Quantum Information and Computation, Vol. 24, No. 7&8 (2024) 0609–0642
© Rinton Press

QUANTUM SOFTWARE DEVELOPMENT: A SURVEY

LUIS JIMNEZ-NAVAJAS
Faculty of Social Sciences & IT, University of Castilla-La Mancha
Talavera de la Reina, Spain

FABIAN BHLER
Institute of Architecture of Application Systems, University of Stuttgart
Stuttgart, Germany

FRANK LEYMANN
Institute of Architecture of Application Systems, University of Stuttgart
Stuttgart, Germany

RICARDO PREZ-CASTILLO[a]
Faculty of Social Sciences & IT, University of Castilla-La Mancha
Talavera de la Reina, Spain

MARIO PIATTINI
Escuela Superior de Informtica, University of Castilla-La Mancha
Ciudad Real, Spain

DANIEL VIETZ
Institute of Architecture of Application Systems, University of Stuttgart
Stuttgart, Germany

Received February 6, 2024
Revised July 15, 2024

[a]Corresponding author

Over the last few years, quantum computing has been growing at an exponential pace. Every day, new techniques, frameworks, modeling, and programming languages are emerging that aim to facilitate the development of quantum software, which is key to achieving the promising applications of quantum computing. However, which of these are actively used and the degree of satisfaction of researchers and developers regarding these quantum software frameworks and languages is not known. To address this, we conducted a survey to characterize which modeling tools and which quantum programming languages are used during the quantum software lifecycle. Researchers in academia and industry developers were surveyed, and a total of 57 responses were collected. The results indicate that during quantum software development, some models and diagrams are used to guide development. In addition, the survey results show what quantum programming languages are the most used alongside the classical programming languages employed to build hybrid programs, among other important insights. The implications of this survey are: (i) to find out what the current trends are within quantum software development and (ii) to find out what the needs are of quantum software developers with respect to current modeling and programming languages and tools.

Keywords: Quantum Computing, Survey, Hybrid Information Systems, Quantum Software Development

1 Introduction

Quantum computing is expected to make an impact on systems (and thus society) when its associated technology is sufficiently mature [1]. Technology giants such as Amazon, IBM, Google, and Microsoft are investing a lot of effort in evolving this new paradigm, whether by creating new languages, researching the creation of better quantum computers, or enabling quantum services. This growing interest in quantum computing and its potential impact in various fields has led to the emergence of a vibrant quantum ecosystem consisting of start-ups, research laboratories, and consortiums.

To support quantum software development, a growing number of frameworks and domain-specific languages can be found. For example, there are programming languages, software development kits, modeling languages, diagrams, and orchestration tools, among many others. Both industry and academia are working (with promising results) to extend the technological landscape available for quantum software development. This includes adaptations of modeling languages [2, 3, 4], the definition of lifecycle development processes [5], and orchestration of quantum software [6].

Although there are already several systematic studies and surveys that depict different aspects of the actual state of the art of quantum software development, such as the quantum programming languages used [7, 8] or testing approaches followed [9], none of them investigates the utilization of modeling languages during the quantum software development lifecycle. Further, it is unclear whether developers are satisfied with them. Also, there are no studies describing the current state of quantum software development distinguishing between the implementation and operation phases. This implies that it is not known whether the proposed tools are useful for quantum software developers in any phase. Furthermore, information about which modeling techniques are used the most and the least, as well as the satisfaction with them, is crucial for researchers in quantum software engineering to create new techniques or improve existing ones. Knowing what trends exist in quantum software implementation would enable a better understanding of the context and situation of this new paradigm.

To address this, we present a survey which gives new insights about modeling, implementation, and operation throughout the quantum software modeling lifecycle. The survey aims to: (i) characterize the quantum software modeling process during the different phases of the quantum software engineering (QSE) lifecycle [10, 11], (ii) illustrate the utilization of tools such as programming languages, software development kits (SDKs), or frameworks during the implementation phase of quantum software, and (iii) depict the use of such tools (programming languages, SDKs, frameworks, etc.) during the operations phase. The study follows the Goal-Question-Metric (GQM) approach [12] in order to conduct the survey in a systematic way. The survey was sent to authors from academia and industry. A total of 57 responses were collected. The responses obtained show that research in this new paradigm is latent in both industry and academia. The results obtained draw several insights. Firstly, different software modeling techniques are used during the development of quantum software where flowcharts, informal sketches, and UML diagrams stand out. Secondly, for the implementation of quantum software, Qiskit and OpenQASM are the most used languages, accompanied by Python as a classical programming language to create hybrid software. Finally, for the operation of quantum software, the whole quantum software paradigm will need to mature.

The paper is structured as follows. Section 2 describes the background of this study. Section 3 explains how the survey was designed and carried out. Section 4 shows an analysis of the results obtained. Section 5 discusses the results obtained from this study and the implications associated with them. Finally, Section 6 gives a conclusion and an outlook on future work.

2 Background and Related Work

This section introduces the underlying concepts related to the conducted survey. Section 2.1 explains the foundations of quantum computing. Sections 2.2 and 2.3 provide a background on quantum software and quantum software engineering. Then, Section 2.4 presents some surveys related to the development of quantum software.

2.1 Quantum Computing

Originally, quantum computing was proposed as an alternative to classical computers to simulate nature in a more accurate way [13]. Nature at subatomic levels behaves probabilistically (as it follows the laws of quantum mechanics). However, classical computers work in a deterministic way, making the simulation of nature on these computers inviable. Quantum computing is a new paradigm of computing that exploits the principles of quantum mechanics to achieve better performance than classical computing in certain scenarios.

There are several ways of working with quantum computers, including gate-based, adiabatic, and topological quantum computing, among others, with the most frequently used being gate-based quantum computing [14]. Gate-based quantum computing operates similarly to the assembly language of traditional computers, where the instructions in the source code directly correspond to hardware-level instructions. However, in quantum computing, the quantum gates influence qubits by modifying their properties, such as applying superposition, entanglement, or altering their amplitude. Quantum circuits are commonly used to develop programs in this paradigm. In visual representations of quantum circuits, qubits are represented by horizontal lines and quantum gates are placed on these lines to change their state. The type of gate applied determines how the qubit's state will be changed, e.g., the Hadamard gate creates a superposition state.

Adiabatic quantum computing involves encoding the solution to a problem in the ground state of a quantum system and then evolving the system slowly enough for it to remain in its ground state. This approach leverages the principles of quantum mechanics to solve optimization problems by finding the lowest energy configuration of the system.

2.2 Quantum Software

Similar to classical computing, there are many different programming languages and software development kits available for quantum computing to choose from. For example, quantum programming languages such as Q# (Microsoft) [15] or OpenQASM (IBM) [16] among others. In addition, it is also possible to import libraries for programming quantum software like Qiskit (IBM) [17], Cirq (Google) [18], or Strawberry Fields (Xanadu) [19]. The quantum programs can then be simulated locally using a quantum computing simulator or executed via a provider that allows running quantum programs on real quantum computers or simulators through an application programming interface (API), e.g., QuantumPath (aQuantum) [20] or Braket (Amazon) [21].

2.3 Quantum Software Engineering

Due to the emerging need to produce quantum software in an industrial controlled manner, the Talavera Manifesto was proposed [25]. This manifesto advocates the need for the creation of QSE, which is indispensable for the proper development of quantum computing. QSE claims to adapt the existing software engineering processes, methods, techniques, practices, and principles for the development of quantum software (or it may imply creating new ones) [26].

Many proposals have been published related to the adaptation of existing software engineering processes to the field of quantum computing. Regarding adapted modeling languages, some examples are the Unified Modeling Language (UML) [2, 3], Business Process Model and Notation (BPMN) [27], and Knowledge Discovery Metamodel (KDM) [28]. Furthermore, there have been proposals for processes for quantum software engineering like the quantum information technology governance system [29] and even a quantum software development lifecycle [10, 11].

2.4 Surveys Related to Quantum Software

Several surveys related to quantum software can be found both in the literature and in industrial projects. From academia, surveys that inquire into various aspects of quantum software development can be found. For example, Li et al. [30] investigate the challenges of quantum programming from a developer's perspective, examining technical Q&A forums where developers ask QSE-related questions and GitHub issue reports where developers raise QSE-related issues. Khan et al. [31] present a survey on the suitability of agile practices in quantum software development, conducting interviews with various software practitioners. De Stefano et al. [32] present an extension of the work of Li et al. [30], including a mining software repository analysis and a survey study in order to assess how far quantum software engineering is from effectively supporting developers in practice. Another interesting survey is presented in [33], which provides a comprehensive analysis of the evolving needs within the quantum industry, highlighting the essential skills and educational requirements for future professionals, thus contributing valuable insights for academia and industry stakeholders. However, this survey does not specifically focus on quantum software development.

Industry and independent projects, surveys such as that from Zapata Computing [7] collected insights on how far enterprises are in their adoption of quantum computing and the challenges they face. Unitary Fundation [8] also published the results of their quantum open-source survey, aiming to obtain a community-wide and industry-wide snapshot that is representative of everyone who codes for and with quantum computing technologies. The quantum computing vendor D-Wave also published the results of a survey with responses from early quantum computing adopters in the United States and Europe [34]. Classiq, the vendor behind the Quantum Algorithm Design platform, also conducted a survey to highlight the interest and future investment in quantum computing in industry sectors [35].

Currently, there are no surveys that address the use of quantum software modeling languages and diagrams. Furthermore, the studies presented above also did not inquire about the use of different tools, programming languages, or frameworks depending on the stage of quantum software development.

3 Survey Design

This section describes the survey's design. The rationale and objectives of the survey are discussed in Section 3.1. Section 3.2 explains the Goal-Question-Metric (GQM) approach used to conduct the survey. Sections 3.3, 3.4, and 3.5 detail the instrumentation, experimental procedure, and data collection methods, respectively.

3.1 Rationale and Objectives

As stated before, currently there are no studies that address the use of modeling tools or languages through the process of quantum software development. Conducting a survey in this regard may help with the characterization of the use of these tools. This survey could also be helpful to help the scientific community to consider possible development processes and supporting tools. This study has two distinct objectives. Firstly, it aims to showcase the current trends in quantum software development. Secondly, it seeks to identify the specific modeling tools being used and assess possible shortcomings. Therefore, the survey contributes to the characterization of the utilization of these tools and understanding the shortcomings of the existing. In addition, it assists the scientific community in exploring potential development processes and improving supporting tools.

3.2 Goal-Question-Metric Approach

This survey has been conducted by following the Goal-Question-Metric (GQM) approach [12] which allows to develop the survey in a purposeful and systematic way. This approach consists of three levels. First, the conceptual level, where one or several goals are defined, each from various points of view, relative to a particular environment (cf. Section 3.2.1). Second, the operational level, in which a set of issues is described for characterizing the way the assessment/achievement of a specific goal is going to be performed based on some characterizing model (cf. Section 3.2.2). Third, the quantitative level, where a data set is defined for every question in order to answer it quantitatively. These metrics can answer several questions (cf. Section 3.2.3). As a result, it provides a framework for interpreting the collected data with respect to the stated goals. The main reason for embracing the GQM approach for the survey design is that GQM allows one to characterize the current state of development in a systematic and straightforward way. This study does not present any hypotheses about the state of quantum software development. This is because no scientific studies, at the moment, allow to make such hypotheses. However, thanks to this study, it may be possible to hypothesize about the progress of the quantum software development process.

3.2.1 Goals

On the conceptual level, three goals have been defined following the template of Basili et al. [12]. The GQM goal template can be used to articulate the purpose of any study [36]. As this research aims to understand the overall context of quantum software development from a software engineering perspective, three main goals have been set.

The first goal is to analyze the quantum software modeling process with respect to the modeling frameworks, standards, and tools used. Therefore, the purpose of this goal is to characterize the use of these elements. So, its definition following the GQM template is as follows:

Goal 1: Analyze quantum software modeling with the purpose of characterizing with respect to the employment of modeling frameworks/standards/tools from the point of view of quantum software engineers in the context of institutions and companies which develop quantum software.

The second goal consists of characterizing what SDKs, tools, compilers, and languages are used during the implementation of quantum software. With this goal, the purpose is to figure out what current trends exist (if any), such as the most used languages or what combination of classical and quantum languages is more widely used, among others. So, its definition following the GQM template is as follows:

Goal 2: Analyze quantum software implementation with the purpose of characterizing with respect to used SDK, Tools, Compilers, languages from the point of view of quantum software engineers in the context of institutions and companies which develop quantum software.

Finally, the third goal concerns the analysis of quantum software operation to characterize which SDKs, tools, compilers, languages are used during the phase of quantum software operation. This distinguishes operations such as execution, integration, orchestration from the coding tasks (second goal). This achieves greater precision when studying the kind of tools that are used during quantum software development. So, its definition following the GQM template is as follows:

Goal 3: Analyze quantum software operation with the purpose of characterizing with respect to used SDK, Tools, Compilers, languages from the point of view of quantum software engineers in the context of institutions and companies which develop quantum software.

All these goals have been formulated from the point of view of quantum software engineers in the context of institutions and companies that develop quantum software.

3.2.2 Questions

Following the GQM approach, for each of the defined goals (see Section 3.2.1) several questions have been set. Table 1 shows the questions defined for each goal. Each question is associated with an identifier that helps to trace it regarding goals and metrics. Table 1 presents the research questions defined for each goal. Each research question (RQ) has an associated identifier to facilitate traceability.

Research questions defined for Goal 1 helps to find out if some modeling approaches exist (and how they are used) during any phase of quantum software development. In addition, it could be possible that existing modeling approaches do not cover all the needs of quantum software engineers, so informal sketches may also be used to cover such modeling needs.

The research questions associated with Goal 2 (see Table 1) attempt to outline how the implementation of quantum software is carried out. Questions have been raised about what languages and tools are used to develop quantum software and whether classical languages exist to support quantum software programming. Furthermore, the testing of quantum software is also evaluated in this goal.

Finally, research questions derived from Goal 3 (see Table 1) are those related to the operation of quantum software. To characterize the operation of quantum software, those questions concern the execution of quantum software and it is the respective orchestration (if any).

Table 1. Questions established for each goal.

Goal	**ID**	**Question**
Goal 1	RQ 1.1	How is modeling used in any quantum software lifecycle phase?
	RQ 1.2	How are informal sketches designed?
	RQ 1.3	How do current modeling approaches satisfy the needs of quantum software engineers?
Goal 2	RQ 2.1	Why is quantum computing software implemented nowadays and from who?
	RQ 2.2	How is quantum software developed?
	RQ 2.3	How is hybrid software composed?
	RQ 2.4	What tools are used to implement quantum software?
Goal 3	RQ 3.1	How are quantum programs executed?
	RQ 3.2	How are workflow technologies used to orchestrate QC applications?
	RQ 3.3	How are language converters used to develop and execute quantum software?

Some of the questions involved the use of certain tools during the quantum software development lifecycle. The definition followed is from Weder et al. [5], and the process is defined within eight phases:

- **Requirement analysis.** The different interested stakeholders identify their requirements (both functional and non-functional).

- **Architecture and design.** The architecture is conceptualized by using both classical and quantum parts (previously split) and specifying corresponding software components with their functionality and interfaces. Then, the architecture is refined with the internals of the different software components.
- **Implementation.** The quantum application is implemented based on the requirements and design from the previous phases. Thereby, the implementation includes the development of the different constituting software artifacts.
- **Testing.** After the implementation, the quantum application is tested to verify the intended behavior according to the specified requirements before delivering it to the users.
- **Deployment.** During the deployment phase, everything is prepared to enable the execution of the quantum application. Thus, the execution environment for the classical programs is set up. Similarly, also the quantum programs and the workflows must be deployed.
- **Observability.** The quantum application, as well as its execution environment, are monitored. Thereby, data is collected for observing the current state of a running quantum application and storing the data in the long term to enable its analysis.
- **Analysis.** In the last phase of the lifecycle, the collected data from the observability phase is analyzed. The goals of this phase are to find bugs that must be fixed or possible improvements for the quantum application.

3.2.3 Metrics

Following the GQM process, for each defined question a set of metrics has been established that will help to answer the defined questions to consequently achieve the goals. These metrics will then be adapted to the questions included in the questionnaire.

The defined metrics can be seen in the Appendix in Tables A.1, A.2, and A.3 for the respective goals. Possible answers that the participants have are indicated within brackets next to each metric, with multiple options separated by semicolons. Most of the questions are close ended, although the possibility has been left for the user to write his/her own answer for those questions with the option *"other"*. Those metrics with two asterisks have multi-choice answers, i.e., participants are able to choose several options to answer the question. All questions have the option of not being answered, as the option *"I don't know/I'd rather not answer"* is provided.

3.3 Instrumentation

To carry out this study, three instruments have been used: (a) a questionnaire to collect information; (b) a survey web application; (c) scientific, digital libraries to get potential respondents from the academia; and (d) a Python script for gathering potential respondents from the industry.

Regarding the questionnaire, it was made up of a total of 42 questions. Although the total number of metrics is 37, 8 additional questions were added to adapt the metrics to a questionnaire-type format and to allow respondents to send us feedback.

A web-based survey was created and implemented by using the platform SurveySparrow [37]. This platform was chosen because: (i) it has integration with other data analysis tools; and (ii) it has a user interface that makes the definition of the questionnaire easy and allows defining conditional branching for interrelated questions. Formal requests for participations were carried out by email, including the link to the survey questionnaire. On 13th December 2022, the survey was sent to the researchers and quantum software developers of the aQuantum team [38] as well as to the network of researchers who endorsed the Talavera's Manifesto [25]. The next day, the survey was sent to users who had published research papers related to quantum software at quantum computing conferences or conferences that are co-located with quantum workshops (including several editions). These conferences were International Conference on Software Engineering (ICSE) [39], IEEE International Conference on Quantum Computing (QCE) [40] and the International Conference on the Quality of Information and Communications Technology (QUATIC) [41]. To obtain their contact details, the contact email address provided by the authors in their research papers was collected. A total of 139 authors were obtained, of which 36 replied. Finally, on January 3rd, the script for gathering users mentioned previously was employed. A total of 408 contacts were collected, of which 20 more users responded to the survey. On January 31st, the survey closed. This gives us a survey response rate of 10.23%.

In addition to the previous method for obtaining potential respondents, a Python script was developed to obtain users who have developed quantum software in the industry. To do this the GitHub API [42] was employed to retrieve the developers from repositories that contain code that has imported a quantum programming library or either code developed in a quantum programming language (Q#, OpenQASM, Qiskit, Cirq, Silq, Pytket, Pennylane, Forest SDK, DWave-Ocean, Alibaba, Braket, QCL, QML and Strawberry Fields). The script to get the repositories' owners can be found in [43]. To search for these users, two different filters have been used, depending on the language to be sought, but both searches by repositories were done using the *"search_repositories"* query.

To search for developers who have developed quantum software using a library, the filter employed consists of retrieving all repositories that contain the name of such quantum library in the *"README"* file and contain Python code (since almost all quantum libraries are implemented in Python). So, to find all users who have developed code in Qiskit with this search method, the filter would be *"in:readme qiskit language:python"*. This method was used because it is not possible to search for code by filtering by the libraries used. However, to find all developers who have used a quantum programming language, all repositories containing the desired programming language were searched using the *"language"* filter as GitHub is able to filter by the programming language. For example, searching repositories with source code developed in Microsoft's Q# language, the filter is *"language:Q#"* From all the repositories obtained, those with no more than 1 commit and those with no available mail were removed. Once this was done and having queried the GitHub API, the data was collected in a MongoDB database.

Fig. 1 shows the results of the number of contacts obtained following this method and the percentage they represent of a total of 553 emails collected. None of these emails was duplicated.

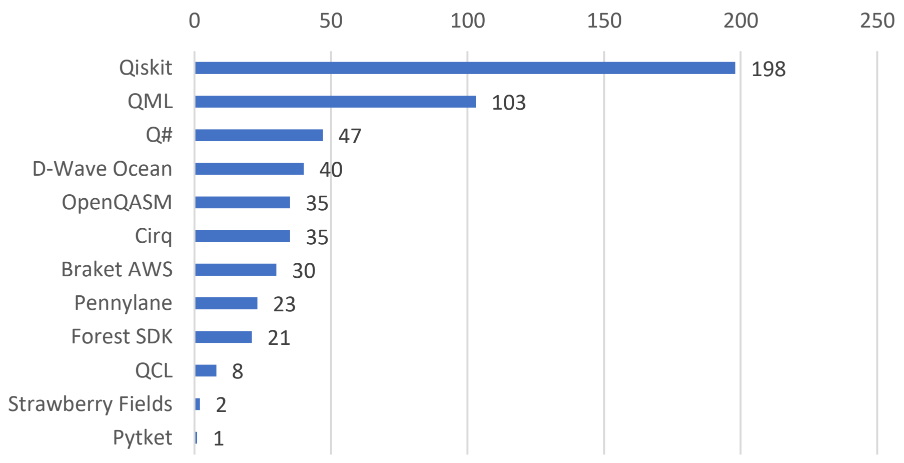

Fig. 1. Possible respondents obtained from GitHub.

3.4 Experimental Procedure

Fig. 2 shows the experimental procedure followed. The authors prepared the questionnaire from August to December 2022. During this time, the six authors worked on survey design, that is, defining the objectives, questions, and issues following the GQM approach. Throughout this period, the formulation criteria for the inquiries were thoroughly deliberated and finalized with the intention of stating questions that are concise, without ambiguity, and easy to comprehend. One author was responsible for technical tasks related to the definition of the questionnaire in the survey web application, including creating and adding survey questions, conducting the survey, and administering the system.

Once the questionnaire was ready, it was distributed between January and March 2023. Then, the responses obtained from the survey questionnaire were collected, pre-processed and, finally, analyzed.

3.5 Data Collection

Using *SurveySparrow* as the platform for implementing the web-based survey enables monitoring of survey results, as well as exporting raw data in CSV format and spreadsheets. This meant that every time someone responded to the survey, the spreadsheet is updated with new data.

Depending on the type of question (single or multiple choice), the format of the answers in the spreadsheet varies, but each row always represents one respondent. If the questions are single choice (i.e., only one answer can be ticked), the respondent's answer choice will appear in the corresponding cell of the answer. For multiple choice questions a column is generated for each possible answer.

Table 2 shows an example of the CSV output formatted. Each row represents the respondents, and the columns represent the questions. In this example, a total of two respondents and two questions can be found. In the *"single response"* column (see Table 2) there is a question where the respondents must answer *"What software components have you modeled?*. Respondent 1 stated that has modeled just quantum software and respondent 2 hybrid only.

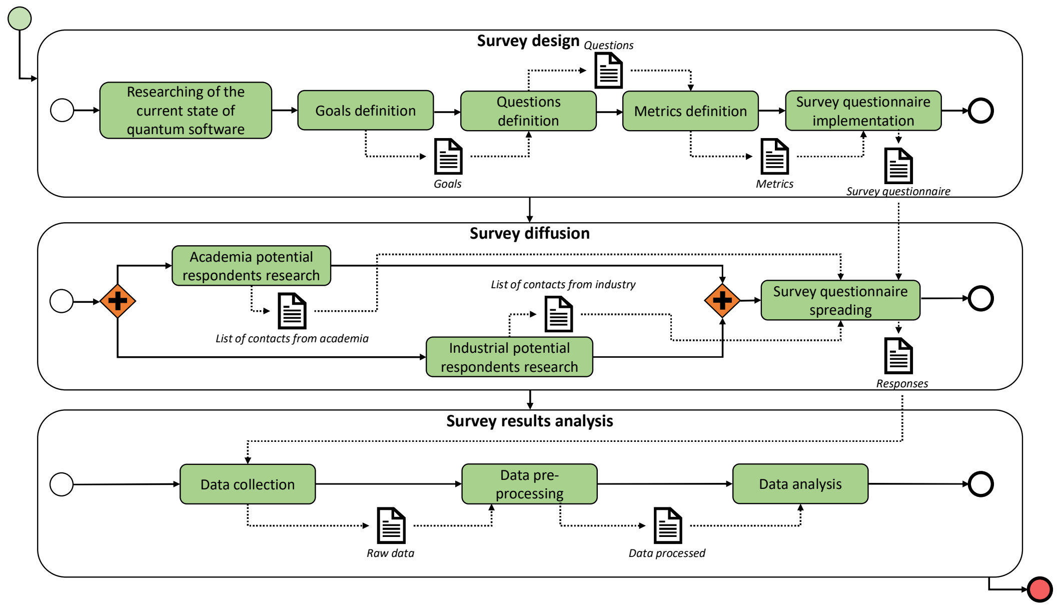
Survey design
Questions
Researching of the current state of quantum software
Goals definition
Questions definition
Metrics definition
Survey questionnaire implementation
Goals
Metrics
Survey questionnaire
Survey diffusion
Academia potential respondents research
List of contacts from industry
Survey questionnaire spreading
List of contacts from academia
Industrial potential respondents research
Responses
Survey results analysis
Data collection
Data pre-processing
Data analysis
Raw data
Data processed

Fig. 2. Experimental procedure followed.

The *"multiple response"* column (see Table 2) asks *"What is the target sector of hybrid software?"*, but each column represents the possible answer options. In the case of the first respondent, he only chose the *"Research"* option, while the second respondent chose both the *"Research"* and the *"IT"* options.

Table 2. Example of the format of the CSV output.

	Single response	Multiple response	
	What software components have you modelled?	What is the target sector of hybrid software? Research	What is the target sector of hybrid software? IT
Respondent 1	Quantum only	Research	
Respondent 2	Hybrid only	Research	IT

3.6 Data Analyisis

As mentioned above, the purpose of this survey is to describe the current state of quantum software development. For this reason, descriptive statistics have been used to analyze the data from the answers obtained. This type of statistic allows to summarize and describe the main features of the data collected. To visually represent the distribution of categorical variables, bar charts generated with Microsoft Excel have been employed.

In addition to descriptive statistics and bar charts, contingency tables were also employed to analyze the survey data. Contingency tables allow one to explore the relationship between two categorical variables. In this study, contingency tables were used to explore the relationships between research questions and objectives. Finally, a correlation analysis has been accomplished to explore the linear relationship between variables with Pythons library *dython*, version 0.7.1 [44]. To measure the strength and direction of the relationship between variables, the *Pearson* correlation coefficient has been used.

4 Results Analysis

The following sections will explain everything related to the results obtained from the survey. Section 4.1 gives a brief description of the profile of the respondents. Sections 4.2 to 4.4 shows the results that help to complete Goal 1 to Goal 3. Section 4.5 presents insights by analyzing various goals in combination. Section 4.6 provides a correlation study between answers. Finally, Section 4.7 addresses the validation of the collected data.

4.1 Respondents Profile

First of all, the respondents were asked questions to find out about the organization where they develop quantum code, in order to better establish a context. As mentioned above, a total of 56 people responded to the survey, where 47 of them do develop quantum software. Fig. 3 shows the results. The majority, 20 respondents, work in ”micro” (43%), 15 in ”medium” sized organizations (32%), 2 in medium-large (4%) and 10 in large organizations (21%). The percentages shown are calculated on the total number of answers obtained for each question. This is the same for both multi- and single-choice questions.

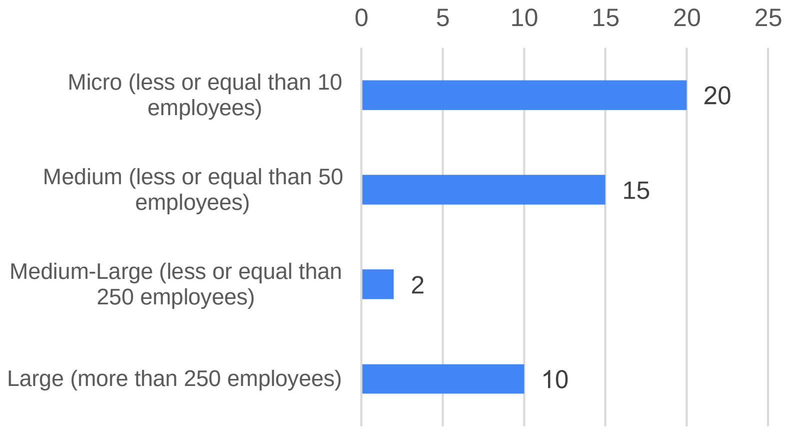

Fig. 3. Size of the organization of the respondents.

In addition, all respondents were asked which sectors the hybrid software they developed was aimed at, so multiple options could be chosen. The most chosen options are research (33; 38%), the IT sector (18; 21%) and education (9; 10%) (see Fig. 4). The first number in the tuples indicates the number of times an option has been chosen, and the second number the percentage that it represents over the total number of responses of the question.

Respondents were asked about the possible motivations behind developing quantum software (see Fig. 5). The vast majority (46; 53%) have in common the research as one of the organizations motivation. Then, to be ready for the future (20; 23%), better performance (15; 17%) and quality of the results (8; 9%).

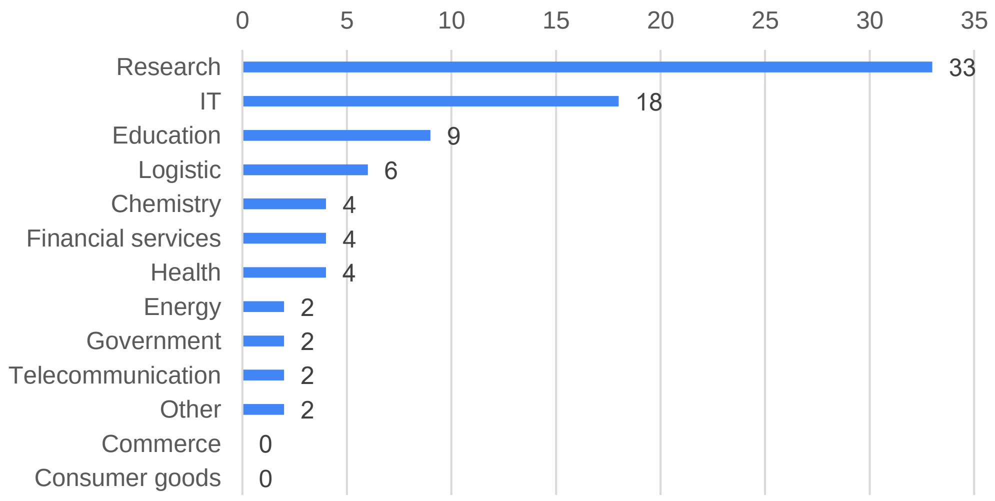

Fig. 4. Target sector of hybrid software.

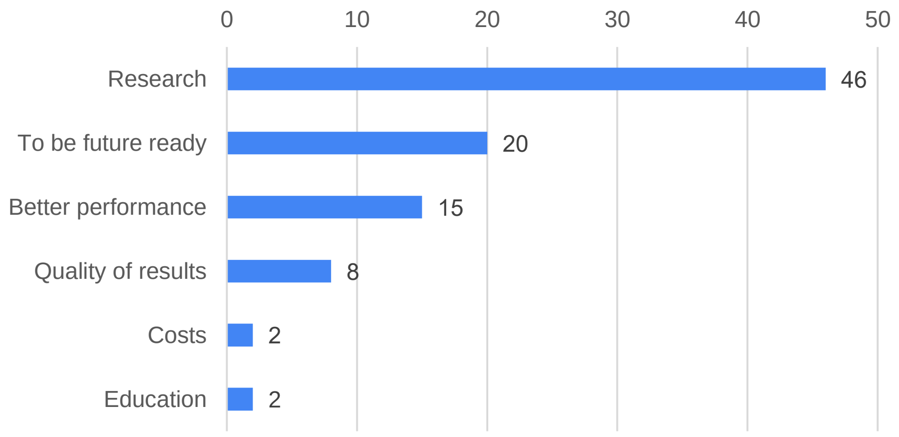

Fig. 5. Organization's motivation regarding the development of quantum software.

It can be concluded that the profile of the respondents to the survey is closely linked to the research. These respondents generally work in organizations with few employees. Furthermore, quantum software development is mainly research- and IT-oriented, which can be reflected in the motivations of the organizations where the respondents work.

4.2 Goal 1: Quantum Software Modeling

To characterize quantum software modeling with respect to modeling frameworks, standards, or tools the survey asked what software components to have been modelled during the development of hybrid quantum software. Fig. 6 shows the results of this question, where most of the respondents answered both (33; 69%). The second most chosen option is none (8; 17%), then only classical (6; 13%) and last only the quantum components (1; 2%).

Now, people who have modelled any type of component including quantum components (those respondents who answered either quantum-only or both classical-quantum in the previous question) have been asked what purposes they have used models for. Fig. 7 depicts the results of this questions, where the main use of models has been the conceptualization (31; 30%), documentation (24; 24%), and generation of code (20; 20%). The respondent who answered 'Other' stated that they use models for teaching.

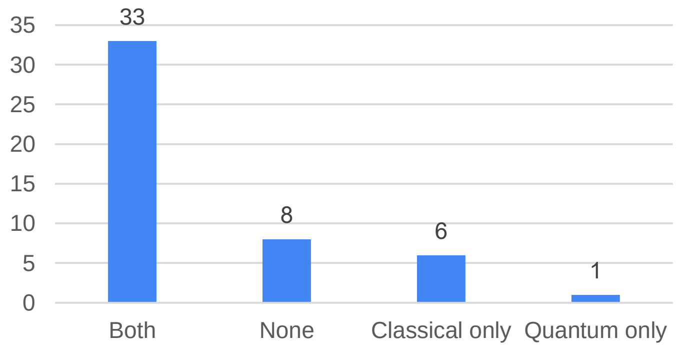

Fig. 6. RQ1.1 - Software components modeled.

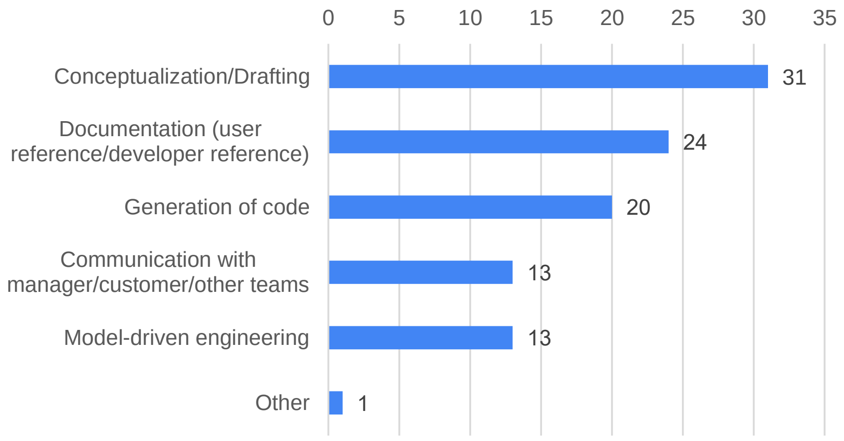

Fig. 7. RQ1.1 - Use of models.

Among the modeling languages or diagram types used during the development of quantum software (see Fig. 8), flowcharts are the most widely used (23; 21%), together with informal sketches (19; 17%). Within the modeling languages, UML stands out (16; 14%). Those who chose the "other" option reported that the diagram types they use are "Circuit diagrams", "LaTeX, ZX Calculus and Pennylane.

Fig. 9 depicts the results of use, creation and updating within the lifecycle phases. Within the phases of requirement analysis, architecture and design, and analysis, the creation of models covers the most. Then, the use of models tends to occur in the deployment, observability, implementation and testing phases. Finally, the updating of models is more common in the testing, analysis and observability phases.

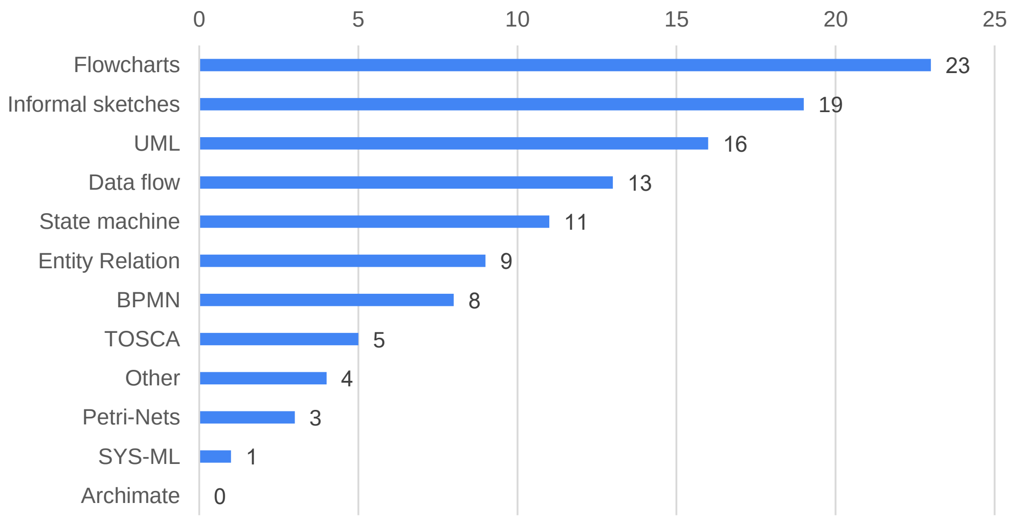

Fig. 8. RQ1.1 - Modeling languages or diagram types used during the development of quantum software.

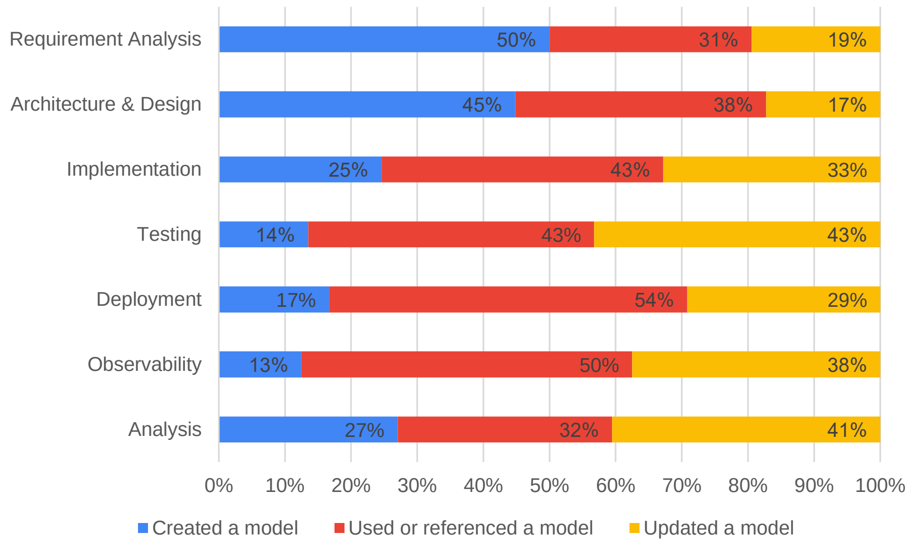

Fig. 9. RQ1.1 - Lifecycle phases where models have been used, created, or updated.

To find out how satisfied respondents are with the modeling languages used, a Likert scale has been used, where they rate their satisfaction from 1 to 5, where 5 represents the highest possible satisfaction and 1 the lowest. Instead of numbers, to represent the satisfaction degree emojis were employed. Fig. 10 illustrates the results of this question and the overall satisfaction is positive. Most respondents have neutral satisfaction, i.e., with a score of 3 (16; 44%). Fourteen respondents rated their satisfaction with 4 (39%) and three have rated their experience with 5 (8%). One person has rated their satisfaction with 2 (3%) and two respondents with 1 (6%).

Those who have a neutral or negative satisfaction rating, i.e., have a satisfaction rating of 3 or less, were asked in which lifecycle phase they would like to have more modeling support. Fig. 11 presents the results, where the architecture and design phase (12; 21%) is the phase requiring the most modeling support together with the implementation (10; 17%) and testing (9; 16%) phases.

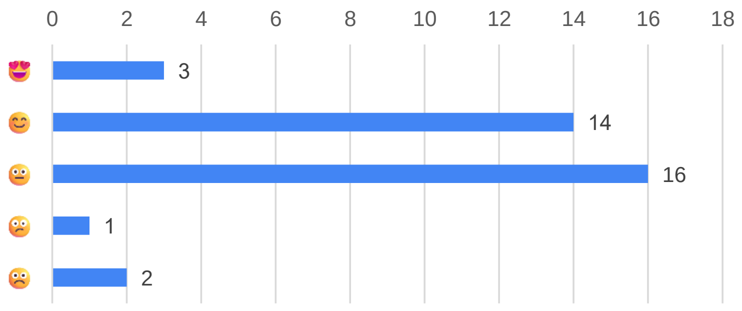

Fig. 10. RQ1.3 - Satisfaction degree with the employed modeling languages.

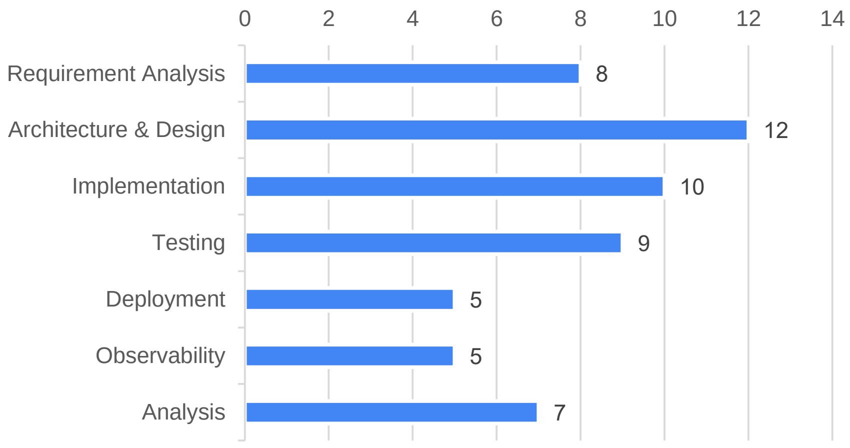

Fig. 11. RQ1.4 - Lifecycle phase where more modeling support is desired.

The nineteen respondents who perceived a lack of modeling were asked for what purpose they felt a lack of modeling. Fig. 12 shows the results of the question. For both hybrid software modeling and quantum software, the same lack of modeling was observed (8; 47%). One respondent stated not to perceive any lack of modeling and no one for modeling classical software. It demonstrates that more modeling techniques are being demanded and while classical software has enough modeling support, these tools might not help much with hybrid software modeling.

The responses to the degree of satisfaction with the modeling languages used have been combined with the programming languages used. This helps to understand how confident quantum software developers feel regarding the modeling languages used. To accomplish these objectives, a contingency table has been built based on the results obtained from the previous questions. Fig. 13 shows the results obtained sorted by the number of times the modeling tool has been chosen. The BPMN modeling language was the highest rated modeling language, along with TOSCA. Although these languages are not the most widely used, those are the best-rated. The lowest rated are ArchiMate, SYS-ML and State machine diagrams.

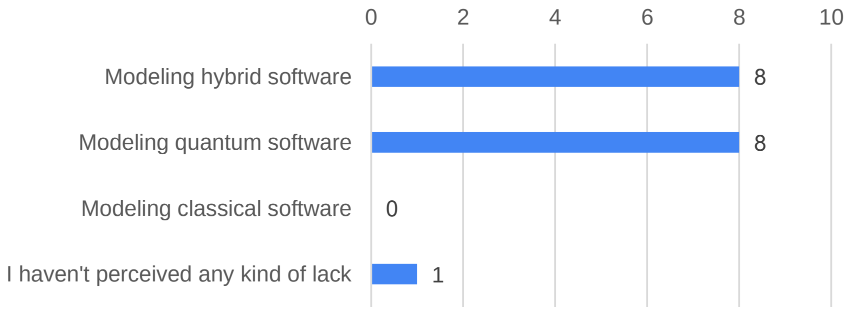

Fig. 12. RQ1.3 - Purposes where a lack of modeling is perceived.

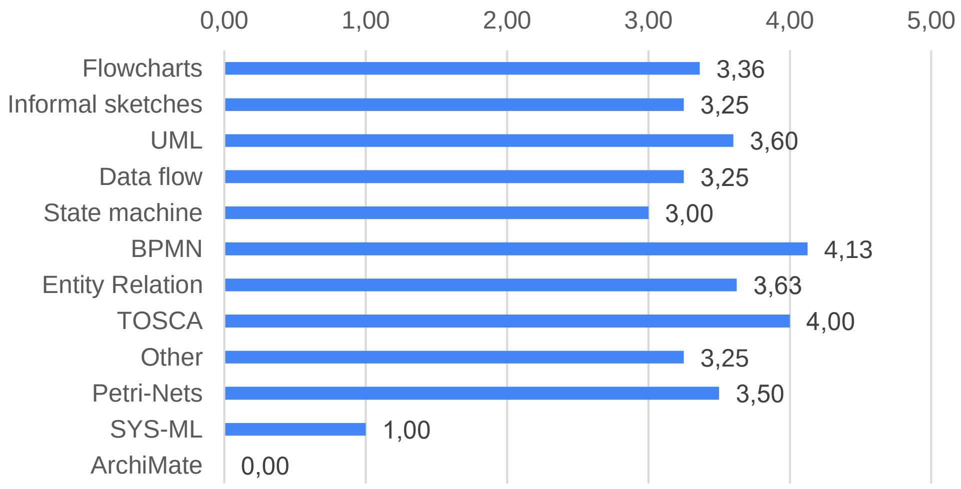

Fig. 13. RQ1.1 & RQ1.3 - Average of the satisfaction degree with the employed modeling languages.

In summary, modeling in quantum software development does exist and is mainly used to model hybrid software for documentation and conceptualization (among other things). For this purpose, UML has been the most widely used modeling language along with flowcharts and informal sketches. Within the phases where modeling has been used the most, the implementation phase and architecture and design are those where models are most used. However, although there is generally positive satisfaction with modeling tools, there is still a perceived lack of modeling in both hybrid and quantum software modeling.

4.3 Goal 2: Implementation of Quantum Software

The second goal consists of characterizing which SDKs, tools, compilers, and languages are used during the implementation of quantum software. To do this, respondents were first asked what programming languages or toolkits they have used to develop quantum code. Fig. 14 shows the results of this question, where the use of IBM's programming language, Qiskit (45; 27%), stands out. Next, the most used languages are Cirq (18; 11%) and OpenQASM (18; 11%). Those who chose "Other" said they used "Duke ARTIQ extensions (DAX)", "Intel Quantum", "Qibo" and "Quantum Tensor Flow".

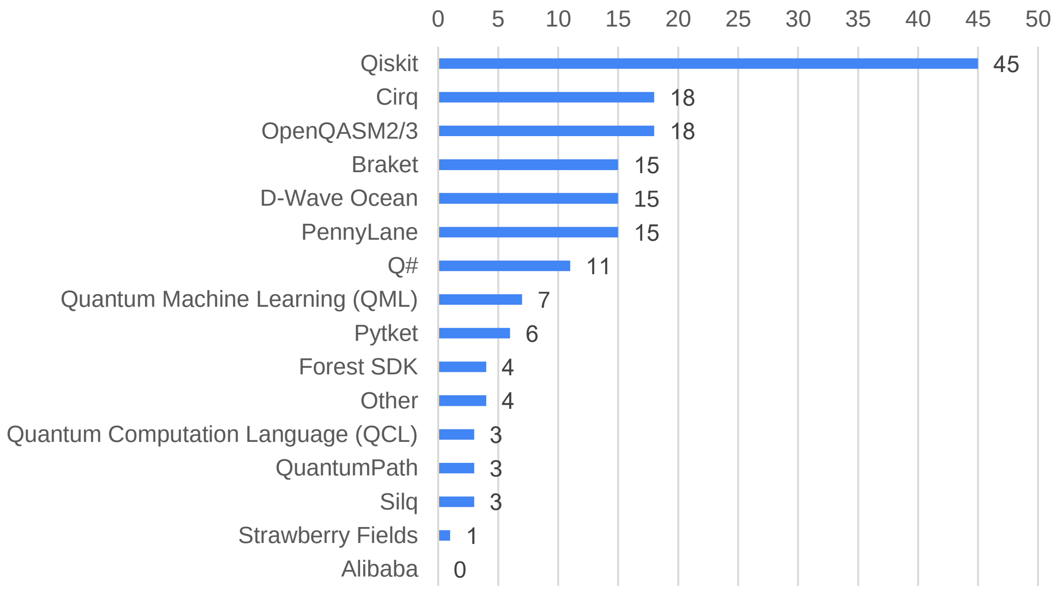

Fig. 14. RQ2.2 - Quantum programming languages or toolkits used.

The respondents were asked which classical programming languages they used to develop the hybrid code. Fig. 15 shows the results, where Python (49; 50%) was the most used programming language, as it is the programming language with the largest number of libraries for the development of quantum software. Next, the languages used are Java (12; 12%), C++ (12; 12%) and C# (9; 9%). Those who chose the "other" option stated that the classical programming language used was Julia.

The respondents were asked about the type of architectures followed to build the hybrid application. Fig. 16 shows that the main architecture type is monolithic type (17; 33%). The next most used is a multi-layered type (14; 27%), followed by a service-oriented type (13; 25%). The last option is the event-based architecture type (8; 15%).

Fig. 17 shows a plot obtained by the intersection of the answers obtained from the classic programming languages used for the development of hybrid software and the type of architecture implemented. Those who have developed software with a monolithic architecture, Python came first along with C++, JavaScript, and C. Respondents who followed service-oriented architectures, Java, C#, and C++ were used. For those who followed a multi-layered architecture used C, C#, and Java. Finally, for those who followed an event-driven architecture have usually used Rust, JavaScript, and C.

The respondents were asked whether hybrid software had been tested. Most of the re-

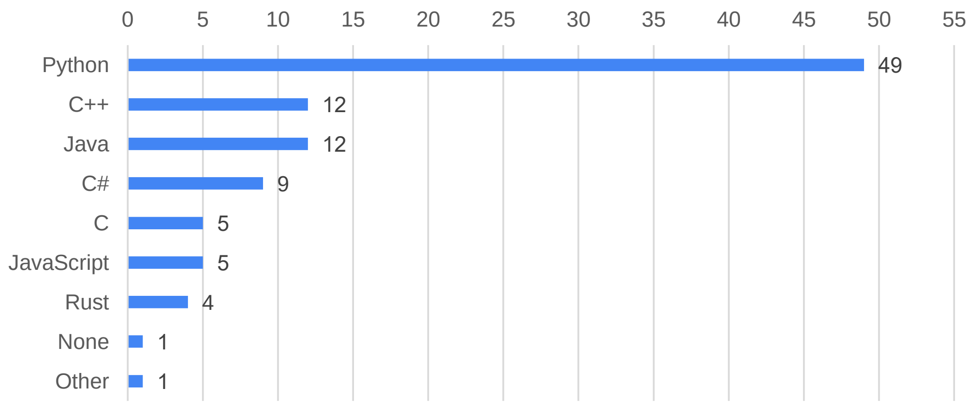

Fig. 15. RQ2.2 - Classical programming languages used for developing hybrid software.

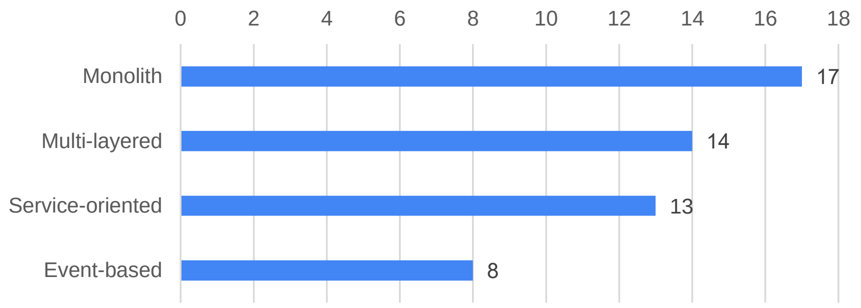

Fig. 16. RQ2.3 - Types of architectures of hybrid application systems.

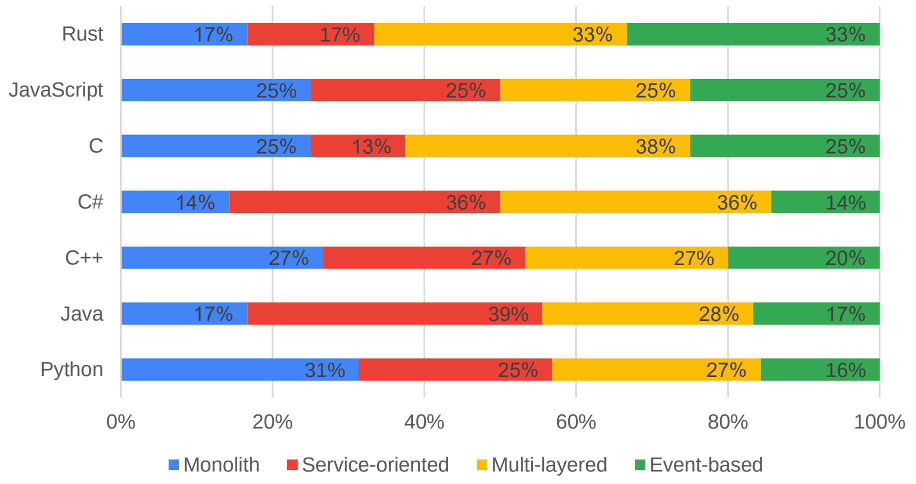

Fig. 17. RQ2.2 & RQ2.3 - Type of architecture depending on the classical programming languages.

spondents answered yes (41; 73%). Those who answered yes were asked which components they had tested. Fig. 18 displays the results, where the vast majority test both quantum and classical software (32; 78%). Five respondents only test quantum components (12%), and four respondents just test classical components (10%).

The responses obtained from the tested software components were compared with the responses of the type of technique followed. Fig. 19 shows the results, where the type of components tested is broken down depending on the type of testing technique carried out. Those respondents who test both classical and quantum components usually follow a black-box testing approach (25; 69%). Those respondents who only test classical software mostly follow a white-box type of testing (3; 60%), while those who only test quantum software employ black-box testing (3; 100%).

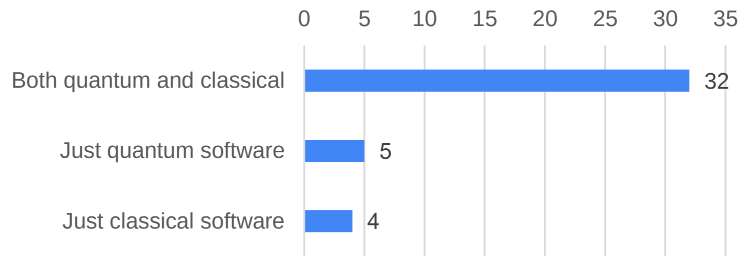

Fig. 18. RQ2.4 - Software components tested.

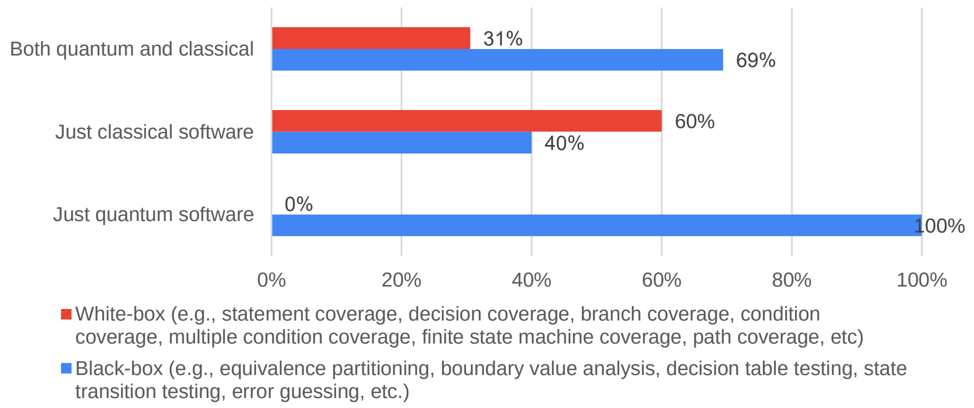

Fig. 19. RQ2.4 - Type of testing technique depending on the components tested.

With this goal in mind, it can be concluded that, according to this survey, the most widely used programming languages are those developed by IBM (Qiskit and OpenQASM), in addition to Cirq. Furthermore, it is possible to observe an absolute predominance of Python as the most widely used classical programming language for developing hybrid software. The fact that most of the quantum software libraries belong to this language contributes to this. In addition, the monolithic software architecture is the most widely used architecture. This may be due to the fact that, among other reasons, research is the primary motivation for hybrid software development (see Section 4.1). Finally, it can be ascertained that testing exists in hybrid software implementation.

4.4 Goal 3: Operation of Quantum Software

Goal 3 consists of characterizing which SDKs, tools, compilers, and languages are used during the operation of quantum software. First, respondents were asked what execution environment they had used (see Fig. 20). The local simulator option (39; 30%) was the most chosen environment together with the cloud simulator (34; 26%). Then, the most chosen environments were cloud QPU (31; 24%), hybrid runtime (20; 16%), and finally private QPU (5; 4%).

Respondents were asked what they consider to be the most important reasons for switching from a simulation environment to a real quantum computer. According to the data collected (see Fig. 21), most of the respondents consider the validity of the results in the simulation environment (38; 31%) and the next reason is that the tests have been passed correctly (27; 22%). Next, compulsion (22; 18%) is an important point, followed by the fact that execution on quantum computers has become affordable (17; 14%) and the need for program complexity (16; 13%). Those who said ”Other” (2; 2%) stated that other criteria are ”Promising results on simulator” and ”reasonable to expect that the quantum hardware can execute the software without succumbing to noise”.

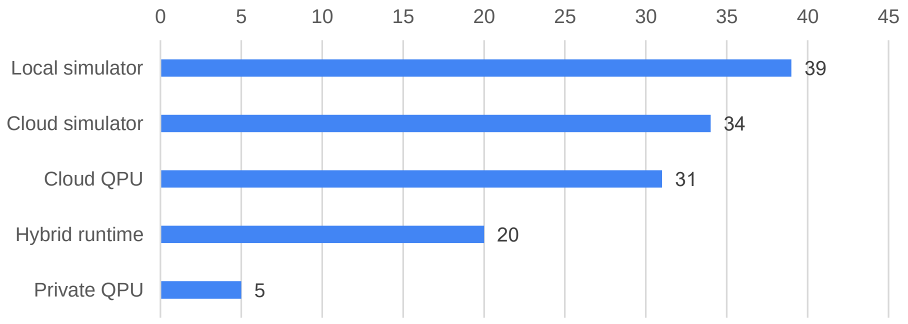

Fig. 20. RQ3.1 - Execution environment used.

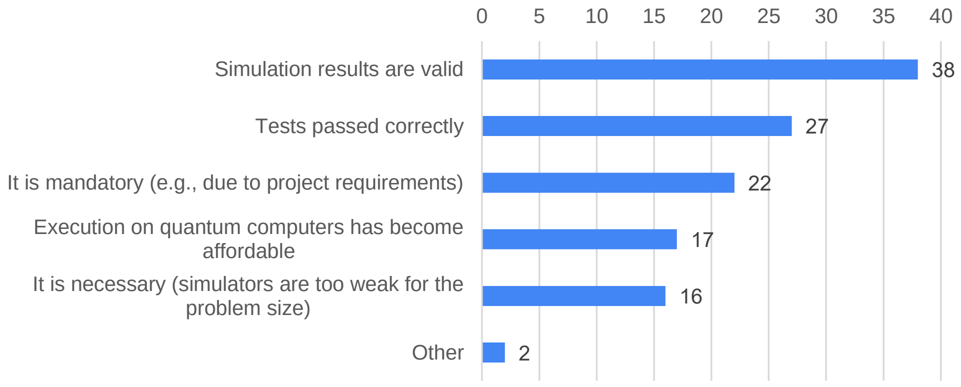

Fig. 21. RQ3.1 - Criteria for moving from a simulation environment to a real quantum computer.

Respondents were asked whether they used workflow technologies and language converters. Fig. 22 illustrates the results of these questions. Regarding workflow technologies, most of the respondents do not use them (39; 87%). Concerning the use of language converters, most of the respondents stated that they did not use them (36; 80%).

Although workflow technologies are not widely used, it is important to know how and for what purpose they are used. To do this, the responses to the question on the type of execution environment respondents have used have been intersected with those who responded that they do use workflow technologies. The results can be seen in Fig. 23, where most of the respondents who use workflow technologies use cloud QPUs (6; 20%), closely followed by cloud simulators (5; 17%).

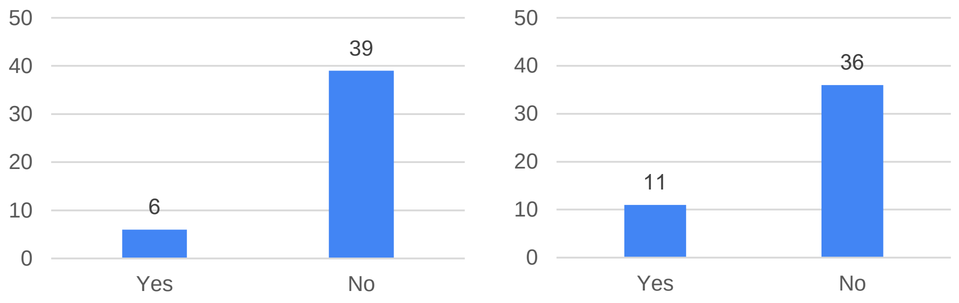

Fig. 22. RQ3.2 - Use of workflow technologies (left) and RQ3.3 - language converters (right).

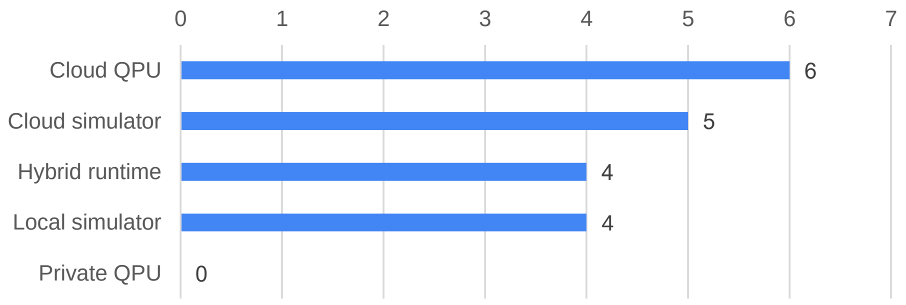

Fig. 23. RQ3.1 & RQ3.2 - Execution environment used with workflow technologies.

In summary, this survey has shown that the operations phase as understood in classical software is not fully developed. This can be assumed since most developers use local simulators as execution environments and the absence of workflow technologies. Moreover, the fact that the main focus of quantum software development is research and that the most commonly used architecture type is monolithic are further indications of this. Perhaps in the future, when the quantum computing paradigm is sufficiently mature, it will be possible to speak of an operations phase as such.

4.5 Cross-goals Insights

The previous sections have been able to fulfill the purpose of addressing the stated objectives. However, in order not to limit the characterization of the current state of quantum software development, some inter-objective analyses have been performed. These analyses use the same approach of intersecting the answers obtained from several questions but with the answers belonging to different objectives.

The first analysis inter-objective accomplished is the itemization of the type of diagrams employed depending on the type of architecture followed. Fig. 24 shows the results. Respondents using informal sketches tend to use a monolithic architecture. Respondents using TOSCA as a modeling language use either a monolithic or a service-oriented architecture, which is to be expected since TOSCA is aimed at modeling service-oriented architectures. The UML modeling language is used with multi-layered projects. This may be due to the versatility of UML to model the different layers of a project. Those that mainly use an event-based architecture are those that model with Petri-Nets and entity relation diagrams.

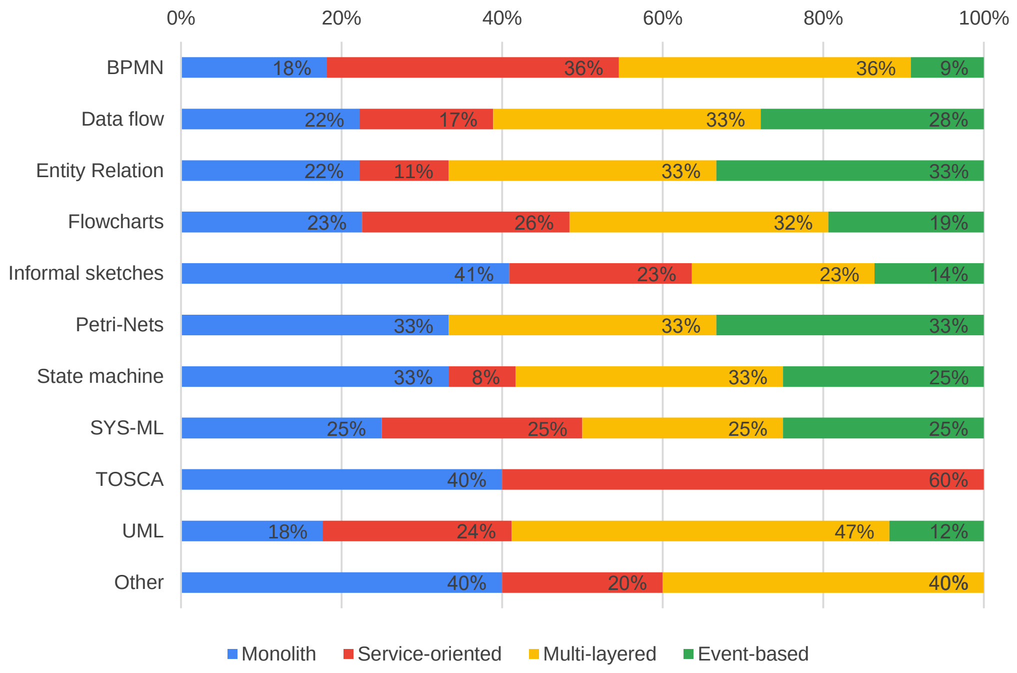

Fig. 24. Architecture followed within each modeling language.

The results obtained from the questions of which quantum programming languages respondents use have been disaggregated by the execution environment used. This provides insight into the execution environments of each programming language. Fig. 25 shows the results obtained. Alibaba is not represented since any respondent chose such an option. Those respondents who use QCL and QuantumPath are the ones who use Cloud QPUs the most on average to run their quantum programs. Those who develop programs in Forest SDK mostly use hybrid runtimes and cloud simulators to run their programs. As expected, there is a general consensus that local execution of programs is the most accessible way to run these programs. Finally, respondents using Silq and Strawberry Fields mostly run their programs on private QPUs compared to the rest. This may be due to the fact that the respondents using these languages work in organizations with access to private quantum computers.

The responses obtained from the use of models in the different phases of the quantum soft-

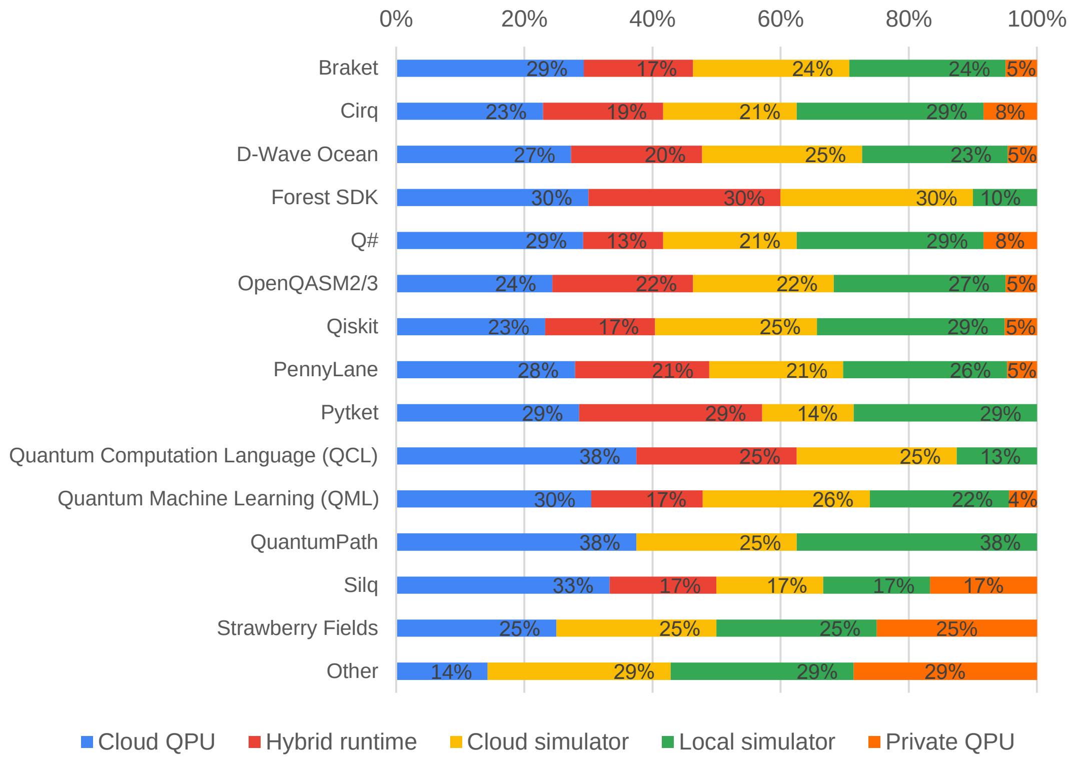

Fig. 25. Execution environment used for each quantum programming language.

ware development cycle have been broken down (see Fig. 9) and intersected with the testing techniques used by the respondents. Fig. 26 shows the results, where those respondents who create models during the testing phase usually follow a black-box testing approach (2; 67%). Respondents who use models during this phase tend to take a black-box testing approach (8; 57%) rather than a white-box approach (6; 43%). Furthermore, those who update models during this phase take a black-box testing approach (9; 64%).

The analyses carried out on the responses obtained are just a few examples of all the analyses that can be carried out. All the answers can be found in [43], and if any researcher would like to carry out further analysis, the data is available.

4.6 Correlation Study

A correlation study has been conducted with the responses obtained. First, to apply the appropriate algorithms, it was necessary to transform the dataset into a numeric format. For this, the One Hot Encoding coding method has been applied to the dataset of the responses obtained. This strategy involves creating binary columns for each unique value that exists in the categorical variable being encoded. In the case that a value is present in the record, the corresponding column for the present value is marked with *1*, and if it is not present, it is marked with *0*.

Once the dataset has been binarized, the 'association' function from the Pythons library

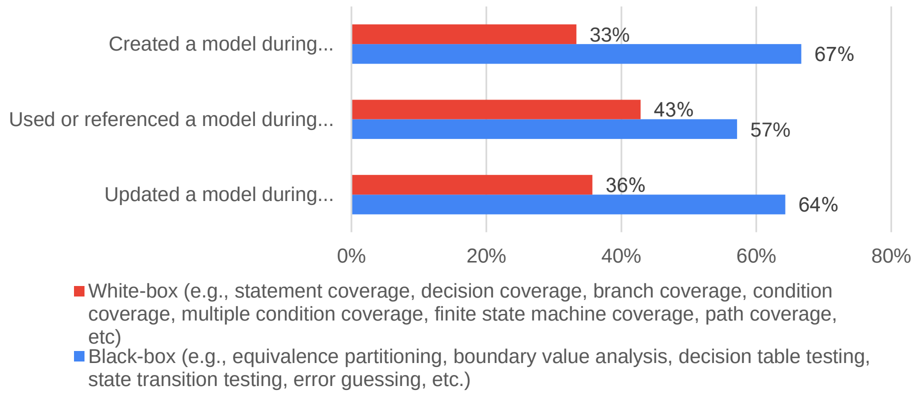

Fig. 26. Testing techniques regarding the use, creation, and update of models during the test phase.

dython (version 0.7.1) [42]. This function calculates the correlation or strength of association between features in the dataset, including both categorical and continuous features, using Pearson's R for continuous-continuous cases, correlation ratio for categorical-continuous cases, and Cramer's V or Theil's U for categorical-categorical cases. To calculate the correlation coefficient, Pearson's correlation coefficient has been used, where values greater than 0.5 indicate a strong correlation. The heat map generated from the association study can be observed in [43]. Table 3 displays the results of filtering the data obtained from the correlation study and obtaining the metrics that have obtained a correlation index equal to or greater than 0.80. Some of the insights that can be extracted are the following:

- Survey respondents who use informal sketches often employ lines and boxes as a modeling approach, and the reason for using them is that sketches are faster.
- Survey respondents who use UML as a modeling language employ class diagrams to design quantum or hybrid software.
- Survey respondents who use deployment diagrams in UML use QPath as a workflow technology.
- Survey respondents who are not satisfied with current modeling tools often use Silq as a quantum programming language.
- Survey respondents who employ workflow technologies consider the most important reasons for choosing a workflow technology to be related to functional requirements, features of the tool, service availability, quality of the documentation, and extensibility of the technology.
- Survey respondents who use Quantum Programming Studio as a language converter do so because it meets the functional requirement of functionality.

Furthermore, the negative correlation between the metrics has been studied (the data set with all the correlations can be found in [43]). However, in general, no significant relationships have been found. Some exceptions are:

Table 3. Metrics with an association over 0.8.

Question 1	**Question 2**	**Correlation**
Which modeling languages or diagram types have you used during the development of quantum software? - Informal sketches	Regarding informal sketches... Which approach did you follow for modeling an informal sketch? - Lines and boxes	0.92
	What were your reasons to employ informal sketches? - Sketches are faster	0.92
Which modeling languages or diagram types have you used during the development of quantum software? - UML	Regarding UML... What kind of UML diagrams have you used during the modeling of (hybrid) quantum software? - Class diagrams	0.95
Regarding UML... What kind of UML diagrams have you used during the modeling of (hybrid) quantum software? - Deployment diagram	What workflow technologies have you used for orchestrating (hybrid) quantum software? - QPath (www.quantumpath.es)	1
Regarding UML... What kind of UML diagrams have you used during the modeling of (hybrid) quantum software? - Other	What were your reasons for choosing these quantum programming languages or toolkits (regarding non-functional requirements)? - Other	1
Regarding informal sketches... Which approach did you follow for modeling an informal sketch? - Lines and boxes	What were your reasons to employ informal sketches? - Sketches are faster	0.91
How satisfied are you with the employed modeling languages? (Not considering informal sketches) - 1	What quantum programming languages or toolkits have you used for developing software? - Silq	0.81
What were your reasons for choosing these quantum programming languages or toolkits (regarding functional requirements)? - Other	What were your reasons for choosing these execution environments (regarding functional requirements)? - Other	1
Do you use workflow technologies or similar ones for orchestrating (hybrid) quantum software? - Yes	What were your reasons for choosing these execution environments (regarding functional requirements)? - Features	0.9
	What were your reasons for selecting these workflow technologies (regarding non-functional requirements)? - Service availability	0.9
	What were your reasons for selecting these workflow technologies (regarding non-functional requirements)? - Quality of documentation/tutorials	0.9
	What were your reasons for selecting these workflow technologies (regarding non-functional requirements)? - Extensibility	0.9
What were your reasons for selecting these workflow technologies (regarding functional requirements)? - Availability of pre-implemented workflows/algorithms	What were your reasons for selecting these workflow technologies (regarding functional requirements)? - Features	0.88
	What were your reasons for selecting these workflow technologies (regarding non-functional requirements)? - Service availability	0.88
	What were your reasons for selecting these workflow technologies (regarding non-functional requirements)? - Quality of documentation/tutorials	0.88
What were your reasons for selecting these workflow technologies (regarding functional requirements)? - Features	What were your reasons for selecting these workflow technologies (regarding non-functional requirements)? - Service availability	1
	What were your reasons for selecting these workflow technologies (regarding non-functional requirements)? - Quality of documentation/tutorials	1
	What were your reasons for selecting these workflow technologies (regarding non-functional requirements)? - Traceability	0.88
What were your reasons for selecting these workflow technologies (regarding non-functional requirements)? - Service availability	What were your reasons for selecting these workflow technologies (regarding non-functional requirements)? - Quality of documentation/tutorials	1
	What were your reasons for selecting these workflow technologies (regarding non-functional requirements)? - Traceability	0.88
What were your reasons for selecting these workflow technologies (regarding non-functional requirements)? - Quality of documentation/tutorials	What were your reasons for selecting these workflow technologies (regarding non-functional requirements)? - Traceability	0.88
Which standalone language converters (translator) have you employed? - Quantum Programming Studio	What were your reasons for selecting these language converters (regarding functional requirements)? - Functionality	1
What were your reasons for selecting these language converters (regarding functional requirements)? - Language support	What were your reasons for selecting these language converters (regarding non-functional requirements)? - Ease-of-use	0.88
What were your reasons for selecting these language converters (regarding non-functional requirements)? - Ease-of-use	What were your reasons for selecting these language converters (regarding non-functional requirements)? - Performance	0.86

- Some respondents who consider the "Compliance" requirement when choosing a modeling language do not use the Qiskit quantum programming language (correlation index between these two metrics of -0.55).
- Survey respondents who rated their satisfaction as 2 out of 5 regarding current modeling languages do not typically work for organizations with Research as motivation regarding the development of quantum software (correlation index between these two metrics of -0.48).
- Survey respondents who use Qiskit as a quantum programming language do not typically use Quantum Programming Studio as a language converter (correlation index between these two metrics of -0.48).
- Survey respondents who develop monolithic quantum or hybrid software have not chosen Execution time as a reason for choosing an execution environment (correlation index between these two metrics of -0.47).

In summary, thanks to the correlation study conducted, several insights have been obtained that would not have been possible to discover otherwise. These insights have been extracted from both strongly correlated metrics and metrics that have a negative correlation.

4.7 Threats to Validity

The validity of the study findings could be undermined by several potential threats. These threats have been classified according to [45].

Construct validity concerns generalizing the result of the experiment to the concept or theory behind the experiment. A major threat that could affect the construct validity is the questionnaire bias. This occurs when the questions on the survey are worded in a way that is confusing, ambiguous, or biased, leading to inaccurate responses. To mitigate this threat, some researchers (who have not participated in the final survey) were surveyed in a pilot study. The collected feedback was used to improve the questionnaire and mitigate this threat.

Threats to internal validity are the extent to which particular factors affect the methodological rigor. These threats might appear during the preparation and cleaning of the data. In order to deal with that, a protocol performed by at least two researchers was established. This allowed a systematic analysis and review of the responses and data generated. Once the questionnaire was closed, the responses were stored and processed, following the experimental procedure explained in Section 3.4.

Threats to the conclusion validity are concerned with issues that affect the ability to draw the correct conclusion about relations between the treatment and the outcome of an experiment. These threats may affect the reliability of the measures to alleviate these threats, measures that have helped address each of the research questions have followed an exhaustive protocol explained in Section 3.6. In addition, the survey responses obtained are available in [43], allowing the experiment to be replicated and ensuring that the same conclusions can be drawn.

Threats to external validity are conditions that limit our ability to generalize the results of our experiment to industrial practice. The sampling bias could be a threat to external validity as a total of 57 answers were obtained. However, after the data filtering a total of 49 answers remained. This could be considered a sample not representative of the target population, leading to results that cannot be generalized to the population. Nevertheless, the fact that the quantum software paradigm is not yet mature and with a narrow community means that findings could be considered representative to a certain extent.

5 Discussion

Thanks to this survey, several previously unknown insights into quantum software development have been discovered. Firstly, it was found that modeling during quantum software development does exist up to a certain extent. Thus, the most commonly used are UML, flowcharts, and informal sketches. Secondly, regarding the implementation of quantum software, according to the survey results, the most commonly used quantum programming languages are Qiskit, OpenQASM, and Cirq, with a predominance of monolithic architectures. Finally, it may be too early to talk about the operation of quantum software, as mainly local simulators are used as execution environments and there is, in most cases, a lack of workflow technologies.

During the analysis of the survey responses, several unexpected findings have been discovered that were previously unknown. Among them, there is a positive satisfaction with the modeling tools used during quantum software development. This may be related to the fact that informal sketches are one of the most widely used tools for modeling quantum software. Informal sketches are a tool that can be adapted to any problem, as they do not have a specific format. Yet, more modeling support would be needed in the phase of architecture and design. Another key finding discovered is that testing in hybrid software does exist and that depending on the component to be tested (hybrid or quantum-only), one testing approach or another is used. This may be due to the lack of testing tools available that are aimed only at testing quantum software.

Although the number of respondents to the survey may seem small, it must be taken into account that there are currently not a large number of quantum software developers. However, the fact that it is not possible to determine a specific number to be considered representative of this population can be seen as a limitation of this study. While we believe this limitation has not impacted the primary outcome of the study, the survey responses and data are offered to the scientific community in order to generate new insights that have not been found during the analysis of the results.

6 Conclusion and Future Work

Quantum computing is increasingly drawing the attention of organizations and governments, as it is expected to be implemented in our information systems in the next few years. To address this task effectively, it is necessary to transfer knowledge from software engineering to quantum computing. This will allow companies and institutions to develop quantum software in an industrial and controlled way. However, there is no evidence of the modeling tools that are currently employed during quantum software development.

This study conducts a survey questionnaire to find out the current status of software modeling within quantum software development. In addition, respondents were also asked about the tools they use to develop quantum software, making a distinction between the implementation and operation phase. This distinction will help to better understand the context of quantum software development and to understand how mature quantum computing is with respect to software.

With this research, it is hoped that the state of the art of quantum computing with respect to software modeling will be studied in more depth in the future. Furthermore, having investigated the implementation of the software has allowed us to discover the architectures that are followed and how the testing of hybrid programs is carried out, among other things. It is also essential to know the operation of quantum software in order to see how it evolves and changes in the whole lifecycle when this paradigm is more mature.

In future work, the knowledge gained from this survey is expected to help in other research areas. These areas include further research on software modeling in the field of quantum computing and additional adaptations of good practices from the field of classical software engineering to quantum software. With this research, we expect to contribute to the (necessary) development of the QSE field.

Acknowledgements

This work has been supported by grants PID2022-137944NB-I00 (SMOOTH Project) and PDC2022-133051-I00 (QU-ASAP Project) funded by MICIU/AEI/ 10.13039/501100011033 and by the European Union NextGenerationEU/PRTR. Also, this project has been partially funded by the BMWi project PlanQK (01MK20005N).

References

1. Zhao, J. Quantum software engineering: Landscapes and horizons. 2020.
2. Prez-Castillo, R., Jimnez-Navajas, L., & Piattini, M. Modelling Quantum Circuits with UML. 2021.
3. Prez-Delgado, C. A., & Perez-Gonzalez, H. G. Towards a quantum software modeling language. In Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops. 2020.
4. Prez-Castillo, R., Jimnez-Navajas, L., & Piattini, M. QRev: migrating quantum code towards hybrid information systems. 2021: p. 1-30.
5. Weder, B. et al. Quantum software development lifecycle in Quantum Software Engineering. 2022 Springer. p. 61-83.
6. Wild, K. et al. TOSCA4QC: Two Modeling Styles for TOSCA to Automate the Deployment and Orchestration of Quantum Applications. In 2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC). 2020.
7. Zapata Computing. The First Annual Report on Enterprise Quantum Computing Adoption. 2022.
8. Unitary Fund Team. Quantum Open Source Software Survey. 2022.
9. Garca de la Barrera, A. et al. Quantum software testing: State of the art. 2021: p. e2419.
10. Weder, B. et al. The quantum software lifecycle. In Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software. 2020.
11. Ricardo Prez-Castillo, Manuel ngel Serrano, Jos A. Cruz-Lemus, Mario Piattini. Guidelines to use the ICSM for developing quantum-classical systems. Quantum Inf. Comput. 24(1&2): 71-88 (2024). https://doi.org/10.26421/QIC24.1-2-4.
12. Basili, V. R., Caldiera, G., & Rombach, D. H. The goal question metric approach. 1994: p. 528-532.
13. Feynman, R. P. Simulating physics with computers in Feynman and computation. 2018 CRC Press. p. 133-153.
14. Gyongyosi, L., & Imre, S. A Survey on quantum computing technology. Computer Science Review

2019. 31: p. 51-71.
15. Svore, K. et al. Q#: Enabling Scalable Quantum Computing and Development with a High-level DSL in Proceedings of the Real World Domain Specific Languages Workshop 2018. 2018 Association for Computing Machinery: Vienna, Austria. p. Article 7.
16. Cross, A. W. et al. Open quantum assembly language. 2017.
17. Cross, A. The IBM Q experience and QISKit open-source quantum computing software. In APS March Meeting Abstracts. 2018.
18. Hancock, A. et al. Cirq: A Python Framework for Creating, Editing, and Invoking Quantum Circuits. 2019.
19. Killoran, N. et al. Strawberry fields: A software platform for photonic quantum computing. 2019. 3: p. 129.
20. Hevia, J. L., Peterssen, G., & Piattini, M. QuantumPath: A quantum software development platform. Software: Practice and Experience 2021.
21. Services, A. W. Amazon Braket. 2020; Available from: https://aws.amazon.com/es/braket/.
22. Hevia, J. L. et al. Quantum computing. 2021. 38(5): p. 7-15.
23. Garcia-Alonso, J. et al. Quantum software as a service through a quantum API gateway. 2021. 26(1): p. 34-41.
24. Moguel, E. et al. Quantum service-oriented computing: current landscape and challenges. 2022. 30(4): p. 983-1002.
25. Piattini, M. et al. The Talavera Manifesto for Quantum Software Engineering and Programming. In QANSWER. 2020.
26. Piattini, M. et al. Toward a Quantum Software Engineering. 2021. 23(1): p. 62-66.
27. Weder, B. et al. Integrating quantum computing into workflow modeling and execution. In 2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC). 2020. IEEE.
28. Jimnez-Navajas, L., Prez-Castillo, R., & Piattini, M. Reverse Engineering of Quantum Programs Toward KDM Models. In International Conference on the Quality of Information and Communications Technology. 2020 Springer. p. 249-262.
29. Blanco, M. ., & Serrano, M. Quantum Information Technology Governance System in Quantum Software Engineering. 2022 Springer. p. 39-59.
30. Li, H., Khomh, F., & M. Openja. Understanding Quantum Software Engineering Challenges: An Empirical Study on Stack Exchange Forums and GitHub Issues. In 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME). 2021. IEEE.
31. Khan, A. A. et al. Agile Practices for Quantum Software Development: Practitioners Perspectives. 2022.
32. De Stefano, M. et al. Software engineering for quantum programming: How far are we? Journal of Systems and Software 2022. 190: p. 111326.
33. C. Hughes, D. Finke, D. -A. German, C. Merzbacher, P. M. Vora and H. J. Lewandowski, "Assessing the Needs of the Quantum Industry," in IEEE Transactions on Education, vol. 65, no. 4, pp. 592-601, Nov. 2022, doi: 10.1109/TE.2022.3153841.
34. D-Wave. Survey about the quantum commercial trends. 2022; Available from: https://www.dwavesys.com/quantum-commercial-trends/.
35. Classiq. Classiq's survey. 2021; Available from: https://www.classiq.io/insights/2021-survey-part-1.
36. Basili, V. R., Shull, F., & Lanubile, F. Building knowledge through families of experiments. 1999. 25(4): p. 456-473.
37. SurveySparrow. SurveySparrow webpage. 2023; Available from: https://surveysparrow.com/.
38. aQuantum. aQuantum's webpage. 2023; Available from: https://www.aquantum.es/.
39. International Conference on Software Engineering (ICSE). ICSE's webpage. 2023; Available from: http://www.icse-conferences.org/.
40. International Conference on Quantum Computing (QCE). QCE's web page. 2022; Available from: https://qce.quantum.ieee.org/2022/.
41. International Conference on the Quality of Information and Communications Technology

(QUATIC). QUATIC's webpage. 2023; Available from: http://www.quatic.org/.
42. GitHub. GitHub REST API documentation. 2022; Available from: https://docs.github.com/en/rest?apiVersion=2022-11-28.
43. Luis Jimnez-Navajas, R. P.-C. Quantum Software Development Survey's GitHub repository. 2023; Available from: https://github.com/luisjimenez24/quantumsoftwaresurvey.
44. Zychlinski, S. Dython's GitHub repository. 2022; Available from: https://github.com/shakedzy/dython.
45. Wohlin, C. et al. Experimentation in software engineering. 2012: Springer Science & Business Media.

Appendix A. Survey questions' metrics

This Appendix provides the tables with the whole list of metrics used in the survey.

Table A.1. Defined metrics for the goal G1.

Question	Metric
RQ1.1	**M111 Software components modelled:** [classical only; both; quantum only; none]
	M112 Modeling languages and diagram types employed: [UML [class diagrams, activity diagrams, use case diagram, state machine diagram, sequence diagram, component diagram, deployment diagram, other]; SYS-ML; Petri-Nets; entity-relation diagrams; flowcharts; BPMN; Archimate; workflow; dataflow; state machine; informal sketches/models; TOSCA; other]
	M113 Purpose of the diagram or informal sketch: [conceptualization/drafting; communication with manager/customer/other teams; documentation; generation of code; model-driven engineering; other]
	M114 Phase of the quantum software lifecycle phase where models were used: [requirement analysis; architecture & design; implementation; testing; deployment; observability; analysis]
	M115 Criteria for selecting a modeling language: [availability and quality of documentation; active community; compliance; personal preference; tooling support; usefulness; other]
	M116 Tools employed for drawing and/or visualization: [IDE integrated solution; circuit composer; general drawing tool (ppt, drawio); specialized tools for modeling; other]
RQ1.2	**M121 Approach followed for modeling an informal sketch:** [lines and boxes; quantum circuits sketches; mathematical notations; UI mockup sketches; none; other]
	M122 Reasons for employing informal sketches (over modeling language): [lack of formal modeling language; no desire to use modeling language; internal format for sketches; sketches are faster; was drawn on the spot; other]
RQ1.3	**M131 Satisfaction degree regarding the modeling languages employed:** [Likert scale: I can model everything with existing languages; there is no modeling language fitting my needs]
	M132 Phase of the development lifecycle where there is a lack of modeling languages: [requirement analysis; architecture & design; implementation; testing; deployment; observability; analysis; all]
	M133 Purposes where there is a lack of modeling languages: [purpose: modeling quantum circuits; modeling hybrid programs; modeling classical parts; other]

Table A.2. Defined metrics for the goal G2.

Question	Metric
RQ2.1	**M221 Size of the organization:** [micro (less than 10); medium (less than 50); medium-large (less than 250); large (more than 250)]
	M222 Market sector: [public; private; mixed; unknown]
	M223 Application area of the quantum software: [chemistry; commerce; consumer goods; education; energy; financial services; government; health; IT; logistic; research; telecommunication; other]
	M224 Motivation for developing quantum software: [better performance; quality of results; to be future ready; cheaper; research; other]
RQ2.2	**M221 Quantum programming language/toolkit/library employed:** [Alibaba Cloud Quantum Development Platform (ACQDP); Braket; Cirq; D-Wave Ocean; Forest SDK; Q#; OpenQASM2/3; Qiskit; Pennylane; Pytket; Quantum Computation Language (QCL); Quantum Machine Learning (QML); QuantumPath; Silq; Strawberry Fields; other]
	M222 Reasons for selecting this quantum programming language (functional requirements): [availability of compilers/runtime; availability of algorithms implemented; compatibility with other programming languages; compatibility with QC resource; Integration with execution environment; language features (OOP, functional programming, threads or processes, Gate-Based QC, Adiabatic QC); tooling support; transpiling capabilities; other]
	M223 Reasons for using this quantum programming language (non-functional requirements): [active community; compilation optimization features; cost; expressiveness of the problem (fit to the problem type); scalability and performance; learning curve; maintainability; portability; productivity (previous experience with similar languages); quality of documentation; supporting tools (IDE, linter, formatter, libraries); traceability (previous experience with similar languages); usability; other]
	M224 Classical programming languages employed in hybrid application system: [C; C++; C#; Java; JavaScript; Python; Rust; none; other]
	M225 Parts of SDKs employed: [quantum gates/circuit definitions; pre-implemented algorithms; cloud execution; local execution; compiler/transpiler; other]
RQ2.3	**M231 Architecture of the hybrid application system:** [monolith; service-oriented; multilayered; event-driven; REST API; other]
	M232 Integration style among classical and quantum components: [hard-coded endpoint calls / RPC; file transfer; shared database; messaging; none/manual; other]
RQ2.4	**M242 Software parts tested:** [quantum software and classical software; just quantum software; just classical software; none; other]
	M243 Employed quantum testing environments/tools/frameworks: [QuTAF; xyz; other]
	M244 Testing techniques followed: [black-box (equivalence partitioning, boundary value analysis, decision table testing, state transition testing, error guessing); white-box (statement coverage, decision coverage, branch coverage, condition coverage, multiple condition coverage, finite state machine coverage, path coverage, mutation testing, other)]

Table A.3. Defined metrics for the goal G3.

Question	**Metric**
RQ3.1	**M311 Used execution environment:** [cloud QPU; hybrid runtime; cloud simulator; local simulator; private QPU; other]
	M312 Reasons for selecting this execution environment (functional requirements): [availability of algorithms (hybrid programs); usable QPUs; execution time; accessibility of the execution environment; other]
	M313 Reasons for selecting this execution environment (non-functional requirements): [service availability; costs; quality of documentation/tutorials; traceability; usability; other]
	M314 Reasons for a move from simulation environment to an execution environment that uses real quantum computers (QPU): [tests passed correctly; simulations results are valid; affordable execution on quantum computers; unavailability of simulate the quantum software; mandatory requirements; other]
RQ3.2	**M321 Used Workflow Technologies:** [Orquestra; BPMN Engines (Camunda); Covalent; QPath; other]
	M322 Reasons for selecting this workflow technology (functional requirements): [availability of pre-implemented workflows/algorithms; features; compatibility with programming language; other]
	M323 Reasons for selecting this workflow technology (non-functional requirements): [service availability; costs; quality of documentation/tutorials; traceability; extensibility; other]
RQ3.3	**M331 Employed standalone language converter (translator):** [Pytket; Pennylane; Quantum Programming Studio; other]
	M332 Reasons for selecting this language converter (functional requirements): [language support; optimization features; functionality; other]
	M333 Reasons for selecting this language converter (non-functional requirements): [costs; ease of use; quality of documentation/tutorials; performance; other]