On Decision Support for Quantum Application Developers: Categorization, Comparison, and Analysis of Existing Technologies

Daniel Vietz[0000−0003−1366−5805], Johanna Barzen[0000−0001−8397−7973], Frank Leymann[0000−0002−9123−259X], and Karoline Wild[0000−0001−7803−6386]

Institute of Architecture of Application Systems, University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
`[firstname.lastname]@iaas.uni-stuttgart.de`

Abstract. Quantum computers have been significantly advanced in recent years. Offered as cloud services, quantum computers have become accessible to a broad range of users. Along with the physical advances, the landscape of technologies supporting quantum application development has also grown rapidly in recent years. However, there is a variety of tools, services, and techniques available for the development of quantum applications, and which ones are best suited for a particular use case depends, among other things, on the quantum algorithm and quantum hardware. Thus, their selection is a manual and cumbersome process. To tackle this challenge, we introduce a categorization and a taxonomy of available tools, services, and techniques for quantum application development to enable their analysis and comparison. Based on that we further present a comparison framework to support quantum application developers in their decision for certain technologies.

Keywords: Quantum Software Development · Quantum Computing Technologies · Quantum Cloud Services · Decision Support

1 Introduction

Quantum computing promises to solve many problems more efficiently or precisely than it is possible with classical computers. In recent years, the number of quantum hardware vendors, such as IBM, Rigetti, or D-Wave has steadily increased. Via the cloud, quantum computing capacities are made publicly available. Not only hardware vendors offer various quantum cloud services, e.g., IBM via IBM Quantum Experience (IBMQ) [25], but also established cloud providers such as Amazon Web Services (AWS) have added quantum cloud services that facilitate executing quantum algorithms on quantum hardware to their portfolio.

Typically, a quantum application comprises not only the implementation of a quantum algorithm, but also pre- and post-processing components [32]. The development of quantum applications, however, differs significantly from classical application development [52] and currently depends heavily on the used hardware. Due to the growing number of hardware vendors, service providers, and constantly improving quantum hardware, the software landscape for the development and execution of quantum applications is also growing steadily: Almost each quantum cloud provider offers a Software Development Kit (SDK) in order to compile and run quantum applications on their corresponding hardware, such as Qiskit [1] for IBMQ [25] and Ocean [13] for D-Wave Leap [14]. But there are also vendor-agnostic SDKs, such as XACC [33] and ProjectQ [51] that are able to connect to quantum cloud services of different providers. In addition, there are libraries, such as Pennylane [8], that not only enable the connection to a variety of providers but also offer specific algorithms for certain problem classes, e.g., machine learning. For the implementation of a quantum algorithm, different programming languages can also be used, for which specific compilers and transpilers are required. Finally, for integrating pre- and post-processing components with quantum algorithms, orchestration tools, such as Orquestra [57], or extensions for existing workflow languages, such as QuantMe [53], are proposed.

Thus, a variety of tools, services, and techniques is available that can be used for developing quantum applications. However, which of these fit best for a certain use case and how they can be combined depends on (i) the quantum cloud service provider, (ii) the quantum hardware used for the execution, and (iii) the implemented quantum algorithm itself. Furthermore, developer preferences and capabilities, such as the programming language and available tutorials, also play an important role in the decision which tools to use. Due to the variety of possibilities and a missing overview and characterization, it is difficult to compare individual tools and services. The decision for certain tools and services is complex since it requires a lot of knowledge and understanding of their implemented concepts. Therefore, identifying suitable software tools for realizing a certain use case is a manual and cumbersome process.

To tackle these issues, this paper introduces (i) a categorization of currently available technologies and provides (ii) a taxonomy for quantum application development. Based on the proposed categorization and taxonomy, we further introduce (iii) a comparison framework that enables to identify and compare different tools, services, and techniques and, thus, provides decision support to a certain degree. For this, we analyzed various technologies and literature and experimented with several tools and services. The categorization gives an overview of different kinds of technologies and identifies the different building blocks of current quantum application development. The taxonomy further provides a broad view on the different aspects that need to be considered in quantum application development. It enables to understand, analyze, and compare different tools, services, and techniques. We also describe the dependencies and relationships of the different categories in the comparison framework to identify interoperabilities of different tools and services.

After having covered the basic principles and related work in Section 2, Section 3 provides a detailed problem statement. Section 4 introduces the categorization, Section 5 the taxonomy, and Section 6 the prototypical comparison framework developed in the context of this work. Finally, Section 7 gives a conclusion and an outlook on future work.

2 Fundamentals and Related Work

The development of quantum applications differs significantly from the development of classical applications [52]. A quantum application typically contains the implementation of a quantum algorithm, pre- and post-processing components, and additional glue code for the execution of the quantum algorithm on a quantum computer. The development of quantum applications is supported by a wide range of different technologies. However, which tools, services, and techniques shall be used for a particular quantum application depends on several factors.

First, it depends on the used quantum hardware. On the one hand, SDKs that enable the implementation and execution of quantum applications are often tailored to the quantum computers of certain vendors and thereby limit the execution on the respective hardware. E.g., Qiskit [1] (IBM), Strawberry Fields [29] (Xanadu), and Ocean [13] (D-Wave) are each designed for their own hardware and by default do not allow quantum applications to run on other hardware. On the other hand, the physical limitations of current quantum hardware play a central role when implementing quantum algorithms. Today's quantum computers are "noisy", i.e., the computational results are not completely accurate, and their size is of "intermediate scale". Thus, they are called Noisy Intermediate Scale Quantum (NISQ) computers [43]. Selecting the best quantum computer for a specific use case is an important task in the current NISQ-era [47] and some approaches, such as TriQ [39] and t$|$ket$\rangle$ [48], offer compilers with hardware-specific optimization in order to use available hardware in the best possible way.

Available libraries are also important when selecting specific SDKs since they come with pre-implemented algorithms that can be adapted to custom use cases, such as Pennylane [8] provides different libraries in the area of machine learning. On top of that, technologies for the integration with classical applications are becoming increasingly important since hybrid applications are emerging as most promising. A hybrid quantum-classical application comprises quantum components as well as classical components. For the integration of these components, orchestration approaches, such as Orquestra [57] and QuantMe [53] can be used.

In different papers certain aspects of the technology landscape of quantum application development have already been analyzed. Hassija et al. [22], for example, describe the overall landscape of quantum computing, identify the key players, and compare their technologies. LaRose [30] compare different SDKs in terms of their requirements, syntax, library support, and simulation abilities. Quantum programming languages have also been studied in terms of paradigms and features (e.g., [18] and [23]). Fingerhuth et al. [16] identify available open-source quantum software projects and their accompanying website[1] lists many available technologies. There are also other websites that list various tools and cloud services [45]. However, none of them provides an insight on the dependencies and interrelationships, nor does any of them provide decision support. Gill et al. [20] provide a taxonomy in the area of quantum computing, however, it focuses on the algorithmic characteristics of tools and libraries.

[1] https://qosf.org

The existing papers and websites are a first step to compare available tools, services, and techniques for the development of quantum applications. However, so far only certain aspects have been considered and the decision support for quantum application developers is not focused by any work known to us. Nevertheless, the importance of comparison and decision support frameworks has already proven in several other domains, such as for service provider selection [5,15], and deployment automation technologies [56].

3 Problem Statement

As shown in the previous section, a variety of different tools, services, and techniques exist for the development of quantum applications. The decision on which SDK and which libraries to use must be made early in the development phase — both for the implementation of classical and quantum applications. For the development of quantum applications, however, this decision restricts very early on which quantum hardware and which additional libraries can be used. Hence the portability of quantum applications is currently very limited. Therefore, the first research question (RQ) is as follows:

RQ 1: *What types of tools, services, and techniques enable the development and execution of quantum applications and how can they be categorized in order to understand, analyze, and compare them?*

Although a detailed categorization and characterization of available tools, services, and techniques is a good basis for the analysis and comparison of technologies, quantum application developers must be supported in their decision for specific tools, services, and techniques. Therefore, the second RQ is as follows:

RQ 2: *How can quantum application developers be supported in the decision for certain tools, services, and techniques?*

To address the introduced RQs, we investigated several technologies and derived a categorization and taxonomy for quantum application development as well as a comparison framework, all introduced in the following sections.

4 Overview and Categorization of Existing Technologies for Quantum Application Development

Based on a systematic literature study on concepts, technologies, and best practices for integrating quantum applications with classical applications (including ACM Digital Library, arXiv, IEEE XPlore, Science Direct, Springer Link, and Wiley Online Library) and a review of related websites ([16,44,45]), we have identified various technologies for the development of quantum applications.

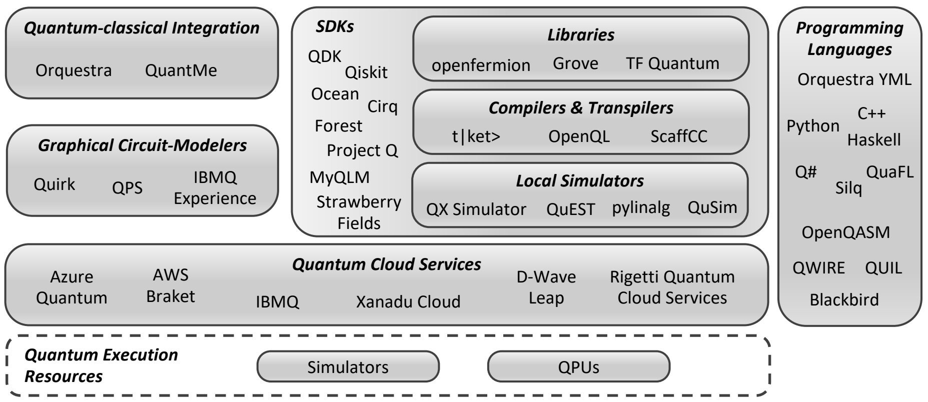


Fig. 1. Categories of quantum application development technologies with examples.

Fig. 1 shows the categorization of existing quantum application development technologies derived from the analysis that we describe in the following. At the bottom of Fig. 1 the category *Quantum Execution Resources* is shown that includes *Quantum Processing Units (QPUs)* as well as *Simulators*. Access to these *Quantum Execution Resources* is typically provided via the cloud by *Quantum Cloud Services* which are not limited to hardware access, e.g., *Graphical Circuit-Modelers* are also often provided as services.

Graphical Circuit-Modelers (left in Fig. 1) enable to model a sequence of operations (called gates) to be applied to the specified qubits. *SDKs*, which are either provided by quantum cloud service providers or by third-party providers, offer advanced developer tools. They can include *Libraries* that provide implementations of algorithms from different areas, such as chemistry [34], cryptography [41], and machine learning [10]. Furthermore, *SDKs* contain *Compilers and Transpilers*, such as the quilc compiler [50] as part of the Forest SDK [46]. Finally, *SDKs* often include a *Local Simulator* to simulate the execution locally, e.g., MyQLM [7] includes the pylinalg [6] simulator. However, Libraries, Compilers, Transpilers, and Simulators are not necessarily part of an SDK but can also be available as standalone technologies, such as t|ket⟩ [48] and ScaffCC [27].

In order to integrate quantum applications with classical applications, there are further tools, such as Orquestra [57], allowing to model the control and data flow between the different components required for pre- and post-processing as well as the execution of the quantum algorithm. This forms the workflow that orchestrates classical and quantum application components.

Finally, the programming languages used for implementing quantum applications are considered as a separate category (on the right of Fig. 1). Since the different tools and services, such as the SDKs, Compilers, Transpilers, and Quantum Cloud Services, support different languages, the language plays an important role when considering the compatibilities between different tools and services. In the next section the different characteristics are discussed in detail.

5 Quantum Application Development Taxonomy

In the previous section we identified categories of current technologies. Based on our literature study, related websites and experiments with various services and tools, we introduce the taxonomy shown in Fig. 2 to enable a systematic analysis of tools, services, and techniques in quantum application development. The taxonomy identifies six main aspects that have to be considered when developing quantum applications: *Quantum Cloud Services*, *Quantum Execution Resources*, *Compilation & Transpilation*, *Knowledge Reuse*, *Programming Languages*, and *Quantum-Classical Integration*.

Each aspect is either divided into multiple sub-aspects or described by its possible values. These values are the lowest refinement considered in our analysis. We abstract from further refinements since we aim to provide a broad overview of current quantum application development. The following subsections describe each aspect in detail.

5.1 Quantum Cloud Services

The *Quantum Cloud Services* aspect identifies at which layer services are available and how these services can be accessed, therefore, the two sub-aspects *Service Model* and *Access Methods* are considered.

In general, three different kinds of *Service Models* can be distinguished, namely *Infrastructure as a Service (IaaS)*, *Platform as a Service (PaaS)*, and *Software as a Service (SaaS)* [36]. *IaaS* offerings provide access to processing, storage, networks, and other fundamental computation resources. Quantum cloud services, such as AWS Braket [3], IBMQ [25], and D-Wave Leap [14], offer quantum computation capabilities as a service and can be assigned to IaaS. *PaaS* offerings provide an application hosting environment to host applications developed using certain programming languages, libraries, services, and tools. A popular platform technology in quantum computing often hosted as a service is Jupyter Notebook, e.g., offered as a service by IBM [25]. Jupyter Notebooks combine source code, console instructions, and documentation in a single document-styled format and, thus, are often used for tutorials. *SaaS* offerings provide users a fully managed software. For example, graphical circuit modelers, such as Quirk [19], belong to this category.

In order to access a cloud service, providers offer different *Access Methods*: *SDKs*, are a typical way by which quantum cloud service providers offer access to their services. For example, Qiskit [1] and Forest [46] offer the ability to execute written source code on the respective quantum cloud services. Furthermore, access can be provided via *Web Services*. For example, IBMQ [25] provides a REST-API [35]. Another way to access quantum cloud services is via *Graphical User Interfaces (GUIs)*. Finally, *Command-Line Interfaces (CLIs)*, such as AWS CLI [4] can also be used to access quantum cloud services.

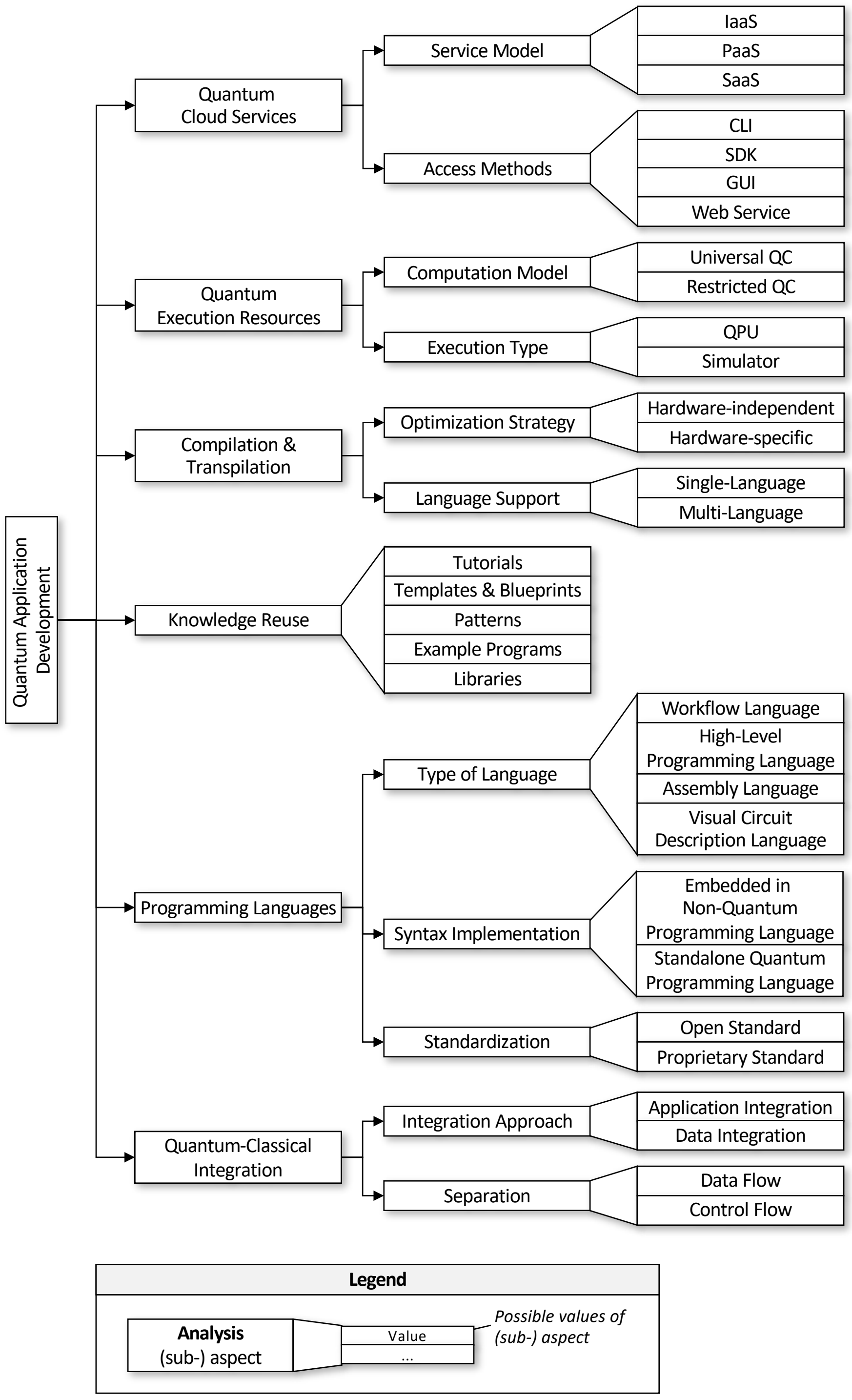


Fig. 2. Taxonomy of Quantum Application Development. Notation is based on [54].

5.2 Quantum Execution Resources

The *Quantum Execution Resources* aspect considers the characteristics of available quantum execution resources for the execution of quantum algorithms. For this, the provided *Execution Type* and the implemented *Computation Model* are considered. Since quantum applications currently depend heavily on the *Quantum Execution Resources*, the available resources must be considered by quantum application developers early in the development phase.

The *Computation Model* defines how quantum computation is modeled and executed. Since it influences the programming style and can further limit the classes of problems that can be efficiently computed with the respective quantum computer, developers must select the appropriate computation model at a very early stage. In general, two kinds of computational models are distinguished: *Universal Quantum Computation* and *Restricted Quantum Computation*. For example, a *Restricted Quantum Computation* model is implemented by the QPUs of D-Wave that support *Stoquastic Adiabatic Quantum Computing*. A well-known example for *Universal Quantum Computation* is the circuit-model (a.k.a. gate-based model). A detailed view on computation models is given by Miszczak [38].

For the execution of quantum algorithms, two general *Execution Types* are available: *QPUs* and *Simulators*. *QPUs* represent physical hardware that is able to compute quantum programs. *Simulators* are an alternative as they simulate quantum programs on classical hardware. Since QPUs are still limited, simulators are essential for developers to develop and test quantum applications before migrating them to real hardware at some point. The ease of switching between execution types is critical to this migration.

5.3 Compilation and Transpilation

The *Compilation and Transpilation* aspect considers characteristics of compilers as well as transpilers. *Compilers* compile source code to lower-level languages, whereas *Transpilers* transpile on the same language level. The taxonomy in Fig. 2 comprises the two sub-aspects *Optimization Strategies* and *Language Support*.

Compilers and transpilers can have multiple *Optimization Strategies* implemented each of which is either *Hardware-specific* or *Hardware-independent*. Häner et al. [24], e.g., describe an end-to-end approach having both hardware-independent and -specific compilation steps implemented. A *Hardware-specific* optimization strategy uses properties and information of a concrete *Quantum Execution Resource*. This is needed because, for example, different qubit connectivity and gate sets of the QPUs have to be considered [32]. A *Hardware-independent* optimization on the other hand provides general optimizations, for example, to rearrange, combine, or remove operations.

Although, theoretically, language functionalities could be abstracted from, the dependency on the language is still high and *Language Support* is an important aspect. Some compilers and transpilers are only built for exactly one input and one output language, such as quilc [50] and ScaffCC [27], and others can consume various different languages or produce various output languages, such as t|ket⟩ [48] and XACC [33].

5.4 Knowledge Reuse

Since the implementation of quantum algorithms from scratch requires a lot of knowledge, there are different ways for *Knowledge Reuse*. A rather abstract way is given by *Tutorials* that can be offered via websites, documents, or packed within a software. *Patterns*, such as proposed by Leymann [31] and Weigold et al. [55], provide an abstract view on common problems and their solutions [2]. Thus, they offer technology-independent knowledge to specific problems. Furthermore, *Templates and Blueprints* are scaffold implementations that provide a basic structure for further source code. *Example Programs*, which are, e.g., provided as Jupyter notebooks, allow insights into the implementation of other applications and can be adapted to own custom use cases. Finally, concrete implementations of algorithms can also be provided as *Libraries* which can be used as subroutines in a quantum application [10,34,41].

5.5 Programming Languages

The *Programming Language* aspect considers the characteristics of available programming languages for quantum applications. For this, the *Type of Language*, the *Syntax Implementation*, and *Standardization* are considered.

Currently, four types of programming languages can be distinguished: *Workflow Languages*, *High-level Programming Languages*, *Assembly Languages*, and *Graphical Circuit Description Languages*. *Assembly Languages*, such as OpenQASM [12], eQASM [17], QWIRE [42], and Quil [49], are low level and provide a textual representation of every operation the quantum computer is to perform. *High-level Programming Languages*, such as Quipper [21] and Q# [37], are machine independent and provide high-level language features, such as loops and recursion. *Workflow Languages*, such as offered in Orquestra [57], allow to model the control flow of (hybrid quantum-classical) applications. *Graphical Circuit Description Languages* provide graphical representations of quantum circuits.

For the *Syntax Implementation* we distinguish between two cases. On the one hand, there are programming languages that have their own independent syntax, such as Silq [9] and Scaffold [26] Thus, they are *Standalone Quantum Programming Languages*. On the other hand, a programming language can also be embedded into another programming language, for example, Quipper [21] is embedded in Haskell, and Qiskit [1], PyQuil [28], and Cirq [11] offer programming languages embedded in Python.

Standardization is important for the interoperability of different tools and services and is considered in the last aspect. *Open Standards* (a.k.a. *de-jure standards*) are developed or adopted by standards organizations. For example, BPMN [40] is an Open Standard of the Object Management Group. *Proprietary Standards* (a.k.a. *de-facto standards*) evolve from vendor-specific solutions and are widely accepted by a broad base of users. Since OpenQASM [12] is supported by many tools and services, including Qiskit [1], t|ket⟩ [48], XACC [33], Project Q [51], and Cirq [11], it has a wide acceptance and can be considered as de-facto standard for describing circuits. However, most of current programming languages for quantum application development do not implement any standard.

5.6 Quantum-Classical Integration

Hybrid applications that integrate quantum and classical components are increasingly appearing as the most promising solutions at present and in the near future. Thus, the last aspect of Fig. 2 considers *Quantum-Classical Integration*.

In general, two *Integration Approaches* can be distinguished. On the one hand, *Data Integration* consolidates data from different sources to provide applications with a uniform access to this data. On the other hand, *Application Integration* integrates different applications on a functional level.

Furthermore, we characterize integration approaches by their ability to separate business functions from *Data Flow* and *Control Flow*. Technologies, such as Orquestra [57] and QuantMe [53], enable business functions to be defined separately from data- and control flow. With SDKs, such as Qiskit [1], integration must be implemented in the source code and, therefore, does not enable separation of business functions and control flow. In general, *Quantum-Classical Integration* is still largely neglected, although it is essential for the realization of future applications.

6 Comparison Framework

In the previous sections we have introduced a categorization (Section 4) and taxonomy (Section 5) to characterize existing tools, services, and techniques for quantum application development. Based on these results, in this section we introduce a comparison framework that supports developers in selecting suitable technologies. The comparison framework enables the comparison of characteristics and dependencies between tools and services of different categories.

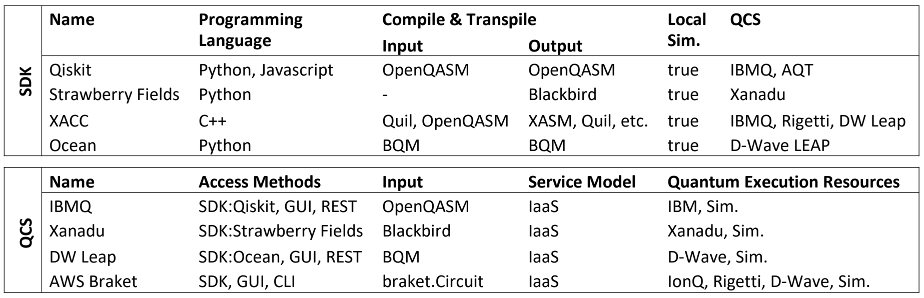
Fig. 3 shows an excerpt of the comparison framework[2] considering exemplary Software Development Kits (SDKs) and Quantum Cloud Services (QCS). An SDK, identified by its name, is available for certain programming languages. Furthermore, since an SDK contains compilers and transpilers that can support various input and output languages, these are separately listed. Besides the availability of a local simulator, which is provided by all examples in Fig. 3, it is finally listed which QCS are directly supported. The category of QCS is another category exemplary sown in Fig. 3. It contains the name, access method, input language, service model, and available quantum execution resources[3]. All further columns of both categories are hidden in Fig. 3 but can be found in the comparison framework.

The comparison framework implements multi-criteria filtering for all attributes of each category. For example, if Python is chosen as programming language and the availability of a local simulator is required, only Qiskit [1], Strawberry Fields [29], and Ocean [13] will be displayed in the example in Fig. 3. Furthermore, the comparison framework enables to identify interoperabilities between different categories via cross-category filtering. When selecting IBMQ [25]

[2] The framework can be found at http://www.github.com/UST-QuAntiL/Qverview

[3] QPUs are grouped by their respective vendor

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77980-1_10



	Name	Programming Language	Compile & Transpile		Local Sim.	QCS
			Input	Output		
SDK	Qiskit	Python, Javascript	OpenQASM	OpenQASM	true	IBMQ, AQT
	Strawberry Fields	Python	-	Blackbird	true	Xanadu
	XACC	C++	Quil, OpenQASM	XASM, Quil, etc.	true	IBMQ, Rigetti, DW Leap
	Ocean	Python	BQM	BQM	true	D-Wave LEAP

	Name	Access Methods	Input	Service Model	Quantum Execution Resources
QCS	IBMQ	SDK:Qiskit, GUI, REST	OpenQASM	IaaS	IBM, Sim.
	Xanadu	SDK:Strawberry Fields	Blackbird	IaaS	Xanadu, Sim.
	DW Leap	SDK:Ocean, GUI, REST	BQM	IaaS	D-Wave, Sim.
	AWS Braket	SDK, GUI, CLI	braket.Circuit	IaaS	IonQ, Rigetti, D-Wave, Sim.

Fig. 3. Excerpt of the Comparison Framework with exemplary SDKs and QCS.

as quantum cloud service, the comparison framework automatically exposes all interoperable SDKs. In addition to Qiskit [1], which is the SDK provided by IBMQ [25], XACC [33], for example, can also be used to execute quantum applications there, as shown in Fig. 3. However, when combined with the programming language filter, only Qiskit would be shown.

The comparison framework supports a multi-criteria cross-category analysis of current technologies for quantum application development. Thus, it (i) provides an overview of technologies of different categories, (ii) enables filtering, comparison and analysis of technologies within each category, and (iii) enables identification of cross-category interoperabilities.

7 Conclusion and Future Work

Currently, there are strong dependencies between the tools and services used to develop quantum applications and the hardware on which they will run. Due to the limited portability, it is important that quantum application developers identify the appropriate tools and services at an early stage. To make a first step towards decision support for quantum application developers, we have introduced in this paper (i) a categorization, (ii) a taxonomy, and (iii) a comparison framework. This is based on an investigation of a variety of technologies and publications to provide guidance for quantum application developers.

With the taxonomy, we have introduced several aspects that need to be considered when selecting specific tools, services, and techniques. While the comparison framework provides guidance in the complex landscape of quantum technologies, it does not yet provide full-automated decision support. In the next step, we therefore plan to further extend our comparison framework to also comprise library support for certain algorithms. This is crucial in order to make a decision based on the problem at hand. The comparison framework so far allows filtering based on the presented taxonomy. In addition, we plan to incorporate weight factors, which will allow categories to be weighted differently.

Acknowledgments

This work was partially funded by the BMWi project *PlanQK (01MK20005N)* as well as the WM BW project *SEQUOIA*.

References

1. Abraham, H., et al.: Qiskit: An Open-source Framework for Quantum Computing (2019). https://doi.org/10.5281/zenodo.2562110
2. Alexander, C., Ishikawa, S., Silverstein, M.: A Pattern Language: Towns, Buildings, Construction. Oxford University Press (1977)
3. Amazon Web Services, Inc: AWS Braket (2021), https://aws.amazon.com/braket
4. Amazon.com, Inc: aws-cli (2020), https://github.com/aws/aws-cli
5. Andrikopoulos, V., Gómez Sáez, S., Leymann, F., Wettinger, J.: Optimal Distribution of Applications in the Cloud. In: Proceedings of the 26th International Conference on Advanced Information Systems Engineering (CAiSE 2014). pp. 75–90. Springer (2014). https://doi.org/10.1007/978-3-319-07881-6_6
6. Atos SE: qat.pylinalg: Python Linear-algebra simulator (2020), https://myqlm.github.io/myqlm_specific/qat-pylinalg.html
7. Atos SE: MyQLM (2021), https://atos.net/en/lp/myqlm
8. Bergholm, V., et al.: PennyLane: Automatic differentiation of hybrid quantum-classical computations (2020), arXiv preprint arXiv:1811.04968
9. Bichsel, B., Baader, M., Gehr, T., Vechev, M.: 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. p. 286–300. PLDI '20, Association for Computing Machinery (2020). https://doi.org/10.1145/3385412.3386007
10. Broughton, M., et al.: TensorFlow Quantum: A Software Framework for Quantum Machine Learning (2020), arXiv preprint arXiv:2003.02989
11. Cirq Developers: Cirq (2021). https://doi.org/10.5281/zenodo.4062499
12. Cross, A.W., Bishop, L.S., Smolin, J.A., Gambetta, J.M.: Open Quantum Assembly Language (2017), arXiv preprint arXiv:1707.03429
13. D-Wave Systems Inc: dwave-ocean-sdk (2021), https://github.com/dwavesystems/dwave-ocean-sdk
14. D-Wave Systems Inc: Leap (2021), https://dwavesys.com/take-leap
15. Farshidi, S., Jansen, S., de Jong, R., Brinkkemper, S.: A Decision Support System for Cloud Service Provider Selection Problem in Software Producing Organizations. In: 2018 IEEE 20th Conference on Business Informatics (CBI). vol. 01, pp. 139–148 (2018). https://doi.org/10.1109/CBI.2018.00024
16. Fingerhuth, M., Babej, T., Wittek, P.: Open source software in quantum computing. PLOS ONE **13**(12) (2018). https://doi.org/10.1371/journal.pone.0208561
17. Fu, X., et al.: eQASM: An Executable Quantum Instruction Set Architecture. In: 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA). pp. 224–237 (2019). https://doi.org/10.1109/HPCA.2019.00040
18. Garhwal, S., Ghorani, M., Ahmad, A.: Quantum Programming Language: A Systematic Review of Research Topic and Top Cited Languages. Archives of Computational Methods in Engineering (2019). https://doi.org/10.1007/s11831-019-09372-6
19. Gidney, C., Marwaha, K., Haugeland, J., ebraminio, Kalra, N.: Quirk: Quantum Circuit Simulator (2021), https://algassert.com/quirk

20. Gill, S.S., et al.: Quantum Computing: A Taxonomy, Systematic Review and Future Directions (2020), arXiv preprint arXiv:2010.15559
21. Green, A.S., Lumsdaine, P.L., Ross, N.J., Selinger, P., Valiron, B.: Quipper: A Scalable Quantum Programming Language. In: Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation. p. 333–342. PLDI '13, Association for Computing Machinery (2013). https://doi.org/10.1145/2491956.2462177
22. Hassija, V., et al.: Present landscape of quantum computing. IET Quantum Communication **1**(2), 42–48 (2020). https://doi.org/10.1049/iet-qtc.2020.0027
23. Heim, B., et al.: Quantum programming languages. Nature Reviews Physics **2**(12), 709–722 (2020). https://doi.org/10.1038/s42254-020-00245-7
24. Häner, T., Steiger, D.S., Svore, K., Troyer, M.: A software methodology for compiling quantum programs. Quantum Science and Technology **3**(2), 020501 (2018). https://doi.org/10.1088/2058-9565/aaa5cc
25. IBM: IBM Quantum Experience (2021), https://quantum-computing.ibm.com
26. Javadi-Abhari, A., et al.: Scaffold: Quantum Programming Language. Tech. rep., Princeton Univ NJ Dept of Computer Science (2012)
27. Javadi-Abhari, A., et al.: ScaffCC: A Framework for Compilation and Analysis of Quantum Computing Programs. In: Proceedings of the 11th ACM Conference on Computing Frontiers. CF '14, Association for Computing Machinery (2014). https://doi.org/10.1145/2597917.2597939
28. Karalekas, P.J., et al.: PyQuil: Quantum programming in Python (2020). https://doi.org/10.5281/zenodo.3631770
29. Killoran, N., et al.: Strawberry Fields: A Software Platform for Photonic Quantum Computing. Quantum **3**, 129 (2019). https://doi.org/10.22331/q-2019-03-11-129
30. LaRose, R.: Overview and Comparison of Gate Level Quantum Software Platforms. Quantum **3**, 130 (2019). https://doi.org/10.22331/q-2019-03-25-130
31. Leymann, F.: Towards a Pattern Language for Quantum Algorithms. In: First International Workshop, QTOP 2019, Munich, Germany, March 18, 2019, Proceedings. Springer (2019). https://doi.org/10.1007/978-3-030-14082-3_19
32. Leymann, F., Barzen, J.: The bitter truth about gate-based quantum algorithms in the NISQ era. Quantum Science and Technology pp. 1–28 (2020), https://doi.org/10.1088/2058-9565/abae7d
33. McCaskey, A.J., Lyakh, D.I., Dumitrescu, E.F., Powers, S.S., Humble, T.S.: XACC: a system-level software infrastructure for heterogeneous quantum–classical computing. Quantum Science and Technology **5**(2), 024002 (2020). https://doi.org/10.1088/2058-9565/ab6bf6
34. McClean, J.R., et al.: OpenFermion: The Electronic Structure Package for Quantum Computers (2019), arXiv preprint arXiv:1710.07629
35. McKay, D.C., et al.: Qiskit Backend Specifications for OpenQASM and OpenPulse Experiments (2018), arXiv preprint arXiv:1809.03452
36. Mell, P., Grance, T.: The NIST definition of cloud computing. Tech. Rep. NIST SP 800-145, National Institute of Standards and Technology (2011). https://doi.org/10.6028/NIST.SP.800-145
37. Microsoft: Q# Language (2021), https://github.com/microsoft/qsharp-language
38. Miszczak, J.A.: Models of quantum computation and quantum programming languages. Bulletin of the Polish Academy of Sciences: Technical Sciences **59**(3), 305–324 (2011). https://doi.org/10.2478/v10175-011-0039-5
39. Murali, P., Baker, J.M., Javadi-Abhari, A., Chong, F.T., Martonosi, M.: Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers.

In: Proc. of the 24th Int. Conf. on Architectural Support for Programming Languages and Operating Systems. p. 1015–1029. ASPLOS '19, Association for Computing Machinery (2019). https://doi.org/10.1145/3297858.3304075
40. OMG: Business Process Model and Notation (BPMN) Version 2.0. Object Management Group (OMG) (2011)
41. Open Quantum Safe Project: liboqs (2021), https://openquantumsafe.org/liboqs/
42. Paykin, J., Rand, R., Zdancewic, S.: QWIRE: A Core Language for Quantum Circuits. In: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages. p. 846–858. POPL 2017, Association for Computing Machinery (2017). https://doi.org/10.1145/3009837.3009894
43. Preskill, J.: Quantum Computing in the NISQ era and beyond. Quantum **2**, 79 (2018). https://doi.org/10.22331/q-2018-08-06-79
44. Quantiki: QC simulators (2021), https://quantiki.org/wiki/list-qc-simulators
45. Quantum Computing Report: Tools (2021), https://quantumcomputingreport.com/tools/
46. Rigetti Computing: Forest SDK (2019), https://pyquil-docs.rigetti.com/
47. Salm, M., Barzen, J., Breitenbücher, U., Leymann, F., Weder, B., Wild, K.: The NISQ Analyzer: Automating the Selection of Quantum Computers for Quantum Algorithms. In: Proceedings of the 14th Symposium and Summer School on Service-Oriented Computing (SummerSOC 2020). pp. 66–85. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-64846-6_5
48. Sivarajah, S., Dilkes, S., Cowtan, A., Simmons, W., Edgington, A., Duncan, R.: t$|$ket$\rangle$: a retargetable compiler for NISQ devices. Quantum Science and Technology **6**(1), 014003 (2020). https://doi.org/10.1088/2058-9565/ab8e92
49. Smith, R.S., Curtis, M.J., Zeng, W.J.: A Practical Quantum Instruction Set Architecture (2017), arXiv preprint arXiv:1608.03355
50. Smith, R.S., Peterson, E.C., Davis, E.J., Skilbeck, M.G.: quilc: An Optimizing Quil Compiler (2020). https://doi.org/10.5281/zenodo.3677537
51. Steiger, D.S., Häner, T., Troyer, M.: ProjectQ: an open source software framework for quantum computing. Quantum **2**, 49 (2018). https://doi.org/10.22331/q-2018-01-31-49
52. Weder, B., Barzen, J., Leymann, F., Salm, M., Vietz, D.: The Quantum Software Lifecycle. In: Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software (APEQS 2020). pp. 2–9. ACM (2020). https://doi.org/10.1145/3412451.3428497
53. Weder, B., Breitenbücher, U., Leymann, F., Wild, K.: Integrating Quantum Computing into Workflow Modeling and Execution. In: Proceedings of the 13th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2020). pp. 279–291. IEEE Computer Society (2020). https://doi.org/10.1109/UCC48980.2020.00046
54. Weerasiri, D., Barukh, M.C., Benatallah, B., Sheng, Q.Z., Ranjan, R.: A Taxonomy and Survey of Cloud Resource Orchestration Techniques. ACM Comput. Surv. **50**(2) (2017). https://doi.org/10.1145/3054177
55. Weigold, M., Barzen, J., Salm, M., Leymann, F.: Data encoding patterns for quantum computing. In: Proceedings of the 27th Conference on Pattern Languages of Programs. The Hillside Group (2021), accepted for publication
56. Wurster, M., et al.: The Essential Deployment Metamodel: A Systematic Review of Deployment Automation Technologies. SICS Software-Intensive Cyber-Physical Systems **35**, 63–75 (2019). https://doi.org/10.1007/s00450-019-00412-x
57. Zapata Computing: Orquestra (2021), https://zapatacomputing.com/orquestra/

All links were lastly followed on March 31, 2021.