IAES Journal of Electrical and Electronics Engineering
Vol. 12, No. 2, December 2025, pp. 35~40
ISSN: 2338-7432, DOI: 10.11591/jeee.v12i2.pp35-40

Quantum programming languages: a review of programming paradigms, software frameworks, and emerging applications

Laksana Talenta Ahmad
Embedded Systems and Power Electronics Research Group, Yogyakarta, Indonesia

Article Info

Article history:

Received Oct 9, 2023
Revised Sep 30, 2024
Accepted Jan 28, 2025

Keywords:

Hybrid quantum–classical computing
Quantum algorithms
Quantum circuits
Quantum programming languages
Quantum software engineering
Quantum software frameworks

ABSTRACT

Quantum programming languages and software frameworks provide the essential interface between quantum algorithms, classical computing environments, and physical quantum processors. Their development has accelerated as quantum computing has moved from theoretical investigation toward accessible cloud-based experimentation and near-term applications. This review examines the evolution and characteristics of four prominent software ecosystems: Qiskit, Cirq, Q#, and PennyLane. It discusses major quantum programming paradigms, including circuit-based, imperative, functional, and hybrid quantum–classical approaches, together with their implications for algorithm development and software abstraction. The review also considers representative quantum algorithms, circuit construction and optimization, and applications in quantum chemistry, combinatorial optimization, and machine learning. Particular attention is given to interoperability, programming-language standardization, classical simulation, testing, and software engineering challenges. The literature indicates that existing frameworks offer complementary capabilities rather than a universally superior programming environment. Nevertheless, fragmented interfaces, hardware-dependent compilation, simulation costs, and immature testing practices continue to restrict portability and reproducibility. Progress therefore depends on stronger intermediate representations, standardized development workflows, scalable simulation, and reliable integration between classical and quantum resources.

This is an open access article under the CC BY-SA license.

Corresponding Author:

Laksana Talenta Ahmad
Embedded Systems and Power Electronics Research Group
Yogyakarta, Indonesia
Email: laksanatadz@gmail.com

1. INTRODUCTION

Quantum computing introduces a fundamentally different approach to information processing by exploiting superposition, entanglement, and quantum measurement. These physical properties enable computational methods that differ substantially from conventional algorithms executed on classical processors. However, translating quantum principles into executable programs requires specialized languages, compilers, simulators, and development environments. The quality of these software components directly affects how effectively researchers can design, validate, and execute quantum algorithms [1], [2].

The development of quantum programming environments has followed several complementary directions. Early research concentrated on formal language definitions, type systems, and mathematical models for expressing quantum operations safely [3], [4]. More recent development has produced practical software ecosystems that connect algorithm design with simulators and accessible quantum hardware.

Consequently, programming frameworks have become important components of the broader quantum computing infrastructure [5], [6].

Despite this progress, the quantum software landscape remains fragmented. Different frameworks employ different abstractions, circuit representations, programming interfaces, and hardware integration mechanisms. This diversity provides flexibility but complicates software portability, collaborative development, and consistent performance evaluation. The central problem addressed in this review is therefore how existing programming paradigms and frameworks can support reliable, interoperable, and scalable quantum software development.

This review evaluates four influential ecosystems—Qiskit, Cirq, Q#, and PennyLane—alongside the broader research on quantum programming paradigms and software engineering. It examines their approaches to quantum algorithms, circuit construction, simulation, and hybrid execution. It also identifies persistent limitations in standardization, verification, and hardware-independent programming. The resulting synthesis highlights research directions that may help establish a more coherent quantum software development environment.

2. QUANTUM PROGRAMMING PARADIGMS

Quantum programming languages have evolved from theoretical models into practical environments for expressing and executing quantum computations. Their design must account for restrictions that do not normally arise in classical programming, particularly the inability to copy arbitrary unknown quantum states and the probabilistic nature of measurement. Quantum programs must also manage the relationship between quantum data and classical control instructions. These requirements have encouraged the development of specialized programming paradigms and type systems [1], [3], [4].

The circuit-based paradigm is currently one of the most widely used approaches. Programs are represented as sequences of quantum gates applied to qubits, followed by measurements that produce classical outcomes. This representation is intuitive for hardware execution and provides a direct connection between algorithms and physical quantum operations. However, circuit-level programming can become difficult to maintain when algorithms involve complex control structures, repeated subroutines, or large numbers of operations [7], [8].

Imperative and domain-specific paradigms provide additional abstraction through familiar programming constructs. They allow developers to express operations, procedures, variables, and control flow without manually specifying every low-level hardware instruction. Functional approaches offer another perspective by emphasizing composition, mathematical structure, and controlled manipulation of quantum information. Research on languages such as Q# and Silq demonstrates how type systems and language-level abstractions can reduce programming errors and simplify quantum algorithm development [9], [10].

Hybrid quantum–classical programming represents a particularly important paradigm for contemporary quantum hardware. In this model, classical processors manage data preparation, parameter optimization, and result interpretation, while quantum processors execute selected computational operations. Variational algorithms exemplify this approach because they repeatedly exchange information between a parameterized quantum circuit and a classical optimization routine. This arrangement is especially relevant to noisy intermediate-scale quantum devices, whose limited circuit depth restricts the direct execution of large algorithms [11], [12].

3. DEVELOPMENT OF MAJOR QUANTUM SOFTWARE FRAMEWORKS

Qiskit, Cirq, Q#, and PennyLane represent four influential but technically different approaches to quantum software development. Their differences concern programming abstraction, hardware awareness, algorithm construction, and integration with classical software. Qiskit and Cirq provide extensive circuit-oriented environments, whereas Q# emphasizes a dedicated high-level quantum language. PennyLane places particular emphasis on differentiable quantum programming and machine-learning workflows [5], [6], [13].

Qiskit was developed as an open-source framework for constructing, optimizing, simulating, and executing quantum circuits. Its Python-based interface makes it accessible to developers familiar with scientific computing and conventional software development. Its ecosystem has supported research, education, circuit transpilation, and interaction with quantum computing services. The availability of a large collection of tutorials and community resources has also contributed to its adoption in academic and experimental settings [14], [15].

Cirq was designed with particular attention to the characteristics of noisy intermediate-scale quantum devices. Its programming model allows developers to manipulate circuits while considering device topology, gate constraints, and execution details. This hardware-aware approach is useful when circuit layout

and connectivity strongly influence algorithm performance. Cirq also provides simulation capabilities and tools for experimenting with circuits before executing them on physical processors [16].

Q# takes a different approach by providing a standalone, high-level language specifically designed for quantum algorithm development. Its design incorporates specialized constructs, type checking, and mechanisms for expressing quantum operations and classical control. These features help developers formulate algorithms without making circuit construction the only programming abstraction. Q# therefore illustrates the value of language-level support for expressing complex quantum procedures in a structured manner [9].

PennyLane focuses on differentiable programming and the integration of quantum circuits with classical machine-learning libraries. Its architecture allows quantum operations to participate in optimization workflows using automatic differentiation. This capability is particularly relevant to variational quantum algorithms, quantum neural networks, and quantum machine-learning experiments. Its device abstraction also supports experimentation across different simulators and quantum hardware providers [13].

Table 1 summarizes the principal characteristics of the four frameworks. The comparison indicates that their strengths are complementary rather than interchangeable. Framework selection should therefore depend on the research objective, programming experience, hardware requirements, and intended classical integration. No single framework can be considered universally optimal for every quantum computing application.

Table 1. Comparison of major quantum software frameworks

Framework	Main programming approach	Key strengths	Typical applications
Qiskit	Python-based circuit programming	Circuit construction, transpilation, simulation, hardware access	Quantum algorithms, education, experimentation
Cirq	Python-based, hardware-aware circuits	Device constraints, circuit optimization, NISQ experimentation	Hardware-oriented algorithms and circuit research
Q#	High-level domain-specific language	Structured quantum operations, type system, algorithm abstraction	Quantum algorithm development and resource estimation
PennyLane	Differentiable quantum programming	Automatic differentiation and machine-learning integration	Variational algorithms and quantum machine learning

4. QUANTUM ALGORITHMS AND CIRCUIT DEVELOPMENT

Quantum algorithms provide the computational foundation upon which programming languages and software frameworks operate. Their implementation generally requires the translation of mathematical operations into sequences of gates, measurements, and classical instructions. The circuit model provides a common representation for many algorithms, although the complexity of the resulting circuits varies considerably. Algorithm design must therefore consider both theoretical computational benefits and the physical limitations of available quantum processors [7], [11].

Several established algorithms illustrate the range of quantum programming applications. Grover's search algorithm demonstrates a quadratic reduction in query complexity for unstructured search, while Shor's algorithm provides a theoretical polynomial-time approach to integer factorization. Quantum phase estimation and Hamiltonian simulation are important for scientific computations involving quantum systems. These algorithms also demonstrate why circuit depth, gate accuracy, and qubit requirements must be considered during software development [17]–[19].

Variational quantum algorithms have received particular attention because they combine parameterized quantum circuits with classical optimization. The variational quantum eigensolver is designed to estimate molecular and material properties, while the quantum approximate optimization algorithm targets combinatorial optimization problems. Both approaches repeatedly execute quantum circuits and use measurement results to update classical parameters. Their practical performance depends on circuit design, optimization stability, measurement cost, and hardware noise [11], [12], [20].

Circuit compilation and optimization are essential steps between algorithm design and hardware execution. A high-level algorithm may need to be decomposed into native gates supported by a particular quantum processor. Compilation may also involve qubit mapping, gate cancellation, circuit scheduling, and reduction of circuit depth. These transformations are important because a theoretically efficient algorithm may become impractical when its compiled circuit exceeds the coherence or connectivity limitations of the target hardware [8], [21].

5. INTEGRATION OF QUANTUM AND CLASSICAL COMPUTING

Hybrid computing is currently a practical foundation for many quantum applications. Classical processors perform tasks such as data encoding, parameter optimization, error mitigation, and statistical analysis, while quantum processors execute circuits that exploit quantum effects. This division of responsibilities allows developers to use existing classical computing infrastructure rather than replacing it. It also creates a programming model in which communication between classical and quantum components becomes a central design consideration [11], [12].

The integration mechanisms differ across software frameworks. Qiskit supports workflows that connect circuit execution with classical Python applications and optimization routines. Cirq provides circuit-level control that can be incorporated into broader classical experiments, while Q# supports the expression of classical control alongside quantum operations. PennyLane provides particularly direct integration with machine-learning libraries through differentiable quantum computations [9], [13]–[16].

Nevertheless, hybrid execution introduces several practical difficulties. Communication latency between classical systems and remote quantum processors can increase total execution time, especially when algorithms require many iterations. Repeated measurements also create statistical uncertainty and may require substantial sampling resources. Furthermore, the performance of hybrid algorithms depends on the quality of classical optimization methods and the stability of quantum hardware. These constraints make efficient orchestration and workload distribution important topics for future development [11], [12], [22].

6. STANDARDIZATION, SIMULATION, AND SOFTWARE ENGINEERING CHALLENGES

The absence of complete interoperability remains a major limitation of the quantum software ecosystem. Frameworks often use different circuit representations, gate conventions, hardware interfaces, and execution models. OpenQASM 3 addresses part of this problem by providing a broader assembly language for describing quantum circuits, classical control flow, and timing-related operations. However, a shared representation alone does not eliminate differences in hardware capabilities, compiler behavior, or software libraries. Classical simulation is another essential but challenging aspect of quantum software development. Simulators allow researchers to inspect quantum algorithms, evaluate circuit behavior, and test programs without continuously accessing physical hardware. State-vector simulation, density-matrix methods, and tensor-network approaches provide different trade-offs in accuracy, memory consumption, and computational cost. Their scalability remains limited because representing general quantum states requires resources that can grow exponentially with the number of qubits [23], [24].

Software verification and testing require specialized techniques because quantum programs produce probabilistic outputs. Conventional testing methods based on deterministic input–output comparisons are therefore insufficient for many quantum applications. Researchers have investigated property-based testing, mutation testing, static analysis, and other approaches to detect errors in quantum programs. Nevertheless, the lack of mature testing standards and widely accepted benchmarks continues to complicate the assessment of software quality [25]–[27].

Quantum software engineering must also address maintainability, reproducibility, and lifecycle management. Developers need reliable methods for documenting circuit assumptions, tracking framework versions, validating compiler transformations, and reproducing experimental results. Recent surveys identify software architecture, development processes, and quality assessment as important areas requiring further research. These findings suggest that quantum software must mature beyond isolated algorithm demonstrations toward systematic engineering practices [5], [6], [28].

Table 2 connects the principal software challenges with their consequences and possible research directions. The table emphasizes that the challenges are interdependent rather than isolated technical problems. For example, standardization can improve portability, but reliable portability also requires compatible compilers and hardware execution interfaces. Similarly, better simulation can strengthen testing, although simulation alone cannot reproduce every physical hardware effect.

Table 2. Major challenges and research directions in quantum software development

Challenge	Main consequence	Potential research direction
Framework fragmentation	Limited portability and code reuse	Shared intermediate representations and interoperable APIs
Hardware dependence	Circuit performance varies across devices	Hardware-aware and adaptive compilation
Simulation complexity	Restricted testing of large circuits	Tensor-network and distributed simulation
Probabilistic execution	Difficulties in conventional testing	Statistical testing and quantum-specific verification
Hybrid communication	Increased execution time and overhead	Efficient orchestration and workload allocation
Software lifecycle maturity	Reproducibility and maintenance problems	Standardized development and quality-assurance practices

7. EMERGING APPLICATIONS AND FUTURE RESEARCH DIRECTIONS

Quantum software frameworks are enabling experimentation across scientific and computational disciplines. Quantum chemistry and materials science are important application areas because quantum processors may eventually provide more natural representations of molecular and many-body systems. Variational algorithms and Hamiltonian simulation are among the methods being investigated for these tasks. However, practical advantages remain dependent on improvements in hardware reliability, circuit efficiency, and resource requirements [20], [29].

Quantum machine learning represents another expanding research direction. Frameworks such as PennyLane facilitate the development of parameterized quantum models and their integration with classical learning algorithms. Researchers have explored quantum-enhanced classification, generative models, optimization, and scientific data analysis. Nevertheless, questions concerning trainability, data encoding, measurement overhead, and demonstrable quantum advantage remain unresolved [13], [30].

Future progress will require closer cooperation between language designers, compiler developers, hardware providers, and software engineers. A more interoperable ecosystem should support the movement of quantum programs across frameworks without losing essential semantic or hardware information. Intermediate representations and common interface specifications can help separate algorithm design from device-specific execution. In parallel, improved testing, simulation, and resource estimation will be necessary to establish reliable development practices [5], [31]-[33].

8. CONCLUSION

Quantum programming languages and software frameworks have developed into essential components of the emerging quantum computing ecosystem. Qiskit, Cirq, Q#, and PennyLane provide complementary capabilities for circuit construction, hardware-aware programming, high-level algorithm development, and differentiable quantum computation. Their evolution demonstrates that quantum software requires multiple programming abstractions rather than a single universal approach. At the same time, hybrid quantum–classical computing has become a practical strategy for developing applications on current hardware. The review identifies fragmented standards, simulation limitations, probabilistic testing, and immature software engineering practices as persistent obstacles. These problems affect portability, reproducibility, and the ability to scale quantum applications beyond experimental demonstrations. Addressing them will require interoperable representations, more efficient compilers, scalable simulators, and systematic software quality assurance. Ultimately, the maturity of quantum computing will depend not only on advances in physical processors but also on the development of reliable, accessible, and maintainable programming environments.

REFERENCES

[1] B. Heim *et al.*, "Quantum programming languages," *Nature Reviews Physics*, vol. 2, no. 12, pp. 709–722, 2020, doi: 10.1038/s42254-020-00245-7.

[2] S. J. Gay, "Quantum programming languages: Survey and bibliography," *Mathematical Structures in Computer Science*, vol. 16, no. 4, pp. 581–600, 2006, doi: 10.1017/S0960129506005378.

[3] P. Selinger, "Towards a quantum programming language," *Mathematical Structures in Computer Science*, vol. 14, no. 4, pp. 527–586, 2004, doi: 10.1017/S0960129504004256.

[4] P. Selinger and B. Valiron, "A lambda calculus for quantum computation with classical control," *Mathematical Structures in Computer Science*, vol. 16, no. 3, pp. 527–552, 2006, doi: 10.1017/S0960129506005238.

[5] M. A. Serrano, J. A. Cruz-Lemus, R. Perez-Castillo, and M. Piattini, "Quantum software components and platforms: overview and quality assessment," *ACM Computing Surveys*, vol. 55, no. 8, 2023, doi: 10.1145/3548679.

[6] K. Dwivedi, M. Haghparast, and T. Mikkonen, "Quantum software engineering and quantum software development lifecycle: a survey," *Cluster Computing*, vol. 27, no. 6, pp. 7127–7145, 2024, doi: 10.1007/s10586-024-04362-1.

[7] A. Barenco *et al.*, "Elementary gates for quantum computation," *Physical Review A*, vol. 52, no. 5, pp. 3457–3467, 1995, doi: 10.1103/PhysRevA.52.3457.

[8] A. Cross *et al.*, "OpenQASM 3: A broader and deeper quantum assembly language," *ACM Transactions on Quantum Computing*, vol. 3, no. 3, 2022, doi: 10.1145/3505636.

[9] K. Svore *et al.*, "Q#: Enabling scalable quantum computing and development with a high-level DSL," 2018, doi: 10.1145/3183895.3183901.

[10] B. Bichsel, M. Baader, T. Gehr, and M. Vechev, "Silq: A high-level quantum language with safe uncomputation and intuitive semantics," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2020, pp. 286–300, doi: 10.1145/3385412.3386007.

[11] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, "The theory of variational hybrid quantum-classical algorithms," *New Journal of Physics*, vol. 18, no. 2, 2016, doi: 10.1088/1367-2630/18/2/023023.

[12] M. Cerezo *et al.*, "Variational quantum algorithms," *Nature Reviews Physics*, vol. 3, no. 9, pp. 625–644, 2021, doi: 10.1038/s42254-021-00348-9.

[13] V. Bergholm *et al.*, "PennyLane: Automatic differentiation of hybrid quantum-classical computations." 2022, doi: 10.48550/arXiv.1811.04968.

[14] G. Aleksandrowicz *et al.*, "Qiskit: An open-source framework for quantum computing," *(No Title)*, vol. 2.2.3. Zenodo, 2025, doi: 10.5281/zenodo.2562111.
[15] J. R. Wootton, F. Harkins, N. T. Bronn, A. C. Vazquez, A. Phan, and A. T. Asfaw, "Teaching quantum computing with an interactive textbook," in *Proceedings - 2021 IEEE International Conference on Quantum Computing and Engineering, QCE 2021*, 2021, pp. 385–391, doi: 10.1109/QCE52317.2021.00058.
[16] Cirq Developers, "Cirq." Zenodo, 2024, doi: 10.5281/zenodo.11398180.
[17] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Annual ACM Symposium on Theory of Computing*, 1996, vol. Part F129452, pp. 212–219, doi: 10.1145/237814.237866.
[18] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, 1997, doi: 10.1137/S0097539795293172.
[19] S. Lloyd, "Universal quantum simulators," *Science*, vol. 273, no. 5278, pp. 1073–1078, 1996, doi: 10.1126/science.273.5278.1073.
[20] L. Zhou, S. T. Wang, S. Choi, H. Pichler, and M. D. Lukin, "Quantum approximate optimization algorithm: performance, mechanism, and implementation on near-term devices," *Physical Review X*, vol. 10, no. 2, 2020, doi: 10.1103/PhysRevX.10.021067.
[21] L. Arufe, M. A. González, A. Oddi, R. Rasconi, and R. Varela, "Quantum circuit compilation by genetic algorithm for quantum approximate optimization algorithm applied to MaxCut problem: Quantum circuit compilation by genetic algorithm for QAOA applied to MaxCut problem," *Swarm and Evolutionary Computation*, vol. 69, 2022, doi: 10.1016/j.swevo.2022.101030.
[22] J. R. McClean, M. E. Kimchi-Schwartz, J. Carter, and W. A. De Jong, "Hybrid quantum-classical hierarchy for mitigation of decoherence and determination of excited states," *Physical Review A*, vol. 95, no. 4, 2017, doi: 10.1103/PhysRevA.95.042308.
[23] R. Schutski, D. Lykov, and I. Oseledets, "Adaptive algorithm for quantum circuit simulation," *Physical Review A*, vol. 101, no. 4, 2020, doi: 10.1103/PhysRevA.101.042335.
[24] B. Villalonga *et al.*, "A flexible high-performance simulator for verifying and benchmarking quantum circuits implemented on real hardware," *npj Quantum Information*, vol. 5, no. 1, 2019, doi: 10.1038/s41534-019-0196-1.
[25] A. García de la Barrera, I. García-Rodríguez de Guzmán, M. Polo, and M. Piattini, "Quantum software testing: State of the art," *Journal of Software: Evolution and Process*, vol. 35, no. 4, 2023, doi: 10.1002/smr.2419.
[26] P. Zhao, Z. Miao, S. Lan, and J. Zhao, "Bugs4Q: A benchmark of existing bugs to enable controlled testing and debugging studies for quantum programs," *Journal of Systems and Software*, vol. 205, 2023, doi: 10.1016/j.jss.2023.111805.
[27] M. Paltenghi and M. Pradel, "Analyzing quantum programs with LintQ: a static analysis framework for qiskit," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 2144–2166, 2024, doi: 10.1145/3660802.
[28] M. De Stefano, F. Pecorelli, D. Di Nucci, F. Palomba, and A. De Lucia, "The quantum frontier of software engineering: A systematic mapping study," *Information and Software Technology*, vol. 175, 2024, doi: 10.1016/j.infsof.2024.107525.
[29] M. Sajjan *et al.*, "Quantum machine learning for chemistry and physics," *Chemical Society Reviews*, vol. 51, no. 15, pp. 6475–6573, 2022, doi: 10.1039/d2cs00203e.
[30] M. Cerezo, G. Verdon, H. Y. Huang, L. Cincio, and P. J. Coles, "Challenges and opportunities in quantum machine learning," *Nature Computational Science*, vol. 2, no. 9, pp. 567–576, 2022, doi: 10.1038/s43588-022-00311-3.
[31] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, 2018, doi: 10.22331/q-2018-08-06-79.
[32] S. Lopez Alarcón, E. Wong, T. S. Humble, E. Dumitrescu, and A. Elster, "Quantum programming paradigms and description languages," *Computing in Science and Engineering*, vol. 25, no. 6, pp. 33–38, 2023, doi: 10.1109/MCSE.2024.3375432.
[33] S. Ali, T. Yue, and R. Abreu, "When software engineering meets quantum computing," *Communications of the ACM*, vol. 65, no. 4, pp. 84–88, 2022, doi: 10.1145/3512340.

BIOGRAPHIES OF AUTHORS



Laksana Talenta Ahmad is a researcher at the Embedded Systems and Power Electronics Research Group (ESPERG) in Yogyakarta, Indonesia, and a student in the Bachelor of Information Technology (BIT) program, Department of Information Systems (DIS), Kulliyyah of Information and Communication Technology (KICT) at the International Islamic University Malaysia (IIUM) in Kuala Lumpur, Malaysia. His research interests encompass various aspects of information technology, including organizational informatics, database programming, robotics, and artificial intelligence. He can be contacted at email: laksanatadz@gmail.com.