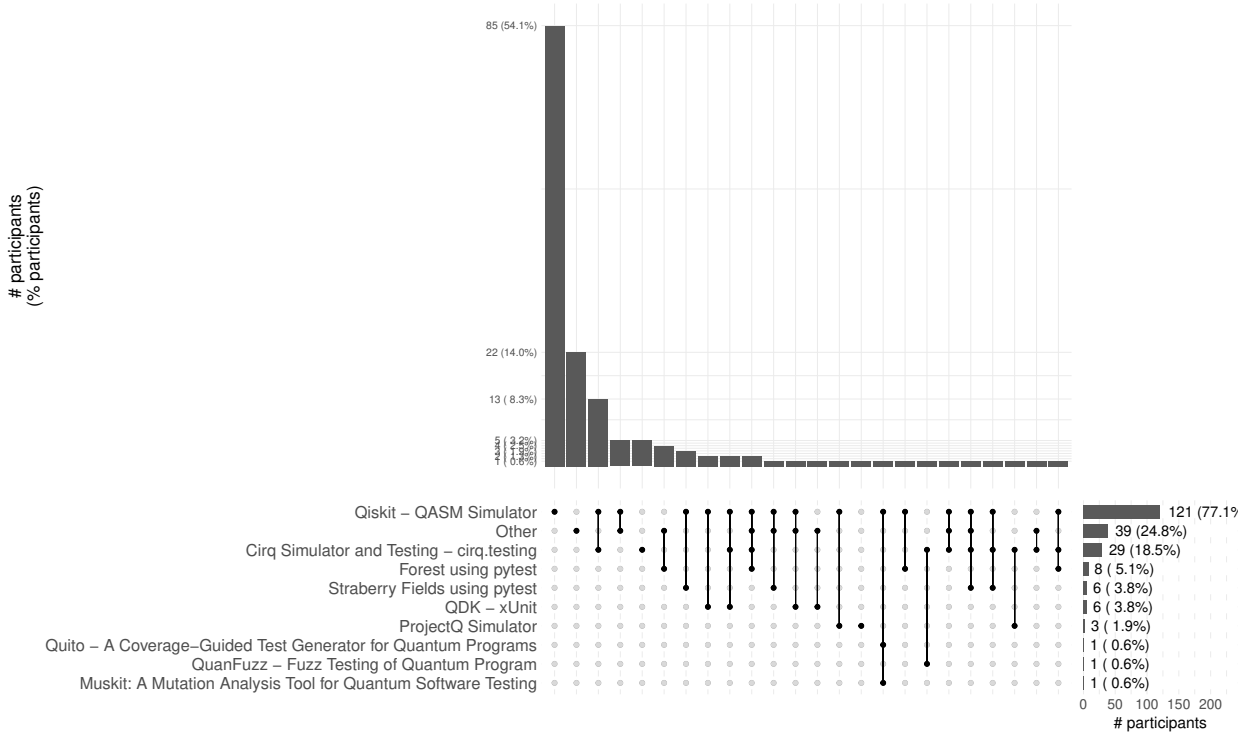

Figure 33: Tools used to test quantum programs accordingly to 157 participants. (Note that survey question number 33 (see Table A.2) was not a mandatory question and 51 participants did not answer it.)
Figure discussed in RQ13 (Section 4.13).
(Please refer to Figure 6 for an explanation of the UpSet plot [88].)

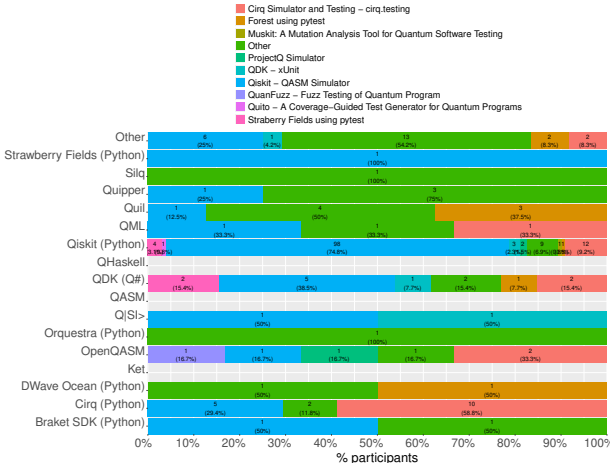

Figure 34: Quantum languages and the tool used to test quantum programs accordingly to 157 participants. According to Fisher's exact test (p-value 0.00000), we reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (quantum languages and tools used to perform testing).
Figure discussed in RQ13 (Section 4.13).

potential or new quantum hardware requires a new language. A universal language for quantum computing would be fascinating, we are not just there yet.

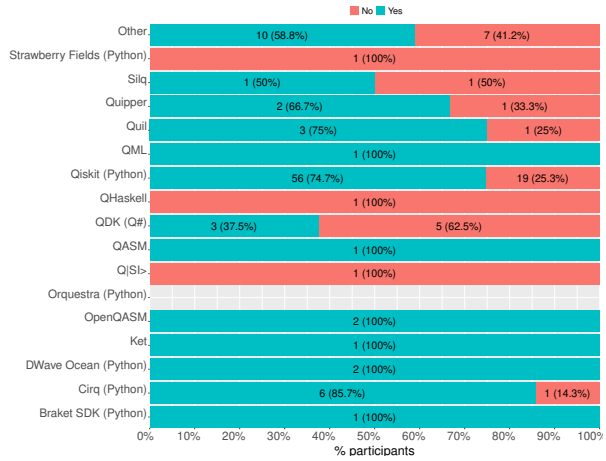

Figure 35: Participants' opinion whether there are too many quantum languages grouped by participants' primary language. According to Fisher's exact test (p-value 0.24532), we do not reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (primary quantum language and whether participants' opinion whether there are too many quantum languages).
Figure discussed in RQ14 (Section 4.14).

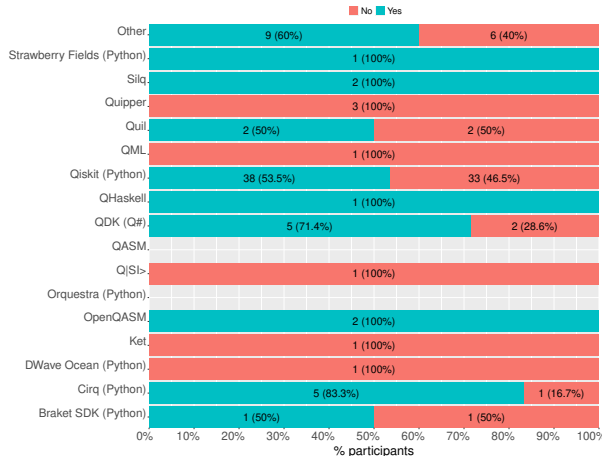

Figure 36: Participants' opinion whether another language is needed grouped by participants' primary language. According to Fisher's exact test (p-value 0.26281), we do not reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (primary quantum language and whether participants' opinion whether another language is needed).
Figure discussed in RQ15 (Section 4.15).

4.15. RQ15: What factors influence participants' opinions about the necessity of introducing new quantum languages?

Regarding the participants' opinions on whether they would need a new quantum language shortly, 31.7% of participants answered affirmatively. One participant mentioned "Yes. Higher-level programming. Right now, everything is low level with very little optimization for compilation.". 25.0% of participants disagreed, and one participant mentioned "No, it is better to learn the current languages and, if necessary, contribute to their improvement.". 43.3% did not answer.

Figure 36 shows that the participant's opinion regarding the need for another quantum language can differ depending on the language used. All the participants who use OpenQASM, QHaskell, Silq, and Strawberry Fields (Python) think there is a need to create a new language. However, the participants that use Quipper, QML, Ket, DWave Ocean (Python), and $Q|SI\rangle$ disagree.

The main arguments participants reported as being necessary for the creation of a new quantum language 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."
- "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."
- "Many languages overlap, and a standard language for quantum computation could accelerate development."
- "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 languages, the arguments are:

- "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."
- "Too many already, and consolidation is needed."
- "The popular languages like Qiskit (Python), Q#, and Cirq (Python) are enough."
- "The existing ones are sufficient."

4.16. Threats to validity

Based on the guidelines reported by Wohlin et al. [101], we discuss below the threats to the validity of our study.

4.16.1. Threats to external validity

The study may contain only some of the quantum languages ever proposed. However, extensive research was conducted to find as many languages as possible to mitigate this risk. Nevertheless, the survey allowed the participants to use the "Others" text box to inform a different programming language, and they did mention Xanadu's Pennylane [90], QUTIP [91, 92], FunQy [93], SQIR [94], Quingo [95], and Perceval [96]. Although some are available on the same data sources we used to search for quantum languages, i.e., [90] is available on arXiv, [94] and [95] are available on the ACM digital library, and [96] is available on the Quantum journal, these works did not match any of our keywords. Furthermore, [93] is only available as a webpage, and [91] is available on Elsevier ScienceDirect, both sources out of our radar. We suggest others also consider those in future studies.

The participants who carried out the survey might differ from the quantum developers out there. Still, to minimize this risk, we tried to reach as many participants as possible by publishing our survey on several quantum computing platforms with hundreds of thousands of potential participants registered. We can see in Figure 3 the diversity of countries where the participants live, in Figure 6 the diversity of majors, and in Figure 7 the current job, which might indicate the diversity of participants.

4.16.2. Threats to internal validity

Although we ensured others could replicate the search procedure described in Section 2, subjective factors during language selection may hinder the guarantee that others could produce exactly the set of quantum languages. To minimize this risk, the search procedure was conducted by both authors.

Data was analyzed with scripts written in R [102], which might have *bugs*. All authors reviewed all scripts to mitigate this risk.

4.16.3. Threats to construct validity

There is a risk that the questions used in the survey needed to be more comprehensive / deep to analyze and answer all research questions proposed in this paper. To mitigate this risk, we first analyzed the questions asked in other surveys [103, 104, 105, 106] and guidelines proposed by others [78, 79, 80] on how to carry out surveys before we designed our questions. Additionally, the survey was validated with a couple of participants before being made available to the public so we could obtain feedback and make the necessary adjustments.

5. Implications of our study and suggestions for future work

Given the results presented in the previous section, this section describes some of the implications of our study and provides suggestions for those planning to developing new quantum languages or enhancing existing ones.

5.1. Know your target audience

Two thirds of the participants are between 18 and 34 years old, live in english speakers countries (USA, India, and Canada), and have learned quantum languages from official documentation (e.g., website, markdown files) or through online courses. This implies that the developers of existing or new languages should provide content to be consumed online and by youngers in order to increse their chance of being used and further adopted. Given the many researchers using quantum languages, advertising new or existing languages at Universities in the USA, India, and Canada might also increase their chance of being used.

5.2. Documentation, examples, and community support

The most popular and used quantum languages are the ones that provide comprehensive documentation with tutorials and usage examples, and have an extensive and active community. Documentation and usage examples are a must, particularly for novices learning a new language. Documentation is not just a requirement / necessity of quantum languages, it is also a well-known case for classical language, as stated by Hope [107]. This suggests that the successful adoption of new or existing languages depends comprehensive and extensive documentation, available to the general public.

5.3. Quantum languages love classical languages

The most popular and used quantum languages are the ones that have been proposed on top of classical languages. This is not surprising. A developer would more likely (and more quickly) adopt a quantum language that follows a syntax similar to one of the developers' familiar languages, than a language with a different semantic and syntax. A quantum 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 language. That is, developers can also use all of the features of the classical language and its libraries.

5.4. High-level quantum languages

Although almost half of the participants agreed that too many programming have been proposed, one third pointed out that existing languages are very low-level and new abstractions and high-level languages must be proposed. Such high-level languages would allow developers to focus more on the quantum programs / algorithms and less on the low-level interactions with a quantum device, as it happens nowadays with classical programming languages.

5.5. Application domain and new features

Specific quantum programming languages have been used in specific domains. Such information could allow developers to refine and expand language features. For example, Strawberry Fields (Python) is exclusively used by participants with a major in physics. Thus, developers of Strawberry Fields (Python) should aim to collect feedback from physicists and to improve or support more physics-related features.

5.6. Tailored quantum integrated development

Jetbrains' annual report [104] reported that most developers now use Visual Studio Code or IntelliJ IDEA as their integrated development environment. The question is whether those environments have the right tools to write a quantum program with a quantum language (i.e., simple support as source code highlight and auto-complete), to run and debug a quantum program, or to test a quantum program.

For example, Qiskit provides a drawing functionality that allows one to translate a given quantum circuit into a graphical image of the circuit. Such feature, if integrated into a development environment, would allow quantum experts, either developers or not, to understand and discuss the code by just looking at the graphical representation of the circuit. Others have explored this feature, for example, to study quantum smells [108] (i.e., bad coding practices) and to perform mutation testing [109, 110, 111]. None has not yet been integrated in any development environment.

5.7. Testing & Debugging tools

Although several tools have been proposed to ease the task of testing a quantum program, more than half of the participants stated that they test their programs manually. This might indicate a lack of dissemination of existing testing tools or the need to make the existing ones easier to use. Participants also pointed out that debugging tools (integrated in the development environment) would be a must.

5.8. Real quantum computers

Although many developers might be willing to learn and try new quantum languages, the fact that most of the development still occurs in simulators might be a downside and a showstopper for some. Companies like Google, IBM, and D-Wave Systems which are developing quantum languages and quantum computers, should foster access to their computers so that developers can try their quantum programs on real devices and try to reach the fully expected potential behind quantum computing.

6. Related work

In this section, we surveyed the literature that is most related to our main contributions or that is in line with the research topics studied in this paper. Selinger [112] provide a brief review of quantum language research. Unruh [113], Sofge [71], Rojas [114] provide an overview of the development of quantum languages. De Stefano et al. [115], the most similar work to this paper, propose a taxonomy of the quantum technologies used in open-source quantum projects and investigate the most used quantum programming frameworks.

6.1. Classical programming languages

According to the online historical encyclopedia of programming languages HOPL [116], nearly 9000 languages have been created since the 18^{th} century. However, according to GitHub [117] only about 370 are still active.

According to Lagutin [118], 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 [119]'s stack overflow post, 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*.

Scott [120] pointed out that 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.

6.2. Quantum programming languages

6.2.1. Towards a quantum programming language by Selinger (2004)

According to Selinger [63], 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 [63] proposed the design of a quantum language, called QPL, defining the syntax and semantics of a functional quantum 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 [63] 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 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.

6.2.2. A brief survey of quantum programming languages by Selinger (2004)

Selinger [112] provided a brief review of quantum 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 [121], 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 [122] with the definition of coherent quantum spaces as possible semantics for higher-order quantum computation. Abramsky and Coecke [123], which models a high-order function and applications that rely on entanglement and quantum measurement. Edalat [124] with a domain-theoretic interpretation of Gleason's theorem and Coecke and Martin [125] with a domain-theoretic treatment of the von Neumann entropy of a quantum state.

Selinger [112] further raised three challenges for quantum 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 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.

6.2.3. Quantum programming language by Unruh (2006)

Unruh [113] investigated the development of quantum 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 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 [113] separated the quantum languages by the syntax and semantics. The author also presented possible features of quantum 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.

6.2.4. Quantum programming language, survey and bibliography by Gay (2006)

Gay [126] briefly summarized the basic concepts of quantum computing, surveyed the literature on quantum 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 [127] as the first model for general quantum computation, which has the property of superposition of machine states. Gay also described the work of Knill [121] that defined a proposal for a formalized quantum 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 languages as QCL [67], Q [72], and qGCL [73].

For the functional languages and Lambda-Calculi, the author summarized both extensions of λ-calculus of Maymin [128], the quantum λ-calculus of Van Tonder [74] and also the work on the definition of the first-order functional programming language QML of Altenkich et al. [60]. He also mentions several works that investigated quantum programming within Haskell.

For the other languages paradigms, he summarized the work of Gay and Nagarajan [61], the definition of the process calculus CQP (Communication Quantum Process), and Jorrand and Lalire [129], 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 languages. The work proposed by Altenkirch and Grattage [60] have developed a compiler for their QML language into a representation of quantum circuits, using categorical semantics as an intermediate form.

6.2.5. A survey of quantum programming languages: history, methods, and tools by Sofge (2008)

Sofge [71] researched some of the essential quantum 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 [130] in 1987, which played an essential role in designing quantum languages, particularly those based on lambda calculus.

According to Donald, the first step to creating a quantum 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 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 languages, grouping them into three different types: imperative, functional, and other quantum languages (which include mathematical formalism that was not defined to run on computers).

The author defined some challenges in quantum 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.

6.2.6. The modern state of quantum programming language by Rojas (2019)

Rojas [114] summarized the core ideals found in a successful quantum 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 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.

6.2.7. Quantum programming languages: a systematic review of research topic and top cited languages by Garhwal et al. (2019)

Garhwal et al. [131] gave an overview of the state of the art in the field of quantum languages and focused on actual high-level quantum 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 languages; which major Companies, Groups, Institutes, and Universities are working on developing new quantum languages; what are the most popular publication venues for quantum; and what are the most cited papers in the area of a quantum language).

The authors divided the types of quantum 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 languages. However, many research groups are working in the area of quantum languages. Garhwal et al. [131] 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 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.

6.2.8. Quantum software engineering: landscapes and horizons by Zhao (2021)

Section 14.1 of the first comprehensive survey on the research of quantum software engineering conducted by Zhao [5], discusses some works in quantum programming languages also discussed in this section, in particular [112, 126, 113, 71, 131, 62].

6.2.9. Quantum software components and platforms: Overview and quality assessment by Serrano et al. (2022)

Serrano et al. [132] gathered and discussed several quantum languages that have been proposed and grouped them by paradigm, either imperative, functional, or other; similar to our comprehensive analyze in Sections 2.2 to 2.4. Figure 5 in [132] shows a roadmap of quantum programming languages similar to our Figure 1. Our figure complements the one in [132] as it shows more languages, whether a language is functional, imperative, or multi-paradigm, and it also shows whether a language has evolved from any other language (either classical or quantum).

6.2.10. Software engineering for quantum programming: how far are we? by De Stefano et al. (2022)

De Stefano et al. [115] mined project repositories on GitHub that use one of three quantum languages (Qiskit, Cirq, and Q#)[10], and defined a taxonomy of quantum technologies used by those projects. They also conducted a survey with developers that uses quantum languages to get their opinion on the current adoption and challenges of the quantum programming field.

[10]De Stefano et al. [115] stated that Qiskit, Cirq, and Q# are the most widely used and our study confirmed that.

They concluded that (i) most of the quantum technologies are being used for personal projects (41%, which is inline with our results 48%, see Figure 27), and (ii) the main challenges related to quantum frameworks are the comprehension of quantum programs and difficulty setting up hardware and software infrastructures.

7. 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 paper, we conducted an exploratory study on the usage of quantum languages. Firstly, we conducted an extensive study of quantum languages' state of the art and then described each one. Secondly, we surveyed 251 participants on the usage of quantum languages. This allowed us to gather data to answer our set of research questions. Thirdly, we presented our results regarding 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 shortly, and the opinion of the participants about the number of quantum languages available, and the need to create new languages. We then concluded that quantum languages are mainly used for learning and research, and that the most popular quantum languages are built on top of the Python classical language. Finally, we provided a few suggestions for developing a quantum language or improving any existing language.

As future work, we suggest extending this study (i) with additional quantum languages that were not included, and (ii) with additional new questions, which were only identified later in our study and therefore were not included, e.g., *what type of quantum projects are developers working on? what type of quantum programs and/or algorithms are developers writing with quantum languages?*, These and other questions might further shed light on the day-to-day usage of quantum languages and the development of quantum programs.

Data availability

The data generated in this paper and the R scripts created to analyze that data are available at `https://github.com/jose/quantum-languages-data`.

Acknowledgments

We would like to express our gratitude to André Souto (Faculdade de Ciências da Universidade de Lisboa, Portugal) and the anonymous reviewers for their feedback on early drafts of this work. This work was supported by Fundação para a Ciência e Tecnologia (FCT) through the LASIGE Research Unit, ref. UIDB/00408/2020 (`https://doi.org/10.54499/UIDB/00408/2020`) and ref. UIDP/00408/2020 (`https://doi.org/10.54499/UIDP/00408/2020`).

Appendix A. Survey questions

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

#	Survey question	RQ	Reason	Answer type	Possible answers
			Section 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	Radio button (Single Choice)	Yes; No
			Section 2		
2	What is your age?*	**RQ1**	This question was chosen to identify the participants of this survey demographically.	Dropdown (Single Choice)	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)*	**RQ1**	Identify where the participants are geographically concentrated.	Dropdown (Single Choice)	Brazil; Portugal; Spain; etc
4	Which of the following describe you?	**RQ1**	Identify the gender of the participants.	Radio button (Single Choice)	Man; Woman; Non-binary, genderqueer, or gender non-conforming; Prefer not to say; Other
			Section 3		

5	How many years have you been coding?*	**RQ3**	Assess the experience and education of the participants in terms of coding.	Dropdown (Single Choice)	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)?*	**RQ3**	Assess the professional experience of the participants.	Dropdown (Single Choice)	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.	**RQ2**	Assess the education of the participants.	Checkboxes (Multiple Choices)	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.	**RQ4**	Identify the languages that the participant has used.	Checkboxes (Multiple Choices)	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?*	**RQ1**	Assess level of education in terms of knowledge in quantum physics.	Radio button (Single Choice)	0 (no knowledge); 1 (novice); 2; 3; 4; 5 (expert)
10	Where did you learn Quantum Physics?	**RQ1**	Assess the education of the participants in terms of learning quantum physics.	Checkboxes (Multiple Choice)	Books; Online Course; Search Sites; University; Work; Other
11	Which of the following best describes the highest level of education that you have completed?*	**RQ1**	Identify the formal education of the participants.	Radio button (Single Choice)	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?	**RQ1**	Assess the work field of the participants.	Checkboxes (Multiple Choices)	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.	**RQ1**	Identify the roles of the participants.	Checkboxes (Multiple Choices)	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
			Section 4		
14	Where and how did you learn Quantum Programming Languages?*	**RQ2**	Assess the education of the participants in terms of learning quantum programming languages.	Checkboxes (Multiple Choices)	Books; Language documentation; University; Online Course; Online Forums; Search Sites; Work; Other
15	How many years have you been coding using Quantum Programming Languages?*	**RQ3**	Assess the experience and education of the participants in using quantum programming languages.	Dropdown (Single Choice)	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)?*	**RQ3**	Assess the professional experience of the participants regarding quantum programming languages.	Dropdown (Single Choice)	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?	**RQ4**	Assess which quantum programming languages the participants use and how long.	Radion button (Multiple Choice Grid)	Rows (Blackbird; Braket SDK; Cirq; Cove; cQASM; CQP (Communication Quantum Processes); cQPL; Forest; Ket; LanQ; $LIQUi\vert\rangle$); 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#; $Q\vert SI\rangle$; 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?	**RQ4**	Identify other quantum programming languages not listed that the participant used.	Open Text	-
19	Which of the following is your primary Quantum Programming Languages?*	**RQ5**	Identify the most used quantum programming language by the participants.	Dropdown (Single Choice)	Blackbird; Braket SDK; Cirq; Cove; cQASM; CQP (Communication Quantum Processes); cQPL; Forest; Ket; LanQ; $LIQUi\vert\rangle$); 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#; $Q\vert SI\rangle$; Sabry's Language; Scaffold; Silq; Strawberry Fields; λ_q (Lambda Calculi); Other
20	In terms of ease, rate your primary Quantum Programming Language.*	**RQ5**	Rate the main characteristics of the participant's favorite quantum programming languages.	Radio button (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?	**RQ5**	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?	**RQ5**	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)	**RQ5**	Evaluate the most used forums to ask questions on quantum computing.	Checkboxes (Multiple Choices)	Devtalk; Quantum Open Source Foundation; Slack; StackOverflow; Other

24	Which Quantum Programming Languages would you like to work or try in the near future?*	**RQ8**	Identify which is the most like quantum programming language to be used in the future.	Checkboxes (Multiple Choice)	Blackbird; Braket SDK; Cirq; Cove; cQASM; CQP (Communication Quantum Processes); cQPL; Forest; Ket; LanQ; *LIQUi|⟩*; 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⟩*; Sabry's Language; Scaffold; Silq; Strawberry Fields; λ_q (Lambda Calculi); Other
25	Why would you like to work or try those languages?*	**RQ8**	The reason why the participant wants to work with the quantum programming language.	Checkboxes (Multiple Choice)	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?*	**RQ10**	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?*	**RQ9**	Assess why the participants want to learn quantum programming languages.	Open Text	-
			Section 5		
28	How do you use Quantum Programming Languages?*	**RQ7**	Assess how the participants use quantum programming languages.	Radio button (Single 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?*	**RQ11**	Evaluate what tools are missing in the quantum programming languages.	Open Text	-
30	Do you test your Quantum Programs?*	**RQ12**	Identify if the participants perform tests.	Radio button (Single Choice)	Yes; No
31	How often do you test your Quantum Programs?	**RQ12**	Evaluate the frequency that the participant tests their quantum programs.	Radio button (Single Choice)	Before go to production; Every day; Every time you change the code; Other
32	How do you test your Quantum Programs?	**RQ12**	Identity if the participants use automatic or manual tests	Radio button (Single Choice)	Automatically (e.g., unit test); Manually
33	What tools do you use to test your Quantum Programs?	**RQ13**	Identify what the most used tools to test quantum programs are.	Checkboxes (Multiple Choice)	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
			Section 6		

34	In your opinion, do you think there are too many or too few Quantum Programming Languages? Why?	**RQ14**	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?	**RQ15**	Assess if the participant thinks that it is necessary the development of another quantum programming language	Open Text	-

Table A.2: Survey questions. * Indicates a mandatory question.

Appendix B. 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 B.3: Social networks contact for the survey. Information was obtained in April 2022.

Appendix C. RQ4 — Additional artifacts

- Table C.4 reports the raw number of participants per quantum language by usage time.

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 C.4: Number of participants per quantum language by usage time.

Appendix D. New tables

- Figure 18 correspondent table in Table D.5.
- Figures 19 and 20 correspondent tables in Tables D.6 and D.7.
- Figures 21 and 22 correspondent tables in Tables D.8 and D.9.
- Figures 23 and 24 correspondent tables in Tables D.10 and D.11.
- Figure 25 correspondent table in Table D.12.
- Figure 34 correspondent table in Table D.13.

	1	2	3	4	5
support	8 (5.9%)	19 (4.1%)	48 (35.6%)	31 (23.0%)	29 (21.5%)
forums	10 (7.4%)	22 (16.3%)	41 (30.4%)	36 (26.7%)	26 (19.3%)
features	5 (3.7%)	14 (10.4%)	39 (28.9%)	48 (35.6%)	29 (21.5%)
easy to code	5 (3.7%)	7 (5.2%)	41 (30.4%)	48 (35.6%)	34 (25.2%)
documentation	7 (5.2%)	9 (6.7%)	28 (20.7%)	47 (34.8%)	44 (32.6%)
code examples	7 (5.2%)	11 (8.1%)	34 (25.2%)	49 (36.3%)	34 (25.2%)

Table D.5: The rate in terms of ease (e.g., features, easy to code, documentation, code examples, support, forums) of Qiskit, the primary quantum language of 135 out of the 208 participants (see Figure 17).
Table discussed in RQ5 (Section 4.5).

	Other	StackOverflow	Slack	Quantum Open Source Foundation	Devtalk
Other	8 (25.0%)	12 (40.6%)	7 (21.9%)	3 (9.4%)	1 (3.1%)
Strawberry Fields (Python)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Silq	2 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Quipper	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Quil	1 (50.0%)	1 (50.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
QML	0 (0.0%)	2 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Qiskit (Python)	14 (6.8%)	94 (45.9%)	65 (31.7%)	31 (15.1%)	1 (0.5%)
QHaskell	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
QDK (Q#)	2 (18.2.%)	6 (54.5%)	1 (9.1%)	2 (18.2%)	0 (0.0%)
QASM	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Q|SI>	1 (50.0%)	1 (50.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Orquestra (Python)	0 (0.0%)	0 (0.0%)	1 (50.0%)	1 (50.0%)	0 (0.0%)
OpenQASM	0 (0.0%)	4 (50.0%)	2 (25.0%)	1 (12.5%)	1 (12.5%)
Ket	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
DWave Ocean (Python)	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Cirq (Python)	1 (11.1%)	5 (55.6%)	3 (33.3%)	0 (0.0%)	0 (0.0%)
Braket SDK (Python)	0 (0.0%)	1 (25.0%)	2 (50.0%)	1 (25.0%)	0 (0.0%)

Table D.6: Participants' primary quantum language and the forums used by them. According to Fisher's exact test (p-value 0.05942), we do not reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (primary quantum language and forums used).
Table discussed in RQ5 (Section 4.5).

	Other	Work	University	Search Sites	Online Forums	Online Course	Language documentation	Books
Other	5 (9.3%)	11 (20.4%)	3 (5.6%)	7 (13.0%)	5 (9.3%)	3 (5.6%)	15 (27.8%)	5 (9.3%)
Strawberry Fields (Python)	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Silq	0 (0.0%)	2 (22.2%)	1 (11.1%)	1 (11.1%)	1 (11.1%)	1 (11.1%)	2 (22.2%)	1 (11.1%)
Quipper	3 (33.3%)	2 (22.2%)	1 (11.1%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	2 (22.2%)	1 (11.1%)
Quil	1 (10.0%)	4 (40.0%)	0 (0.0%)	1 (10.0%)	0 (0.0%)	0 (0.0%)	3 (30.0%)	1 (10.0%)
QML	0 (0.0%)	0 (0.0%)	1 (16.7%)	1 (16.7%)	1 (16.7%)	0 (0.0%)	2 (33.3%)	1 (16.7%)
Qiskit (Python)	6 (1.7%)	23 (6.4%)	35 (9.7%)	42 (11.6%)	46 (12.7%)	71 (19.7%)	78 (21.6%)	60 (16.6%)
QHaskell	0 (0.0%)	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
QDK (Q#)	0 (0.0%)	3 (10.7%)	2 (7.1%)	5 (17.9%)	3 (10.7%)	2 (7.1%)	7 (25.0%)	6 (21.4%)
QASM	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (100.0%)
Q|SI>	1 (20.0%)	0 (0.0%)	1 (20.0%)	1 (20.0%)	1 (20.0%)	0 (0.0%)	1 (20.0%)	0 (0.0%)
Orquestra (Python)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (100.0%)	0 (0.0%)
OpenQASM	1 (10.0%)	2 (20.0%)	1 (10.0%)	1 (10.0%)	0 (0.0%)	1 (10.0%)	2 (20.0%)	2 (20.0%)
Ket	0 (0.0%)	0 (0.0%)	1 (33.3%)	1 (33.3%)	0 (0.0%)	0 (0.0%)	1 (33.3%)	0 (0.0%)
DWave Ocean (Python)	0 (0.0%)	2 (22.2%)	1 (11.1%)	1 (11.1%)	1 (11.1%)	2 (22.2%)	2 (22.2%)	0 (0.0%)
Cirq (Python)	1 (3.1%)	4 (12.5%)	6 (18.8%)	1 (3.1%)	3 (9.4%)	5 (15.6%)	9 (28.1%)	3 (9.4%)
Braket SDK (Python)	0 (0.0%)	1 (16.7%)	0 (0.0%)	1 (16.7%)	2 (33.3%)	1 (16.7%)	1 (16.7%)	0 (0.0%)

Table D.7: Participants' primary quantum language and how they learned it. According to Fisher's exact test (p-value 0.01074), we reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (primary quantum language and how participants learned it).
This table reports the intersection between Figures 10 and 17's data, and it is discussed in RQ6 (Section 4.6).

	Other	Software Engineering	Social Sciences	Physics	Other Engineering	Math	Economics	Computer Science	Art / Humanities
Other	0 (0.0%)	1 (3.0%)	1 (3.0%)	12 (36.4%)	3 (9.1%)	7 (21.2%)	1 (3.0%)	8 (24.2%)	0 (0.0%)
Strawberry Fields (Python)	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Silq	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (50.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (50.0%)	0 (0.0%)
Quipper	1 (16.7%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	3 (50.0%)	0 (0.0%)	2 (33.3%)	0 (0.0%)
Quil	0 (0.0%)	1 (16.7%)	0 (0.0%)	1 (16.7%)	1 (16.7%)	1 (16.7%)	0 (0.0%)	2 (33.3%)	0 (0.0%)
QML	0 (0.0%)	1 (50.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (50.0%)	0 (0.0%)
Qiskit (Python)	11 (7.1%)	16 (10.4%)	3 (1.9%)	46 (29.9%)	9 (5.8%)	11 (7.1%)	1 (0.6%)	54 (35.1%)	3 (1.9%)
QHaskell	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
QDK (Q#)	0 (0.0%)	2 (18.2%)	0 (0.0%)	2 (18.2%)	1 (9.1%)	1 (9.1%)	0 (0.0%)	5 (45.5%)	0 (0.0%)
QASM	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (100.0%)	0 (0.0%)
Q|SI>	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (50.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (50.0%)	0 (0.0%)
Orquestra (Python)	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (50.0%)	1 (50.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
OpenQASM	0 (0.0%)	2 (25.0%)	0 (0.0%)	0 (0.0%)	2 (25.0%)	0 (0.0%)	0 (0.0%)	4 (50.0%)	0 (0.0%)
Ket	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (100.0%)	0 (0.0%)
DWave Ocean (Python)	1 (20.0%)	1 (20.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (20.0%)	0 (0.0%)	2 (40.0%)	0 (0.0%)
Cirq (Python)	1 (5.9%)	1 (5.9%)	0 (0.0%)	6 (35.3%)	0 (0.0%)	1 (5.9%)	1 (5.9%)	7 (41.2%)	0 (0.0%)
Braket SDK (Python)	1 (50.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (50.0%)

Table D.8: Participants' primary quantum language and their major. According to Fisher's exact test (p-value 0.27723), we do not reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (primary quantum language and participants' major).
This table reports the intersection between Figures 6 and 17's data, and it is discussed in RQ6 (Section 4.6).

	1 (novice)	2	3	4	5 (expert)
Other	3 (12.5%)	6 (25.0%)	3 (12.5%)	4 (16.7%)	8 (33.3%)
Strawberry Fields (Python)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (100.0%)
Silq	0 (0.0%)	0 (0.0%)	1 (50.0%)	1 (50.0%)	0 (0.0%)
Quipper	0 (0.0%)	2 (40.0%)	0 (0.0%)	1 (20.0%)	2 (40.0%)
Quil	0 (0.0%)	2 (40.0%)	1 (20.0%)	0 (0.0%)	2 (40.0%)
QML	1 (50.0%)	1 (50.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Qiskit (Python)	28 (20.7%)	29 (21.5%)	37 (27.4%)	22 (16.3%)	19 (14.1%)
QHaskell	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (100.0%)	0 (0.0%)
QDK (Q#)	4 (44.4%)	1 (11.1%)	3 (33.3%)	0 (0.0%)	1 (11.1%)
QASM	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Q|SI>	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Orquestra (Python)	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (100.0%)	0 (0.0%)
OpenQASM	0 (0.0%)	1 (25.0%)	2 (50.0%)	0 (0.0%)	1 (25.0%)
Ket	0 (0.0%)	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)
DWave Ocean (Python)	1 (33.3%)	0 (0.0%)	1 (33.3%)	1 (33.3%)	0 (0.0%)
Cirq (Python)	0 (0.0%)	3 (27.3%)	2 (18.2%)	1 (9.1%)	5 (45.5%)
Braket SDK (Python)	2 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)

Table D.9: Participants' primary quantum language and their knowledge in quantum physics. According to Fisher's exact test (p-value 0.12501), we do not reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (primary quantum language and participant's knowledge in quantum physics).
Table discussed in RQ6 (Section 4.6).

	Less than 1 year	1 to 4 years	5 to 9 years	10 to 14 years	15 to 19 years	20 to 24 years
Other	5 (20.8%)	12 (50.0%)	6 (25.0%)	1 (4.2%)	0 (0.0%)	0 (0.0%)
Strawberry Fields (Python)	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Silq	0 (0.0%)	2 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Quipper	0 (0.0%)	1 (20.0%)	1 (20.0%)	2 (40.0%)	0 (0.0%)	1 (20.0%)
Quil	0 (0.0%)	4 (80.0%)	1 (20.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
QML	1 (50.0%)	1 (50.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Qiskit (Python)	36 (26.7%)	81 (60.0%)	16 (11.9%)	2 (1.5%)	0 (0.0%)	0 (0.0%)
QHaskell	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
QDK (Q#)	5 (55.6%)	3 (33.3%)	1 (11.1%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
QASM	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Q|SI>	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Orquestra (Python)	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
OpenQASM	1 (25.0%)	2 (50.0%)	1 (25.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Ket	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
DWave Ocean (Python)	1 (33.3%)	2 (66.7.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Cirq (Python)	2 (18.2%)	7 (63.6%)	1 (9.1%)	1 (9.1%)	0 (0.0%)	0 (0.0%)
Braket SDK (Python)	1 (50.0%)	1 (50.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)

Table D.10: Participants' primary quantum language and their personal experience with quantum languages. According to Fisher's exact test (p-value 0.20924), we do not reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (primary quantum language and participants' personal experience).
Note: Ranges 25 to 29 years, 30 to 34, 35 to 39, 40 to 44, 45 to 49, and more than 50 years were excluded from this table because (i) there was no answer on those ranges and (ii) to improve table's readability.
This table reports the intersection between Figures 13 and 17's data, and it is discussed in RQ6 (Section 4.6).

	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
Other	7 (29.2%)	6 (25.0%)	5 (20.8%)	5 (20.8%)	1 (4.2%)	0 (0.0%)	0 (0.0%)
Strawberry Fields (Python)	0 (0.0%)	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Silq	0 (0.0%)	0 (0.0%)	2 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Quipper	2 (40.0%)	0 (0.0%)	0 (0.0%)	1 (20.0%)	1 (20.0%)	0 (0.0%)	1 (20.0%)
Quil	0 (0.0%)	2 (40.0%)	2 (40.0%)	1 (20.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
QML	1 (50.0%)	1 (50.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Qiskit (Python)	56 (41.5%)	33 (24.4%)	35 (25.9%)	9 (6.7%)	2 (1.5%)	0 (0.0%)	0 (0.0%)
QHaskell	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
QDK (Q#)	7 (77.8%)	0 (0.0%)	1 (11.1%)	1 (11.1%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
QASM	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Q|SI>	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Orquestra (Python)	0 (0.0%)	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
OpenQASM	1 (50.0%)	0 (0.0%)	1 (50.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Ket	0 (0.0%)	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
DWave Ocean (Python)	0 (0.0%)	2 (66.7%)	1 (33.3%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Cirq (Python)	1 (9.1%)	2 (18.2%)	6 (54.5%)	1 (9.1%)	1 (9.1%)	0 (0.0%)	0 (0.0%)
Braket SDK (Python)	1 (50.0%)	1 (50.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)

Table D.11: Participants' primary quantum language and their professional experience with quantum languages. According to Fisher's exact test (p-value 0.20924), we do not reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (primary quantum language and participant's professional experience with quantum languages).
Note: Ranges 25 to 29 years, 30 to 34, 35 to 39, 40 to 44, 45 to 49, and more than 50 years were excluded from this table because (i) there was no answer on those ranges and (ii) to improve table's readability. This table reports the intersection between Figures 14 and 17's data, and it is discussed in RQ6 (Section 4.6).

	Like to learn	Use it for research	Use it for work	Other
Other	7 (29.2%)	7 (29.2%)	6 (25.0%)	4 (16.7%)
Strawberry Fields (Python)	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)
Silq	0 (0.0%)	2 (100.0%)	0 (0.0%)	0 (0.0%)
Quipper	0 (0.0%)	5 (100.0%)	0 (0.0%)	0 (0.0%)
Quil	0 (0.0%)	0 (0.0%)	4 (80.0%)	1 (20.0%)
QML	0 (0.0%)	1 (50.0%)	1 (50.0%)	0 (0.0%)
Qiskit (Python)	51 (37.8%)	58 (43.0%)	20 (14.8%)	6 (4.4%)
QHaskell	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
QDK (Q#)	5 (55.6%)	4 (44.4%)	0 (0.0%)	0 (0.0%)
QASM	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)
Q|SI>	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Orquestra (Python)	0 (0.0%)	0 (0.0%)	1 (100.0%)	0 (0.0%)
OpenQASM	2 (50.0%)	0 (0.0%)	1 (25.0%)	1 (25.0%)
Ket	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)
DWave Ocean (Python)	2 (66.7%)	1 (33.3%)	0 (0.0%)	0 (0.0%)
Cirq (Python)	3 (27.3%)	8 (72.7%)	0 (0.0%)	0 (0.0%)
Braket SDK (Python)	0 (0.0%)	0 (0.0%)	1 (50.0%)	1 (50.0%)

Table D.12: Participants' primary quantum language and for what they use it. According to Fisher's exact test (p-value 0.00185), we reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (primary quantum language and their usage).
Table discussed in RQ7 (Section 4.7).

	Cirq Simulator and Testing	Forest using pytest	Muskit	Other	ProjectQ Simulator	QDK–xUnit	Qiskit–QASM Simulator	QuanFuzz	Quito	Straberry Fields using pytest
Other	2 (8.3%)	2 (8.3%)	0 (0.0%)	13 (54.2%)	0 (0.0%)	1 (4.2%)	6 (25.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Strawberry Fields (Python)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Silq	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Quipper	0 (0.0%)	0 (0.0%)	0 (0.0%)	3 (75.0%)	0 (0.0%)	0 (0.0%)	1 (25.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Quil	0 (0.0%)	3 (37.5%)	0 (0.0%)	4 (50.0%)	0 (0.0%)	0 (0.0%)	1 (12.5%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
QML	1 (33.3%)	0 (0.0%)	0 (0.0%)	1 (33.3%)	0 (0.0%)	0 (0.0%)	1 (33.3%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Qiskit (Python)	12 (9.2%)	1 (0.8%)	1 (0.8%)	9 (6.9%)	2 (1.5%)	3 (2.3%)	98 (74.8%)	0 (0.0%)	1 (0.8%)	4 (3.1%)
QHaskell	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
QDK (Q#)	2 (15.4%)	1 (7.7%)	0 (0.0%)	2 (15.4%)	0 (0.0%)	1 (7.7%)	5 (38.5%)	0 (0.0%)	0 (0.0%)	2 (15.4%)
QASM	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Q|SI>	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (50.0%)	1 (50.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Orquestra (Python)	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
OpenQASM	2 (33.3%)	0 (0.0%)	0 (0.0%)	1 (16.7%)	1 (16.7%)	0 (0.0%)	1 (16.7%)	1 (16.7%)	0 (0.0%)	0 (0.0%)
Ket	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
DWave Ocean (Python)	0 (0.0%)	1 (50.0%)	0 (0.0%)	1 (50.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Cirq (Python)	10 (58.8%)	0 (0.0%)	0 (0.0%)	2 (11.8%)	0 (0.0%)	0 (0.0%)	5 (29.4%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Braket SDK (Python)	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (50.0%)	0 (0.0%)	0 (0.0%)	5 (50.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)

Table D.13: Quantum languages and the tool used to test quantum programs accordingly to 157 participants. According to Fisher's exact test (p-value 0.00000), we reject the null hypothesis, i.e., that there is a significant relationship between the two categorical variables (quantum languages and tools used to perform testing).
Table discussed in RQ13 (Section 4.13).

References

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

[2] C. H. Bennett, G. Brassard, Quantum cryptography: Public key distribution and coin tossing, Theoretical Computer Science 560 (2014) 7–11. URL: https://doi.org/10.1016%2Fj.tcs.2014.05.025. doi:10.1016/j.tcs.2014.05.025.

[3] R. Barends, J. Kelly, A. Megrant, A. Veitia, D. Sank, E. Jeffrey, T. C. White, J. Mutus, A. G. Fowler, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, C. Neill, P. O'Malley, P. Roushan, A. Vainsencher, J. Wenner, A. N. Korotkov, A. N. Cleland, J. M. Martinis, Superconducting quantum circuits at the surface code threshold for fault tolerance, Nature 508 (2014) 500–503. URL: https://doi.org/10.1038/nature13171. doi:10.1038/nature13171.

[4] M. Benedetti, J. Realpe-Gómez, R. Biswas, A. Perdomo-Ortiz, Estimation of effective temperatures in quantum annealers for sampling applications: A case study with possible applications in deep learning, Phys. Rev. A 94 (2016) 022308. URL: https://link.aps.org/doi/10.1103/PhysRevA.94.022308. doi:10.1103/PhysRevA.94.022308.

[5] J. Zhao, Quantum Software Engineering: Landscapes and Horizons, 2021. URL: https://arxiv.org/abs/2007.07047. arXiv:2007.07047.

[6] M. A. Nielsen, I. L. Chuang, Quantum computation and quantum information, Cambridge university press, 2010.

[7] P. Mateus, A. Sernadas, A. Souto, Universality of quantum Turing machines with deterministic control, Journal of Logic and Computation 27 (2017) 1–19. doi:10.1093/logcom/exv008.

[8] A. W. Cross, L. S. Bishop, J. A. Smolin, J. M. Gambetta, Open Quantum Assembly Language, arXiv preprint arXiv:1707.03429 (2017). URL: https://arxiv.org/abs/1707.03429.

[9] Microsoft, Azure quantum documentation, https://docs.microsoft.com/en-us/azure/quantum/?view=qsharp-preview, 2021. [Online; accessed December-2021].

[10] Google, Cirq - an open source framework for programming quantum computers, https://quantumai.google/cirq, 2019. [Online; accessed March-2022].

[11] G. Aleksandrowicz, T. Alexander, P. Barkoutsos, L. Bello, Y. Ben-Haim, D. Bucher, F. J. Cabrera-Hernández, J. Carballo-Franquis, A. Chen, C.-F. Chen, J. M. Chow, A. D. Córcoles-Gonzales, A. J. Cross, A. Cross, J. Cruz-Benito, C. Culver, S. D. L. P. González, E. D. L. Torre, D. Ding, E. Dumitrescu, I. Duran, P. Eendebak, M. Everitt, I. F. Sertage, A. Frisch, A. Fuhrer, J. Gambetta, B. G. Gago, J. Gomez-Mosquera, D. Greenberg, I. Hamamura, V. Havlicek, J. Hellmers, Łukasz Herok, H. Horii, S. Hu, T. Imamichi, T. Itoko, A. Javadi-Abhari, N. Kanazawa, A. Karazeev, K. Krsulich, P. Liu, Y. Luh, Y. Maeng, M. Marques, F. J. Martín-Fernández, D. T. McClure, D. McKay, S. Meesala, A. Mezzacapo, N. Moll, D. M. Rodríguez, G. Nannicini, P. Nation, P. Ollitrault, L. J. O'Riordan, H. Paik, J. Pérez, A. Phan, M. Pistoia, V. Prutyanov, M. Reuter, J. Rice, A. R. Davila, R. H. P. Rudy, M. Ryu, N. Sathaye, C. Schnabel, E. Schoute, K. Setia, Y. Shi, A. Silva, Y. Siraichi, S. Sivarajah, J. A. Smolin, M. Soeken, H. Takahashi, I. Tavernelli, C. Taylor, P. Taylour, K. Trabing, M. Treinish, W. Turner, D. Vogt-Lee, C. Vuillot, J. A. Wildstrom, J. Wilson, E. Winston, C. Wood, S. Wood, S. Wörner, I. Y. Akhalwaya, C. Zoufal, Qiskit: An Open-source Framework for Quantum Computing, 2019. URL: https://doi.org/10.5281/zenodo.2562111. doi:10.5281/zenodo.2562111.

[12] A. Cervera-Lierta, Quantum computing languages landscape, https://medium.com/@quantum_wa/quantum-computing-languages-landscape-1bc6dedb2a35, 2018. [Online; accessed December-2021].

[13] B. Kitchenham, S. Charters, Guidelines for Performing Systematic Literature Reviews in Software Engineering, 2007.

[14] K. Petersen, S. Vakkalanka, L. Kuzniarz, Guidelines for conducting systematic mapping studies in software engineering: An update, Information and Software Technology 64 (2015) 1–18. URL: https://www.sciencedirect.com/science/article/pii/S0950584915000646. doi:https://doi.org/10.1016/j.infsof.2015.03.007.

[15] P. S. Leitao-Junior, D. M. Freitas, S. R. Vergilio, C. G. Camilo-Junior, R. Harrison, Search-based fault localisation: A systematic mapping study, Information and Software Technology 123 (2020) 106295. URL: https://www.sciencedirect.com/science/article/pii/S0950584920300458. doi:https://doi.org/10.1016/j.infsof.2020.106295.

[16] A. Zakari, S. P. Lee, R. Abreu, B. H. Ahmed, R. A. Rasheed, Multiple fault localization of software programs: A systematic literature review, Information and Software Technology 124 (2020) 106312. URL: https://www.sciencedirect.com/science/article/pii/S0950584920300641. doi:https://doi.org/10.1016/j.infsof.2020.106312.

[17] A. Zakari, S. P. Lee, K. A. Alam, R. Ahmad, Software fault localisation: a systematic mapping study, IET Software 13 (2019) 60–74. URL: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-sen.2018.5137. doi:https://doi.org/10.1049/iet-sen.2018.5137. arXiv:https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-sen.2018.5137.

[18] S. Ouhbi, A. Idri, J. L. Fernández-Alemán, A. Toval, Requirements engineering education: a systematic mapping study, Requirements Engineering 20 (2015) 119–138.

[19] K. A. Alam, R. Ahmad, A. Akhunzada, M. H. N. M. Nasir, S. U. Khan, Impact analysis and change propagation in service-oriented enterprises: A systematic review, Information Systems 54 (2015) 43–73. URL: https://www.sciencedirect.com/science/article/pii/S0306437915001179. doi:https://doi.org/10.1016/j.is.2015.06.003.

[20] P. Benioff, The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines, Journal of Statistical Physics 22 (1980) 563–591. URL: https://doi.org/10.1007/BF01011339. doi:10.1007/BF01011339.

[21] E. C. R. Da Rosa, R. De Santiago, Ket quantum programming, J. Emerg. Technol. Comput. Syst. 18 (2021). URL: https://doi.org/10.1145/3474224. doi:10.1145/3474224.

[22] E. C. R. da Rosa, Ket quantum programming git, https://github.com/quantum-ket/ket, 2021. [Online; accessed March-2022].

[23] R. d. S. Evandro Chagas Ribeiro da Rosa, Ket quantum programming, https://quantumket.org/, 2021. [Online; accessed March-2022].

[24] Riverlane, Qhal - quantum hardware abstraction layer, https://github.com/riverlane/QHAL, 2021. [Online; accessed December-2021].

[25] B. Bichsel, M. Baader, T. Gehr, M. 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, Association for

Computing Machinery, New York, NY, USA, 2020, p. 286–300. URL: https://doi.org/10.1145/3385412.3386007. doi:10.1145/3385412.3386007.
[26] B. Bichsel, M. Baader, T. Gehr, M. Vechev, Silq, https://github.com/eth-sri/silq, 2016. [Online; accessed December-2021].
[27] Amazon, Amazon braket documentation, https://docs.aws.amazon.com/braket/?id=docs_gateway, 2020. [Online; accessed December-2021].
[28] Amazon, Amazon braket github, https://github.com/aws/amazon-braket-sdk-python, 2020. [Online; accessed December-2021].
[29] N. Killoran, J. Izaac, N. Quesada, V. Bergholm, M. Amy, C. Weedbrook, Strawberry fields: A software platform for photonic quantum computing, Quantum 3 (2019) 129. URL: https://doi.org/10.22331/q-2019-03-11-129. doi:10.22331/q-2019-03-11-129.
[30] N. Khammassi, G. G. Guerreschi, I. Ashraf, J. W. Hogaboam, C. G. Almudever, K. Bertels, cQASM v1.0: Towards a Common Quantum Assembly Language, arXiv preprint arXiv:1805.09607 (2018). URL: https://arxiv.org/abs/1805.09607.
[31] Q. Inspire, cqasm: A quantum programming language, https://www.quantum-inspire.com/kbase/cqasm/, 2018. [Online; accessed March-2022].
[32] D-Wave, D-wave ocean software documentation, https://docs.ocean.dwavesys.com/en/stable/, 2018. [Online; accessed December-2021].
[33] D-Wave, Ocean is d-wave's suite of tools for solving hard problems with quantum computers., https://github.com/dwavesystems/dwave-ocean-sdk, 2018. [Online; accessed December-2021].
[34] Google, Cirq, https://github.com/quantumlib/Cirq, 2019. [Online; accessed March-2022].
[35] D. Bacon, Quantum super entangled language (qsel), https://github.com/dabacon/qsel, 2018. [Online; accessed March-2022].
[36] A. Z. C. Product, Orquestra, https://www.orquestra.io/, 2020. [Online; accessed December-2021].
[37] Rigetti, Projects developed using forest, https://github.com/rigetti/forest-software, 2018. [Online; accessed December-2021].
[38] R. S. Smith, M. J. Curtis, W. J. Zeng, A practical quantum instruction set architecture, arXiv preprint arXiv:1608.03355 (2016). URL: https://arxiv.org/abs/1608.03355.
[39] Microsoft, Quantum development kit, https://github.com/microsoft/Quantum, 2017. [Online; accessed November-2021].
[40] K. Svore, A. Geller, M. Troyer, J. Azariah, C. Granade, B. Heim, V. Kliuchnikov, M. Mykhailova, A. Paz, M. Roetteler, Q#: Enabling Scalable Quantum Computing and Development with a High-Level DSL, in: Proceedings of the Real World Domain Specific Languages Workshop 2018, RWDSL2018, Association for Computing Machinery, New York, NY, USA, 2018, pp. 1–10. URL: https://doi.org/10.1145/3183895.3183901. doi:10.1145/3183895.3183901.
[41] Microsoft, Azure quantum documentation, https://docs.microsoft.com/pt-br/azure/quantum/, 2021. [Online; accessed 26-October-2021].
[42] IBM, Qiskit: An open-source sdk for working with quantum computers at the level of pulses, circuits, and algorithms, https://github.com/Qiskit, 2017. [Online; accessed November-2021].
[43] S. Liu, X. Wang, L. Zhou, J. Guan, Y. Li, Y. He, R. Duan, M. Ying, $q|si\rangle$: A quantum programming environment, 2017. URL: https://arxiv.org/abs/1710.09500. arXiv:1710.09500.
[44] S. Pakin, A quantum macro assembler, in: 2016 IEEE High Performance Extreme Computing Conference (HPEC), 2016, pp. 1–8. doi:10.1109/HPEC.2016.7761637.
[45] S. Pakin, Qmasm: A quantum macro assembler, https://github.com/lanl/qmasm, 2017. [Online; accessed December-2021].
[46] D. S. Steiger, T. Häner, M. Troyer, Projectq: an open source software framework for quantum computing, Quantum 2 (2018) 49. URL: https://doi.org/10.22331/q-2018-01-31-49. doi:10.22331/q-2018-01-31-49.
[47] D. Wecker, K. M. Svore, $LIQUi|\rangle$: A Software Design Architecture and Domain-Specific Language for Quantum Computing, arXiv preprint arXiv:1402.4467 (2014). URL: https://arxiv.org/abs/1402.4467.
[48] Microsoft, The language-integrated quantum operations ($liqui|\rangle$) simulator, https://github.com/StationQ/Liquid, 2016. [Online; accessed March-2022].
[49] Microsoft, $liqui|\rangle$ the language integrated quantum operations simulator, http://stationq.github.io/Liquid/, 2016. [Online; accessed March-2022].
[50] Microsoft, Language-integrated quantum operations: $liqui|\rangle$, https://www.microsoft.com/en-us/research/project/language-integrated-quantum-operations-liqui/, 2016. [Online; accessed March-2022].
[51] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, B. Valiron, Quipper: A scalable quantum programming language, SIGPLAN Not. 48 (2013) 333–342. URL: https://doi.org/10.1145/2499370.2462177. doi:10.1145/2499370.2462177.
[52] A. Lapets, M. P. da Silva, M. Thome, A. Adler, J. Beal, M. Roetteler, QuaFL: A Typed DSL for Quantum Programming, in: Proceedings of the 1st Annual Workshop on Functional Programming Concepts in Domain-Specific Languages, FPCDSL '13, Association for Computing Machinery, New York, NY, USA, 2013, p. 19–26. URL: https://doi.org/10.1145/2505351.2505357. doi:10.1145/2505351.2505357.
[53] A. J. Abhari, A. Faruque, M. J. Dousti, L. Svec, O. Catu, A. Chakrabati, C.-F. Chiang, S. Vanderwilt, J. Black, F. Chong, M. Martonosi, M. Suchara, K. Brown, M. Pedram, T. Brun, Scaffold: Quantum Programming Language, Department of Computer Science, Princeton University, Tech. Rep. TR-934-12 (2012). URL: https://apps.dtic.mil/sti/citations/ADA571279.
[54] M. Purkeypile, Cove: A practical quantum computer programming framework, arXiv preprint arXiv:0911.2423 (2009). URL: https://arxiv.org/abs/0911.2423.
[55] J. Xu, F. Song, Quantum programming languages: A tentative study, Science in China Series F: Information Sciences 51 (2008) 623–637. URL: https://doi.org/10.1007/s11432-008-0059-4. doi:10.1007/s11432-008-0059-4.
[56] H. Mlnarik, Operational Semantics and Type Soundness of Quantum Programming Language LanQ, arXiv preprint arXiv:0708.0890 (2007). URL: https://arxiv.org/abs/0708.0890.
[57] J. K. Vizzotto, A. C. da Rocha Costa, Towards Quantum Haskell via Quantum Arrows, in: Workshop-Escola de Computação e Informação Quântica, volume 52, 2006, pp. 1–10.
[58] J. Xu, F. Song, Quantum programming languages, Frontiers of Computer Science in China 2 (2008) 161–166. URL: https://doi.org/10.1007/s11704-008-0013-z. doi:10.1007/s11704-008-0013-z.

[59] W. Mauerer, Semantics and simulation of communication in quantum programming, arXiv e-prints (2005) quant–ph/0511145. URL: https://arxiv.org/abs/quant-ph/0511145. arXiv:quant-ph/0511145.
[60] T. Altenkirch, J. Grattage, A functional quantum programming language, in: 20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05), 2005, pp. 249–258. URL: https://doi.org/10.1109/LICS.2005.1. doi:10.1109/LICS.2005.1.
[61] S. J. Gay, R. Nagarajan, Communicating quantum processes, in: Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming languages, 2005, pp. 145–157.
[62] P. Jorrand, M. Lalire, From quantum physics to programming languages: A process algebraic approach, in: J.-P. Banâtre, P. Fradet, J.-L. Giavitto, O. Michel (Eds.), Unconventional Programming Paradigms, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 1–16. URL: https://doi.org/10.1007/11527800_1. doi:10.1007/11527800_1.
[63] P. Selinger, Towards a Quantum Programming Language, Mathematical Structures in Computer Science 14 (2004) 527–586. URL: https://doi.org/10.1017/S0960129504004256. doi:10.1017/S0960129504004256.
[64] S. Bettelli, T. Calarco, L. Serafini, Toward an architecture for quantum programming, The European Physical Journal D - Atomic, Molecular and Optical Physics 25 (2003) 181–200. URL: http://dx.doi.org/10.1140/epjd/e2003-00242-2. doi:10.1140/epjd/e2003-00242-2.
[65] A. Sabry, Modeling quantum computing in haskell, in: Proceedings of the 2003 ACM SIGPLAN Workshop on Haskell, Haskell '03, Association for Computing Machinery, New York, NY, USA, 2003, p. 39–49. URL: https://doi.org/10.1145/871895.871900. doi:10.1145/871895.871900.
[66] J. W. Sanders, P. Zuliani, Quantum programming, in: R. Backhouse, J. N. Oliveira (Eds.), Mathematics of Program Construction, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 80–99. URL: https://doi.org/10.1007/10722010_6. doi:10.1007/10722010_6.
[67] B. Ömer, Procedural Quantum Programming, AIP Conference Proceedings 627 (2002) 276–285. URL: https://doi.org/10.1063/1.1503695. doi:10.1063/1.1503695. arXiv:https://pubs.aip.org/aip/acp/article-pdf/627/1/276/11571870/276_1_online.pdf.
[68] B. Ömer, Qcl – a programming language for quantum computers, http://tph.tuwien.ac.at/~oemer/qcl.html, 2014. [Online; accessed October-2021].
[69] B. Oemer, Qcl (quantum computing language), https://github.com/aviggiano/qcl, 2018. [Online; accessed November-2021].
[70] P. Maymin, The lambda-q calculus can efficiently simulate quantum computers, arXiv e-prints (1997) quant–ph/9702057. URL: https://arxiv.org/abs/quant-ph/9702057. arXiv:quant-ph/9702057.
[71] D. 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.
[72] S. Bettelli, L. Serafini, T. Calarco, Toward an architecture for quantum programming, The European Physical Journal D 25 (2001). doi:10.1140/epjd/e2003-00242-2.
[73] J. W. Sanders, P. Zuliani, Quantum programming, in: R. Backhouse, J. N. Oliveira (Eds.), Mathematics of Program Construction, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 80–99.
[74] A. Van Tonder, A lambda calculus for quantum computation, SIAM Journal on Computing 33 (2004) 1109–1135.
[75] P. Selinger, B. Valiron, A lambda calculus for quantum computation with classical control, in: P. Urzyczyn (Ed.), Typed Lambda Calculi and Applications, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 354–368.
[76] W. Mauerer, Semantics and simulation of communication in quantum programming, arXiv preprint quant-ph/0511145 (2005).
[77] V. R. Basili, G. Caldiera, H. D. Rombach, The goal question metric approach, Encyclopedia of software engineering (1994) 528–532.
[78] U. Kuter, C. Yilmaz, Survey methods: Questionnaires and Interviews, Choosing Human-Computer Interaction (HCI) Appropriate Research Methods (2001) 1–9.
[79] S. Dalati, J. Gómez, Surveys and Questionnaires, 2018, pp. 175–186. doi:10.1007/978-3-319-74173-4_10.
[80] P. Regmi, E. Waithaka, A. Paudyal, P. Simkhada, E. Van Teijlingen, Guide to the design and application of online questionnaire surveys, Nepal Journal of Epidemiology 6 (2017) 640. doi:10.3126/nje.v6i4.17258.
[81] SoGoSurvey, Sogosurvey homepage, https://www.sogosurvey.com, 2021. [Online; accessed 12-October-2021].
[82] Google, Google forms homepage, https://www.google.com/forms/about/, 2021. [Online; accessed 12-October-2021].
[83] Survio, Survio homepage, https://www.survio.com, 2021. [Online; accessed 12-October-2021].
[84] MindMiners, Mindminers homepage, https://mindminers.com/, 2021. [Online; accessed 12-October-2021].
[85] TypeForm, Typeform homepage, https://www.typeform.com/, 2021. [Online; accessed 12-October-2021].
[86] SurveyMonkey, Surveymonkey homepage, https://https://www.surveymonkey.com/, 2021. [Online; accessed 12-October-2021].
[87] R. A. Fisher, On the Interpretation of χ^2 from Contingency Tables, and the Calculation of P, Journal of the Royal Statistical Society 85 (1922) 87–94. URL: http://www.jstor.org/stable/2340521.
[88] A. Lex, N. Gehlenborg, H. Strobelt, R. Vuillemot, H. Pfister, UpSet: Visualization of Intersecting Sets, IEEE Transactions on Visualization and Computer Graphics 20 (2014) 1983–1992. doi:10.1109/TVCG.2014.2346248.
[89] J. R. Conway, A. Lex, N. Gehlenborg, UpSetR: an R package for the visualization of intersecting sets and their properties, Bioinformatics 33 (2017) 2938–2940. URL: https://doi.org/10.1093/bioinformatics/btx364. doi:10.1093/bioinformatics/btx364.
[90] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, S. Ahmed, V. Ajith, M. S. Alam, G. Alonso-Linaje, B. AkashNarayanan, A. Asadi, J. M. Arrazola, U. Azad, S. Banning, C. Blank, T. R. Bromley, B. A. Cordier, J. Ceroni, A. Delgado, O. D. Matteo, A. Dusko, T. Garg, D. Guala, A. Hayes, R. Hill, A. Ijaz, T. Isacsson, D. Ittah, S. Jahangiri, P. Jain, E. Jiang, A. Khandelwal, K. Kottmann, R. A. Lang, C. Lee, T. Loke, A. Lowe, K. McKiernan, J. J. Meyer, J. A. Montañez-Barrera, R. Moyard, Z. Niu, L. J. O'Riordan, S. Oud, A. Panigrahi, C.-Y. Park, D. Polatajko, N. Quesada, C. Roberts, N. Sá, I. Schoch, B. Shi, S. Shu, S. Sim, A. Singh, I. Strandberg, J. Soni, A. Száva, S. Thabet, R. A. Vargas-Hernández, T. Vincent, N. Vitucci, M. Weber, D. Wierichs, R. Wiersema, M. Willmann, V. Wong, S. Zhang, N. Killoran, PennyLane: Automatic differentiation of hybrid quantum-classical computations, 2022. URL: https://arxiv.org/abs/1811.04968. arXiv:1811.04968.
[91] J. Johansson, P. Nation, F. Nori, QuTiP: An open-source Python framework for the dynamics of open quantum systems, Computer Physics Communications 183 (2012) 1760–1772. URL: https://www.sciencedirect.com/science/article/pii/S0010465512000835. doi:https://doi.org/10.1016/j.cpc.2012.02.021.

[92] J. Johansson, P. Nation, F. Nori, QuTiP 2: A Python framework for the dynamics of open quantum systems, Computer Physics Communications 184 (2013) 1234–1240. URL: https://www.sciencedirect.com/science/article/pii/S0010465512003955. doi:https://doi.org/10.1016/j.cpc.2012.11.019.
[93] Quarkus, Funqy, https://quarkus.io/guides/funqy, 2021. [Online; accessed December-2021].
[94] K. Hietala, R. Rand, S.-H. Hung, X. Wu, M. Hicks, A Verified Optimizer for Quantum Circuits, Proceedings of the ACM on Programming Languages 5 (2021). URL: https://doi.org/10.1145/3434318. doi:10.1145/3434318.
[95] X. Fu, J. Yu, X. Su, H. Jiang, H. Wu, F. Cheng, X. Deng, J. Zhang, L. Jin, Y. Yang, L. Xu, C. Hu, A. Huang, G. Huang, X. Qiang, M. Deng, P. Xu, W. Xu, W. Liu, Y. Zhang, Y. Deng, J. Wu, Y. Feng, Quingo: A Programming Framework for Heterogeneous Quantum-Classical Computing with NISQ Features, ACM Transactions on Quantum Computing 2 (2021). URL: https://doi.org/10.1145/3483528. doi:10.1145/3483528.
[96] N. Heurtel, A. Fyrillas, G. d. Gliniasty, R. Le Bihan, S. Malherbe, M. Pailhas, E. Bertasi, B. Bourdoncle, P.-E. Emeriau, R. Mezher, L. Music, N. Belabas, B. Valiron, P. Senellart, S. Mansfield, J. Senellart, Perceval: A Software Platform for Discrete Variable Photonic Quantum Computing, Quantum 7 (2023) 931. URL: https://doi.org/10.22331/q-2023-02-21-931. doi:10.22331/q-2023-02-21-931.
[97] M. 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.
[98] X. Wang, P. Arcaini, T. Yue, S. Ali, Quito: a Coverage-Guided Test Generator for Quantum Programs, in: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2021, pp. 1237–1241. doi:10.1109/ASE51524.2021.9678798.
[99] J. Wang, F. Ma, Y. Jiang, Poster: Fuzz Testing of Quantum Program, in: 2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST), 2021, pp. 466–469. doi:10.1109/ICST49551.2021.00061.
[100] E. Mendiluze, S. Ali, P. Arcaini, T. Yue, Muskit: A Mutation Analysis Tool for Quantum Software Testing, in: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2021, pp. 1266–1270. doi:10.1109/ASE51524.2021.9678563.
[101] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, A. Wessln, Experimentation in Software Engineering, Springer Publishing Company, Incorporated, 2012.
[102] 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/.
[103] S. Overflow, Stack overflow annual developer survey, https://insights.stackoverflow.com/survey, 2021. [Online; accessed November-2021].
[104] JetBrains, The state of developer ecosystem 2021, https://www.jetbrains.com/lp/devecosystem-2021/, 2021. [Online; accessed November-2021].
[105] Increment.com, Six questions on programming languages, https://increment.com/programming-languages/six-questions-on-programming-languages/, 2018. [Online; accessed November-2021].
[106] ComputerScience.org, Computerscience.org website, https://www.computerscience.org/, 2021. [Online; accessed November-2021].
[107] C. Hope, Why are there so many programming language, https://www.computerhope.com/issues/ch000569.htm, 2020. [Online; accessed March-2022].
[108] Q. Chen, R. Câmara, J. Campos, A. Souto, I. Ahmed, The Smelly Eight: An Empirical Study on the Prevalence of Code Smells in Quantum Computing, in: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), 2023, pp. 358–370. URL: https://doi.org/10.1109/ICSE48619.2023.00041. doi:10.1109/ICSE48619.2023.00041.
[109] D. Fortunato, J. Campos, R. Abreu, Mutation Testing of Quantum Programs Written in QISKit, in: Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings, ICSE '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 358–359. URL: https://doi.org/10.1145/3510454.3528649. doi:10.1145/3510454.3528649.
[110] D. Fortunato, J. Campos, R. Abreu, QMutPy: A Mutation Testing Tool for Quantum Algorithms and Applications in Qiskit, in: Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2022, Association for Computing Machinery, New York, NY, USA, 2022, p. 797–800. URL: https://doi.org/10.1145/3533767.3543296. doi:10.1145/3533767.3543296.
[111] D. Fortunato, J. Campos, R. Abreu, Mutation Testing of Quantum Programs: A Case Study With Qiskit, IEEE Transactions on Quantum Engineering 3 (2022) 1–17. URL: https://doi.org/10.1109/TQE.2022.3195061. doi:10.1109/TQE.2022.3195061.
[112] P. 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.
[113] D. Unruh, Quantum programming languages, Inform., Forsch. Entwickl. 21 (2006) 55–63. doi:10.1007/s00450-006-0012-y.
[114] D. Rojas, The modern state of quantum programming language (2019).
[115] M. De Stefano, F. Pecorelli, D. Di Nucci, F. Palomba, A. De Lucia, Software engineering for quantum programming: How far are we?, arXiv preprint arXiv:2203.16969 (2022).
[116] HOPL, Online historical encyclopaedia of programming languages, https://hopl.info/, 2020. [Online; accessed March-2022].
[117] GitHub, Developer feedback helps steer github public policy commitments, https://octoverse.github.com/#developer-feedback-helps-steer-git-hub-public-policy-commitments, 2021. [Online; accessed March-2022].
[118] V. Lagutin, Why are there so many programming language, https://www.freecodecamp.org/news/why-are-there-so-many-programming-languages/, 2021. [Online; accessed March-2022].
[119] M. Sherman, Why are there so many programming language, https://stackoverflow.blog/2015/07/29/why-are-there-so-many-programming-languages/, 2015. [Online; accessed March-2022].
[120] M. L. Scott, Programming language pragmatics, http://www.cs.yorku.ca/~billk/cse3301_S06/lectures/cse3301_S06_july17_6slides.pdf, 2017. [Online; accessed March-2022].
[121] E. Knill, Conventions for quantum pseudocode, 2022. arXiv:2211.02559.
[122] J.-Y. Girard, Between logic and quantic: a tract, Technical Report, MATHEMATICAL STRUCTURES IN COMPUTER SCIENCE, 2003.
[123] S. Abramsky, B. Coecke, Physical traces: Quantum vs. classical information processing, CoRR cs.CG/0207057 (2002). URL: https://arxiv.org/abs/cs/0207057.

[124] A. Edalat, An extension of gleason's theorem for quantum computation, International Journal of Theoretical Physics 43 (2004) 1827–1840. URL: http://dx.doi.org/10.1023/B:IJTP.0000048823.93080.7e. doi:10.1023/b:ijtp.0000048823.93080.7e.
[125] B. Coecke, K. Martin, A Partial Order on Classical and Quantum States, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 593–683. URL: https://doi.org/10.1007/978-3-642-12821-9_10. doi:10.1007/978-3-642-12821-9_10.
[126] S. J. Gay, Quantum programming languages: Survey and bibliography, Mathematical Structures in Computer Science 16 (2006) 581–600.
[127] D. Deutsch, Quantum theory, the church–turing principle and the universal quantum computer, Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences 400 (1985) 97–117.
[128] P. Maymin, The lambda-q calculus can efficiently simulate quantum computers, arXiv preprint quant-ph/9702057 (1997).
[129] P. Jorrand, M. Lalire, Toward a quantum process algebra, in: Proceedings of the 1st Conference on Computing Frontiers, 2004, pp. 111–119.
[130] J.-Y. Girard, Linear logic, Theoretical computer science 50 (1987) 1–101.
[131] S. Garhwal, M. Ghorani, A. Ahmad, Quantum programming language: A systematic review of research topic and top cited languages, Archives of Computational Methods in Engineering 28 (2019) 289–310.
[132] M. A. Serrano, J. A. Cruz-Lemus, R. Perez-Castillo, M. Piattini, Quantum software components and platforms: Overview and quality assessment, ACM Comput. Surv. 55 (2022). URL: https://doi.org/10.1145/3548679. doi:10.1145/3548679.