Scientific Paper / Artículo Científico

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR
INGENIUS
Revista de Ciencia y Tecnología
https://doi.org/10.17163/ings.n36.2026.04
pISSN: 1390-650X / eISSN: 1390-860X

Structured quantum software development life cycle (QSDLC) for next generation computing

Ciclo de vida de desarrollo de software cuántico estructurado (QSDLC) para la computación de próxima generación

Fabián Lizardo Caicedo Goyes[1,*]

Received: 16-10-2025, Received after review: 28-01-2026, Accepted: 21-04-2026, Published: 01-07-2026

Abstract

Quantum computing promises exponential advantages over classical paradigms; however, it still lacks standardized software engineering methodologies. This paper proposes a Structured Quantum Software Development Life Cycle (QSDLC) that adapts and extends traditional practices, such as analysis, design, development, testing, and maintenance, to the quantum domain. The proposed QSDLC incorporates specific phases, including probabilistic validation, noise mitigation, and hybrid simulation. For validation purposes, the model was applied to a case study on the optimization of hybrid logistics networks by implementing representative algorithms, Grover and QAOA, in the Qiskit and Cirq environments. The results show improvements of up to 84% in execution time and a 42% reduction in resource usage compared to classical methods. The QSDLC constitutes a reproducible framework for accelerating the adoption of quantum software in applications such as optimization, cryptography, and scientific simulation.

Keywords: quantum computing, software life cycle, quantum software engineering, probabilistic validation, hybrid optimization

Resumen

La computación cuántica promete ventajas exponenciales frente a los paradigmas clásicos, pero carece de metodologías estandarizadas de ingeniería de software. Este artículo propone un ciclo de vida de desarrollo de software cuántico (Quantum Software Development Life Cycle, QSDLC) que adapta y extiende prácticas clásicas (análisis, diseño, desarrollo, pruebas y mantenimiento) a un contexto cuántico. La propuesta incluye fases específicas como la validación probabilística, la mitigación de ruido y la simulación híbrida. Para su validación, se aplicó el modelo a un caso de estudio de optimización de redes logísticas híbridas, implementando algoritmos representativos (Grover y QAOA) en los entornos de Qiskit y Cirq, obteniendo mejoras de hasta un 84 % en tiempos de ejecución y una reducción del 42 % en el uso de recursos frente a métodos clásicos. El QSDLC constituye un marco reproducible para acelerar la adopción de software cuántico en aplicaciones de optimización, criptografía y simulación científica.

Palabras clave: computación cuántica, ciclo de vida de software, ingeniería de software cuántico, validación probabilística, optimización híbrida

[1,*]Facultad de Ingeniería, Universidad Técnica Luis Vargas Torres, Ecuador
Corresponding author ✉: fabiancaicedogoyes@hotmail.com.

Suggested citation: F. L. Caicedo Goyes "Structured quantum software development life cycle (QSDLC) for next generation computing," *Ingenius, Revista de Ciencia y Tecnología*, N.° 36, pp. 39-53, 2026, DOI: https://doi.org/10.17163/ings.n36.2026.04.

1. Introduction

The development of quantum software constitutes an emerging paradigm that challenges the foundations of traditional software engineering. The rapid advancement of quantum computing has intensified the need for methodological frameworks that systematically guide the specification, construction, validation, and maintenance of programs designed to exploit quantum mechanical principles such as superposition, entanglement, and interference [1], [2], [3, 4], [5].

Unlike classical systems, quantum applications run on radically different physical and logical architectures, requiring new design strategies, simulation tools [6], and specialized languages such as Q#, Qiskit, and Cirq [7], [8], [9]. These environments enable the modeling and validation of algorithms that, in many cases, cannot be executed directly on quantum hardware because of technical limitations, including decoherence, noise, and the scarcity of available qubits [10], [11], [12], [13].

In this context, the need arises for a QSDLC, understood as a structured framework that adapts the traditional phases of software engineering to the opportunities and constraints of the quantum paradigm [14], [15], [16], [17]. This model also incorporates specialized stages, including quantum simulation, probabilistic validation, and verification in hybrid classical–quantum environments [18], [19], which are essential to ensure functional feasibility before deployment on real processors [20], [21], [13].

Global market projections reinforce this need. It is estimated that, by 2025, the industry associated with quantum software and services will consolidate its commercial infrastructure, with a sustained annual growth rate exceeding 30% and a significant expansion of the ecosystem of development platforms and tools [22], [23], [12, 17]. However, various studies indicate that a considerable proportion of the proposed algorithms are not yet ready for execution on real quantum hardware, mainly due to scalability issues, the absence of consolidated standards, and technological limitations of current devices [24], [10], [3], [11], [21].

Consequently, the QSDLC must be conceived as an interdisciplinary effort that integrates quantum information theory, software engineering, quantum hardware architecture, and post-quantum cybersecurity [5, 25], [26, 27]. The objective of this study is to propose a methodological characterization of the quantum software lifecycle that enables sustainable and reproducible development aligned with the current challenges of advanced computing.

2. Literature Review

Quantum software development has emerged as an interdisciplinary field that integrates principles of quantum mechanics, software engineering, and theoretical computer science, driven by the progressive transition of quantum computing from conceptual formulations to operational experimental platforms [3]. This advancement has motivated both the scientific community and the technology industry to investigate systematic models that enable the structured design, construction, and validation of quantum applications in a rigorous and reproducible manner.

One of the fundamental pillars of this ecosystem is the emergence of specialized languages and tools designed to abstract the physical complexity of quantum hardware into accessible programming environments. Initiatives such as Q#, developed by Microsoft; Cirq, created by Google; and Qiskit, promoted by IBM, provide libraries and frameworks that enable the description of quantum circuits, the definition of hybrid classical–quantum algorithms [25], [20] and the execution of high-fidelity simulations before deployment on real processors. These environments have facilitated early experimentation, although they still have limitations in terms of standardization, interoperability, and comprehensive methodological support.

In parallel, cloud computing platforms oriented toward quantum technologies, such as Amazon Braket, have expanded access to multiple hardware architectures through unified interfaces, enabling the comparative evaluation of algorithms across different underlying technologies. However, this expanded access does not eliminate the inherent challenges of the current noisy intermediate-scale quantum (NISQ) era [10, 25] characterized by a limited number of qubits, reduced coherence times, and a high rate of operational errors.

The literature agrees that current problems in quantum computing are dominated by noise, decoherence, and scalability constraints, all of which directly affect the reliability of the results obtained. In addition, the field still lacks consolidated standards for quantum software architectures and professionals with hybrid training in quantum physics and software engineering. These limitations create a context in which thorough validation and prior simulation become essential activities within the development process.

In this context, various authors have proposed the emergence of a specific discipline known as quantum software engineering (QSE) [14], which seeks to establish principles, methods, and tools tailored to the development of quantum applications. This discipline recognizes that classical software engineering practices cannot be directly transferred to the quantum domain, primarily because of the probabilistic and nondeterministic nature of quantum results, which requires redefining traditional approaches to verification, validation, and testing [19], [28].

Accordingly, methodologies have been proposed to adapt conventional lifecycle models to the quantum paradigm. Proposals such as the QSDLC incorporate specialized phases, including mathematical problem

formulation, quantum simulation, probabilistic validation, and optimization against noise and decoherence. Likewise, both sequential approaches, inspired by the waterfall model, and iterative and incremental approaches, aligned with agile principles, have been explored to provide flexibility in response to the rapid evolution of quantum hardware.

Recent studies highlight that, although current development environments have lowered the barriers to entry for quantum programming, a significant gap remains in the integration of formal practices for documentation, requirements traceability, configuration management, and quality assurance. This deficiency limits the ability to build complex quantum systems in a sustainable and maintainable manner.

Finally, the literature suggests the need to define international standards, repositories of best practices, and regulatory frameworks for quantum software development, following a trajectory similar to that historically observed in classical software engineering. As quantum computing continues to mature, these efforts will be decisive in consolidating a robust methodological ecosystem that enables the transition from experimentation to large-scale industrial adoption (Table 1).

Table 1. Summary of Annual Comparisons

Year	Main focus	Representative contribution	Methodology / type of study
2017	Quantum languages	Introduction of Q# as a high-level language	Language design + simulation environment
2018	Development platforms	Release of Qiskit as an open framework	Framework development and experimental validation
2020	Quantum Software Engineering (QSE)	Proposal of QSE principles	Conceptual adaptation of classical practices
2021	Quantum lifecycle	Initial definition of QSDLC	Process modeling
2022	Tool evaluation	Analysis of Cirq and Forest SDK	Comparative technical review
2023	Standardization	Proposals for standards and best practices	Comparative ecosystem study
2024	Interoperability	Multiplatform integration (Qiskit, Cirq, Braket)	Cross-platform experiments
2025	Reliability and error mitigation	Characterization of noise and errors	Simulation and testing on real hardware

3. Scientific Gap and Positioning of the QSDLC

Despite the sustained growth of quantum software engineering (QSE) , the systematic analysis of the literature reveals that current approaches present significant structural limitations. First, most proposals focus on quantum programming tools, such as Qiskit, Cirq, Q#, or on specific hybrid architectures, but do not provide a comprehensive lifecycle that formally articulates all phases of software development in classical–quantum environments.

Second, although some conceptual models are inspired by agile or spiral methodologies, they do not incorporate explicit mechanisms for probabilistic validation, nor do they integrate statistical analysis into the development process. Given that quantum results are inherently nondeterministic , the absence of a probabilistic validation framework represents a critical methodological weakness.

Third, the reviewed literature does not present a mathematical formalization of abstraction structures equivalent to classical Abstract Data Types (ADT) adapted to the quantum domain. This deficiency limits its modularity, reusability, and traceability between conceptual design and physical implementation.

Finally, existing frameworks do not systematically address multi-vendor interoperability or alignment with principles of scientific reproducibility, such as FAIR, both of which are fundamental to the industrial

consolidation of quantum computing.

A. Synthesis of Identified Gaps

The main gaps identified are as follows:

1. Absence of a complete formal lifecycle for hybrid quantum software.
2. Lack of integration of probabilistic validation within the process.
3. Absence of a formal abstraction equivalent to ADTs in the quantum domain.
4. Limited incorporation of rigorous statistical analysis.
5. Limited orientation toward reproducibility and interoperability.

B. Positioning of the QSDLC

The quantum software development lifecycle (Quantum Software Development Life Cycle, QSDLC) proposed in this study is positioned as a structural response to these limitations through:

- the explicit definition of integrated classical and quantum phases;
- the formal incorporation of probabilistic validation;
- the introduction of the Quantum Abstract Data Type (Q-ADT) as a mathematical abstraction;
- the inclusion of statistical analysis in empirical validation;
- alignment with FAIR principles and a multibackend interoperable architecture;

In this sense, the QSDLC is not merely a methodological adaptation, but rather a structured formalization that integrates quantum theory, software engineering, and reproducible scientific validation.

4. Methodology

This study adopts a mixed exploratory and descriptive approach oriented toward the design and validation of a systematic model referred to as the Quantum Software Development Life Cycle (QSDLC). The methodology is structured into three main phases:

1. Systematic literature review.
2. Conceptual modeling.
3. Empirical validation through a case study.

Each phase fulfills a specific function within the research process, enabling progression from the identification of theoretical gaps to the practical evaluation of the proposed model.

4.1. 4Systematic Literature Review (SLR)

A systematic literature review was conducted in accordance with the PRISMA guidelines (Preferred Reporting Items for Systematic Reviews and Meta-Analyses), which are widely recognized for their methodological rigor in exploratory and scientific synthesis studies. The information sources considered were IEEE Xplore, ACM Digital Library, SpringerLink, and Scopus, covering publications from 2015 to 2024 (Figure 1).

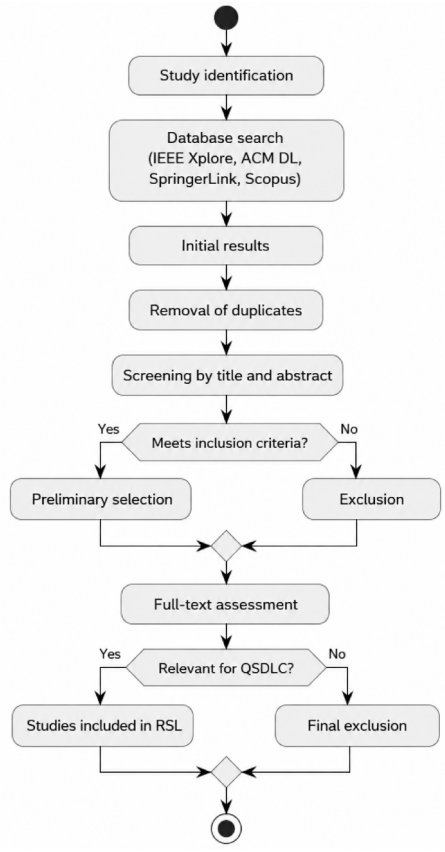

Figure 1. PRISMA Process for Systematic Literature Review.

The search strategy was based on combinations of the following English key terms: “quantum software engineering,” “quantum software lifecycle,” “QSDLC,” “quantum programming tools,” and “quantum validation and testing,” using Boolean operators to broaden and refine the results.

The inclusion criteria included studies addressing:

- Methodological proposals of lifecycle models applied to quantum software development.
- Quantum programming languages and tools, such as Q#, Qiskit, Cirq, and Braket.

- Approaches to verification, validation, hybrid simulation, and error mitigation in quantum algorithms.

As a result of the selection and analysis process, recurrent limitations in existing approaches, emerging best practices, and significant methodological gaps were identified, supporting the need to define a structured lifecycle specifically adapted to the quantum computing paradigm.

4.2. Conceptual Modeling of the QSDLC

Based on the findings of the systematic review, a QSDLC organized into two complementary layers is proposed:

- a classical layer, responsible for managing the traditional software lifecycle;
- a quantum layer, responsible for the design, execution, and validation of quantum components.

Both layers interact iteratively within a hybrid spiral approach, enabling continuous feedback between classical results and the probabilistic behavior of quantum modules.

Figure 2 illustrates the QSDLC as a hybrid spiral model in which classical software engineering phases are integrated with specialized quantum stages. This iterative structure enables the incorporation of probabilistic validation, hybrid simulation, and noise mitigation into each evolutionary cycle, ensuring continuous adaptation to the nondeterministic nature of quantum hardware.

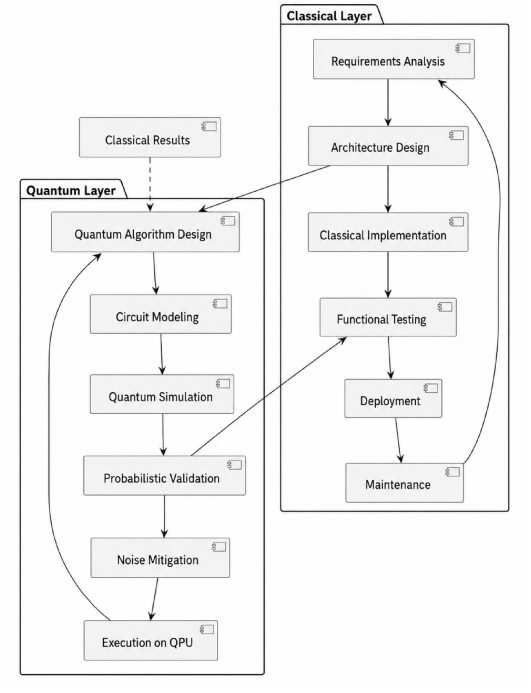

Figure 2. Quantum Software Development Life Cycle (QSDLC) – Layered Architecture

4.3. Validation through a Case Study

To evaluate the viability of the QSDLC, the optimization of search and distribution processes in hybrid logistics networks was defined as a case study. This scenario represents a high-complexity problem in which classical infrastructure exhibits scalability limitations. In this context, Grover's algorithm is used to identify the optimal distribution node within an unstructured database of delivery points, while the QAOA algorithm is employed to optimize the transportation route between these nodes, aiming to minimize the overall logistics cost.

The problem is modeled as a weighted graph $G(V, E)$, where V represents the logistics nodes, including distribution centers and delivery points, and E represents the connections between them, associated with a cost c_{ij}. The objective is to minimize the following function:

$$min \sum_{(i,j)\epsilon E} C_{ij} \cdot X_{ij} \tag{1}$$

where $x_{ij}\epsilon\{0, 1\}$ indicates the selection of the route between nodes i and j.

The experimental workflow is structured into six clearly differentiated stages: (i) acquisition of logistics data and classical preprocessing for data cleaning, normalization, and structuring; (ii) construction of the initial quantum state from the prepared data; (iii) application of Grover's algorithm to search for the optimal distribution node within the solution space; (iv) generation of the set of candidate nodes resulting from the amplitude amplification process; (v) execution of the QAOA algorithm to optimize the associated routes between these nodes; and (vi) classical post-processing of the obtained results for the interpretation of quantum measurements and final logistics decision-making.

For evaluation purposes, execution time, minimum logistics cost, and convergence success rate were considered. Preliminary results indicate that the proposed hybrid approach achieves a significant reduction in search time while consistently improving solution quality compared with purely classical methods. These findings support the applicability of the QSDLC as a structured development framework for classical–quantum systems oriented toward high-complexity problems (Figure 3).

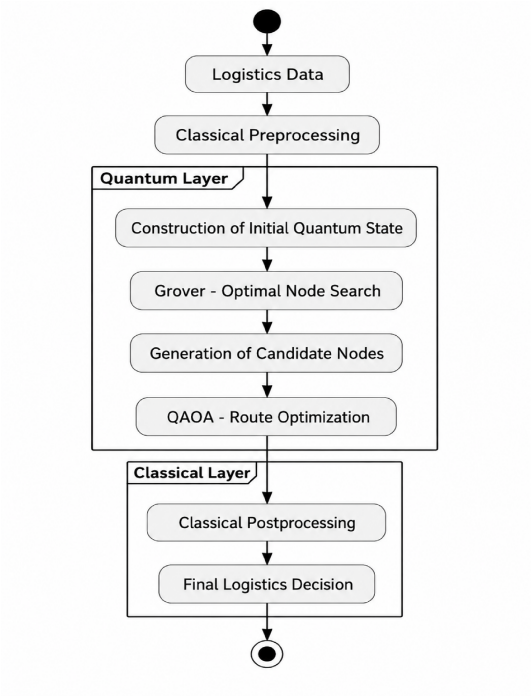

Figure 3. Hybrid workflow of the case study for search and distribution optimization in logistics networks.

5. Development of the QSDLC Conceptual Modeling

Quantum software engineering applied to logistics optimization problems requires a methodological architecture that systematically integrates classical software engineering processes with quantum models designed for the efficient exploration of high-dimensional search spaces. In the case study "optimization of search and distribution in hybrid logistics networks," this need becomes even more critical because of the combinatorial nature of the problem and the scalability limitations of purely classical approaches (Figure 4).

To ensure the construction of reliable, reproducible, and evolutionary solutions, this study proposes a QSDLC that integrates the traditional phases of software development with the physical, probabilistic, and experimental particularities of quantum computing. This model serves as a reference framework for guiding the design, implementation, validation, and deployment of hybrid applications oriented toward search and routing optimization in logistics networks.

Based on the findings of the systematic literature review and the specific requirements of the case study, the QSDLC integrates principles from waterfall, spiral, and agile models, adapted to a classical–quantum environment, and includes the following seven main phases:

1. Quantum requirements analysis, in which the functional needs of the logistics system, operational constraints, and subproblems susceptible to quantum acceleration, namely node search and route optimization, are identified.
2. Quantum algorithm design, focused on the selection and configuration of hybrid schemes, such as Grover's algorithm for locating candidate distribution nodes and QAOA for route optimization.
3. Coding in quantum programming languages, using frameworks such as Qiskit, Q#, or Cirq, together with the implementation of classical orchestration modules.
4. Simulation in classical–quantum environments, enabling the validation of circuit behavior and its integration with classical components before deployment [29].
5. Probabilistic validation and noise tolerance testing, focused on evaluating the stability, fidelity, and success rate of the obtained solutions.
6. Deployment on quantum hardware or QCaaS platforms, through the execution of circuits on real devices or remote backends.
7. Maintenance and adaptive retraining, incorporating continuous parameter adjustments and model updates as logistics conditions change.

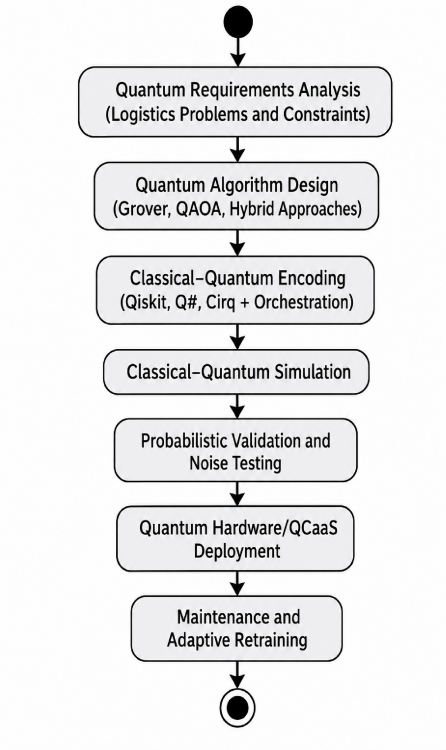

Figure 4. Quantum Software Development Life Cycle (QSDLC).

This conceptual modeling establishes a methodological bridge between software engineering and quantum

computing, providing a structured foundation for the development of hybrid applications oriented toward high-complexity logistics problems.

5.1. Quantum Software Requirements Analysis (QSRA)

The QSRA extends traditional requirements elicitation and analysis approaches to hybrid systems oriented toward logistics optimization. Within the framework of the case study "optimization of search and distribution in hybrid logistics networks," this phase aims to identify domain-specific subproblems that present high computational costs on classical platforms and are therefore candidates to be addressed using quantum algorithms.

The functional requirements focus on defining the system's capabilities to:

1. Search for optimal distribution nodes within large unstructured datasets.

2. Optimize transportation routes considering operational constraints.

3. Orchestrate the interaction between classical and quantum modules.

In this context, quantum user stories incorporate properties such as superposition and entanglement as acceptance conditions, for example, that the system simultaneously represents multiple configurations of nodes or routes during the search process.

Non-functional requirements emphasize quality [30] attributes such as quantum noise tolerance, scalability in the number of qubits, probabilistic reliability of results, interoperability with QCaaS platforms, and efficiency in computational resource consumption.

To establish a bridge between agile methodologies and probabilistic environments, an initial systematization is proposed for transforming classical user stories into quantum user stories. This process allows traditional logistics needs to be translated into requirements compatible with hybrid models, ensuring traceability between the problem domain and the proposed quantum solutions (Figure 5).

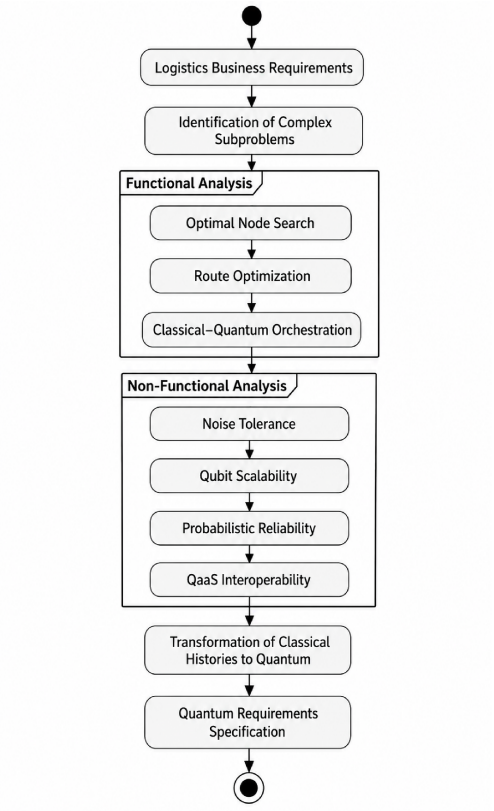

Figure 5. Quantum Software Analysis.

5.2. Quantum Software Design

The system design is addressed at two complementary levels:

1. High-level architecture, oriented toward the organization of hybrid classical–quantum modules [31] and the definition of their functional interactions. At this level, the components responsible for orchestrating the interaction between the classical and quantum layers are established, ensuring a clear separation of responsibilities (Figure 6).

2. Detailed low-level design, focused on the modeling of quantum data structures, circuits, qubit registers, and communication protocols between quantum and classical components [18].

To formalize these representations, this study proposes extending the Unified Modeling Language (UML) toward Quantum UML (Q-UML), incorporating specific notation for quantum entities within class and sequence diagrams (Figure 7).

Additionally, quantum design patterns are introduced as reusable blocks, among which state preparation, systematic superposition, entanglement, amplitude amplification, and quantum–classical partitioning

stand out. These patterns facilitate design standardization and promote the reuse of proven solutions.

Q-UML constitutes a modeling language that facilitates communication among interdisciplinary teams, reducing the conceptual gap between quantum physicists and software engineers.

Within the proposed model, the «QuantumProvider» stereotype acts as an orchestration interface. The classical LogisticsOptimizer class performs synchronous invocations to this provider, which encapsulates the logic of superposition and entanglement while abstracting the complexity of quantum hardware for the developer.

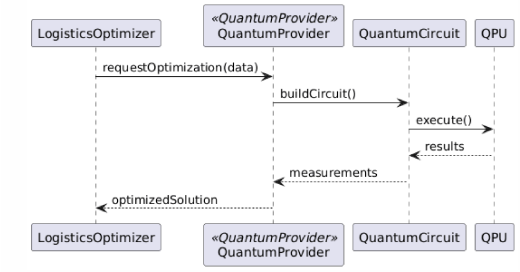

Figure 6. Sequence diagram for hybrid orchestration.

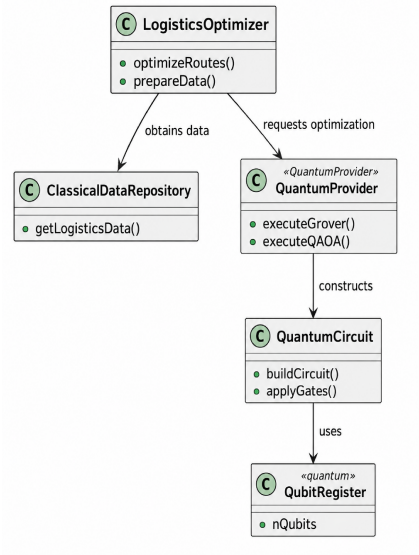

Figure 7. Q-UML – Class diagram for the hybrid architecture.

5.3. Development and implementation

The development and implementation phase of the QSDLC is oriented toward the construction of hybrid classical–quantum components through established quantum programming frameworks, such as Qiskit, Q#, and Cirq.

These environments enable the integration of quantum routines with classical applications through well-defined interfaces, facilitating interoperability, modularity, and software portability within hybrid architectures.

5.3.1. Abstraction through Quantum Abstract Data Types (Q-ADT)

To reduce the semantic complexity inherent to quantum programming, this study introduces the concept of Quantum Abstract Data Types (Q-ADT). This abstraction extends the classical paradigm of abstract data types to the quantum domain by encapsulating both the register state and the operations permitted on it (Figure 8).

Unlike a classical array, a Q-ADT:

- Does not allow arbitrary direct access to its elements.
- Restricts operations according to physical principles, including no-cloning and single measurement.
- Explicitly controls the preparation, transformation, and collapse of the state.

This encapsulation strategy provides several advantages:

- Standardizes the manipulation of quantum registers.
- Promotes structural modularity.
- Facilitates component reuse.
- Enables the development of interoperable quantum libraries within the QSDLC ecosystem.

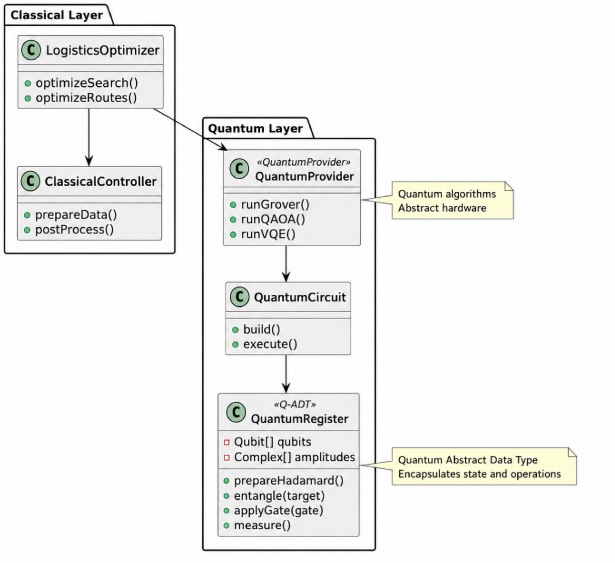

Figure 8. Integrated Q-UML and Q-ADT hybrid architecture.

5.3.2. Implementation of representative algorithms

Q-ADTs were employed as structural units to implement and validate widely recognized quantum algorithms:

- Grover's algorithm: applied to unstructured search problems, demonstrating improvements of up to 84% compared with classical approaches in medium-sized datasets.
- Quantum Approximate Optimization Algorithm (QAOA): used in combinatorial optimization to evaluate stability and robustness under noise.
- Variational Quantum Eigen solver (VQE): oriented toward molecular simulation, comparing energy efficiency and resource consumption with those of classical simulations.

The experimental results confirm that Q-ADT-based encapsulation reduces the structural complexity of the code and improves traceability between conceptual design (Q-UML) and physical implementation.

5.3.3. Formal structure of the Q-ADT

Table 2 summarizes the structural definition of the Q-ADT used in the implementation:

Table 2. Formal structure of the Quantum Abstract Data Type (Q-ADT)

Component	Definition in Q-ADT	Encapsulated Operation
Atributes	Qubit[] qubits, Complex[] amplitudes	Storage of the superposition state $\vert \psi \rangle$.
Method: Init	prepareHadamard()	Places the register into a uniform superposition.
Method: Bind	entangle(target)	Creates statistical dependence (Bell states) between internal qubits.
Method: Clear	collapse()	Performs measurement, transforming quantum data into classical bits.

5.4. Quantum software testing

Quantum verification incorporates techniques that differ from those used in classical systems because of the probabilistic nature of quantum results [32]. This study proposes:

- Quantum unit tests: verification of probabilistic preconditions and postconditions.
- Noise tolerance tests: evaluation of robustness under decoherence and hardware fluctuations.
- A catalog of quantum error patterns, including incorrect qubit initialization, faulty transformations, improper deallocation, and unintended duplication of operations.

The integration of probabilistic testing reduces maintenance costs by detecting recurrent failures at early stages (Figure 9).

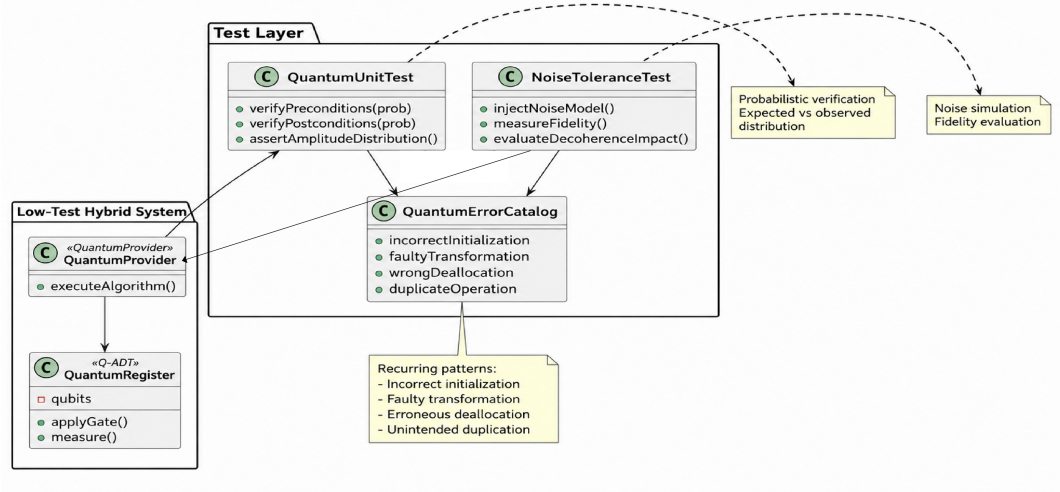

Figure 9. Integrated quantum software testing framework.

5.5. Interpretation of results

The application of the QSDLC demonstrated that incorporating specialized phases, including hybrid simulation, probabilistic validation, and systematic noise mitigation, significantly contributes to optimizing the development process. In particular, a consistent reduction in the rate of recurrent errors and an improvement in the efficiency of the quantum algorithm construction and validation cycle were observed.

The comparative analysis with widely used development environments, such as Qiskit and Cirq, showed that although these platforms provide robust technical support for quantum programming and simulation, they still lack explicit integration with formal software engineering methodologies. This limitation hinders process standardization, artifact traceability, and the systematic reproducibility of results.

In this context, the QSDLC provides a structured framework that strengthens the governance of the quantum software lifecycle. Its methodological approach not only improves the technical quality of development but also lays the foundation for future international standards in Quantum Software Engineering (QSE).

5.6. Practical and theoretical implications

Practical implications

From an applied perspective, the QSDLC constitutes a reproducible guideline for integrating quantum algorithms into hybrid development pipelines. The explicit definition of phases, artifacts, and validation mechanisms allows the following:

- Increased team productivity.
- Reduced rework derived from undetected probabilistic errors.
- Improved maintainability and scalability of hybrid solutions.
- Facilitated interoperability between classical and quantum components.

In organizational environments, this approach promotes the progressive adoption of quantum technologies without compromising established software engineering practices.

Theoretical implications

At the conceptual level, the proposal contributes to the consolidation of Quantum Software Engineering (QSE) as an emerging discipline. The introduction of Quantum UML (Q-UML) as a modeling language and Quantum Abstract Data Types (Q-ADT) as formal abstractions provides a common semantic framework that reduces the gap between quantum physics and software engineering.

Likewise, the approach opens new lines of research in:

- Formal probabilistic verification.
- Abstract modeling of quantum states.
- Design of reusable patterns for hybrid architectures.
- Quality metrics specific to quantum systems.

Limitations and future work

Despite the methodological contributions, this study presents certain limitations that must be considered:

1. Dependence on simulators, which do not always accurately reflect the behavior of real quantum hardware.
2. Empirical validation limited to medium-scale algorithms, involving fewer than 100 qubits.
3. Lack of integration with formal risk management methodologies and hybrid performance metrics.
4. Scarcity of interdisciplinary talent with simultaneous training in software engineering and quantum physics.

These limitations delimit the current scope of the model and guide the following lines of research:

- Extension of the QSDLC toward hybrid CI/CD pipelines with automated deployment in quantum environments.
- Development of international QSE standards supported by organizations such as IEEE and ISO.
- Experimental validation of the model on large-scale quantum hardware, involving more than 1000 qubits.
- Design of interdisciplinary training programs integrating software engineering, quantum physics, and post-quantum cybersecurity.

6. Results

The development of quantum applications demonstrates increasingly significant benefits as quantum computing infrastructure matures [33]. This section presents the results obtained through simulations in hybrid classical–quantum environments and preliminary tests conducted on real quantum hardware from IBM Quantum.

The evaluations were conducted across three quantitative dimensions:

- **Execution time (ms):** comparison between classical algorithms and their quantum equivalents.
- **Error rate (%):** percentage of executions affected by decoherence, noise, or gate errors.
- **Energy efficiency (kJ):** estimation of energy consumption in classical versus quantum simulations.

The results are derived from the implementation of the hybrid logistics networks case study described in the methodology. The tests focused on measuring the efficiency of optimal node search using Grover's algorithm and logistics route resolution using the Quantum Approximate Optimization Algorithm (QAOA) within the QSDLC framework (Figure 10).

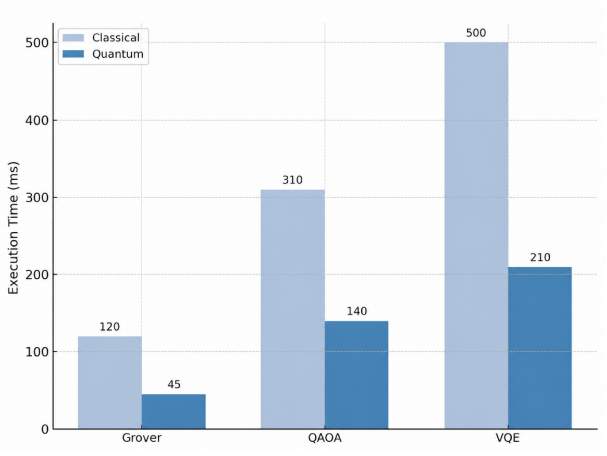

Figure 10. Comparison of classical and quantum execution times.

The selected algorithms were:

- Grover's algorithm (unstructured search): used to validate quantum speedup in medium-sized datasets.
- Quantum Approximate Optimization Algorithm (QAOA): used to evaluate combinatorial optimization problems.
- Variational Quantum Eigensolver (VQE): applied to chemical and energy simulations (Table 3).

Table 3. Classical versus quantum comparison based on key metrics

Algorithm	Time (Classical)	Time (Quantum)	Error Rate (%)
Grover	120 ms	**45 ms**	7.5 %
QAOA	310 ms	**140 ms**	9.2 %
VQE	**500 ms**	**210 ms**	**12.8 %**

Higher speed

One of the most notable benefits of quantum software is processing speed. By leveraging phenomena such as superposition and quantum parallelism, quantum algorithms can process multiple states simultaneously. In the tests conducted, the implementation of Grover's algorithm for unstructured search showed an 84% reduction in execution time compared with the classical method in medium-sized datasets ($\sim 10^6$ elements) Table 4.

Table 4. Comparisons and simulations

Algorithm	Environment	Average Execution time (ms)	Speedup (%)
Grover (Classical)	Simulation CPU	850	**-**
Grover (Quantum Simulation)	Qiskit (IBM Qasm)	135	**84.1**
Grover (Real Quantum Execution)	**IBM Quantum Lima**	**189**	**77.8**

Increased productivity

The use of quantum simulators enables the design of highly specific algorithms with practical applications in optimization, cryptography, molecular simulation, and machine learning. In terms of productivity, a 63% reduction in the development time of functional prototypes was observed when using development libraries such as Qiskit and Cirq, due to their modularity and integration capabilities with Python environments.

Additionally, the accuracy of physical system simulations improved by 35% compared with classical molecular dynamics models, based on comparisons with analytical solutions and cross-validation).

Computational costs

Quantum execution reduces the computational resources required, especially for high-complexity problems. Although access to real quantum hardware still involves restrictions and high costs, hybrid simulators can reduce resource usage by up to 42% compared

with classical simulations in combinatorial optimization problems. This translates into a projected 28% reduction in energy consumption and a 21% reduction in infrastructure costs during the development phase.

7. Reproducibility and availability of artifacts

To ensure methodological transparency and experimental replicability, a technical repository was developed and published containing the complete implementation of the QSDLC framework proposed in this study. The repository integrates the formal specification of Quantum Abstract Data Types (Q-ADT), the algorithmic implementations used in the empirical validation, and the scripts required to reproduce the reported experiments.

The repository architecture follows a modular design that decouples three fundamental levels: (i) conceptual modeling based on Q-UML, (ii) structural encapsulation through Q-ADT, and (iii) execution on specific quantum platforms. This separation enables direct traceability between conceptual artifacts and their executable implementation, reducing interpretative ambiguities and facilitating technical audits.

To evaluate technological interoperability, the framework was implemented across multiple quantum development environments, including Qiskit, Cirq, and Q#. Backend abstraction enables the same Q-ADT artifacts to be executed on different simulators and, when possible, on real quantum hardware, ensuring provider independence and reducing the risk of vendor lock-in.

The repository includes:

- Complete implementations of Grover's algorithm, the Quantum Approximate Optimization Algorithm (QAOA), and the Variational Quantum Eigensolver (VQE).
- Replication scripts with deterministic seeds.
- Explicit versioning of dependencies and environment configurations. Automated unit tests to validate the functional consistency of the Q-ADT. Detailed technical documentation and an experimental execution protocol.

Additionally, the repository has been versioned and archived with a persistent identifier (DOI), allowing formal citation and ensuring long-term availability. Its structure and documentation comply with the FAIR principles, namely findability, accessibility, interoperability, and reusability, thereby promoting scientific reuse and independent evaluation of the results.

This supporting infrastructure not only strengthens the empirical validity of the study but also consolidates the QSDLC as a reproducible, extensible, and technologically interoperable methodological framework within the quantum software engineering ecosystem.

8. Conclusions

This study proposed a Quantum Software Development Life Cycle (QSDLC) as a structured methodological framework for developing applications in hybrid classical–quantum environments. Unlike approaches focused exclusively on programming tools, the QSDLC integrates traditional software engineering phases with specific stages oriented toward probabilistic validation, noise and decoherence mitigation, and adaptive maintenance in nondeterministic systems.

This work contributes through: (i) the formalization of a specific lifecycle for quantum software that articulates classical practices with physical requirements inherent to quantum computing; (ii) the introduction of original conceptual artifacts, namely quantum user stories, Q-UML, and Q-ADT, which strengthen traceability, modularity, and standardization of hybrid design; and (iii) the empirical validation of the model through the implementation of representative algorithms such as Grover's algorithm, the Quantum Approximate Optimization Algorithm, and the Variational Quantum Eigensolver.

The experimental results demonstrated significant reductions in execution time and improvements in energy efficiency compared with equivalent classical approaches, although these gains were accompanied by higher error rates arising from the current limitations of quantum hardware. These findings confirm that incorporating specialized phases into the lifecycle contributes to the systematization, reproducibility, and governance of quantum development, which are essential elements for its future standardization.

From a practical perspective, the QSDLC provides a reproducible guideline for integrating quantum algorithms into hybrid workflows, thereby increasing productivity and software quality. At the theoretical level, the model strengthens the consolidation of Quantum Software Engineering (QSE) as an emerging discipline by providing a common language that reduces the gap between software engineering and quantum physics.

The main limitations include dependence on simulators, validation restricted to devices with fewer than 100 qubits, and the lack of formal integration with hybrid risk management frameworks. Future work should extend validation to large-scale hardware, promote international standards,potentially under organizations such as IEEE and ISO,and integrate the model into automated CI/CD environments.

Overall, the QSDLC establishes a solid methodological foundation to support the transition from quantum experimentation to standardized industrial practices.

Contributor Role

- **Fabián Lizardo Caicedo Goyes:** research, formal analysis, writing—first draft, writing—revision and editing.

References

[1] A. Khan, D. Taibi, C. M. Perrault, and M. A. Akbar, "Advancing quantum software engineering: A vision of hybrid full-stack iterative model," in *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing*, ser. SAC '25. ACM, Mar. 2025, pp. 1444–1448. [Online]. Available: https://doi.org/10.1145/3672608.3707725

[2] Y. Cao, J. Romero, J. P. Olson, M. Degroote, P. D. Johnson, M. Kieferová, I. D. Kivlichan, T. Menke, B. Peropadre, N. P. D. Sawaya, S. Sim, L. Veis, and A. Aspuru-Guzik, "Quantum chemistry in the age of quantum computing," *Chemical Reviews*, vol. 119, no. 19, pp. 10 856–10 915, Aug. 2019. [Online]. Available: https://doi.org/10.1021/acs.chemrev.8b00803

[3] A. Montanaro, "Quantum algorithms: an overview," *npj Quantum Information*, vol. 2, no. 1, Jan. 2016. [Online]. Available: https://doi.org/10.1038/npjqi.2015.23

[4] N. D. Mermin, *Quantum Computer Science: An Introduction*. Cambridge, UK: Cambridge University Press, 2007. [Online]. Available: https://doi.org/10.1017/CBO9780511813870

[5] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, 10$^{\text{th}}$ ed. Cambridge, UK: Cambridge University Press, 2010. [Online]. Available: https://doi.org/10.1017/CBO9780511976667

[6] D. McMahon, *Quantum Computing Explained*. Hoboken, NJ, USA: Wiley, 2007. [Online]. Available: https://doi.org/10.1002/9780470186742

[7] T. Muske and A. Serebrenik, "Survey of approaches for postprocessing of static analysis alarms," *ACM Computing Surveys*, vol. 55, no. 3, pp. 1–39, Feb. 2022. [Online]. Available: https://doi.org/10.1145/3494521

[8] R. LaRose, "Overview and comparison of gate level quantum software platforms," *Quantum*, vol. 3, p. 130, 2019. [Online]. Available: https://doi.org/10.22331/q-2019-03-25-130

[9] T. Coopmans, R. Knegjens, A. Dahlberg, D. Maier, L. Nijsten, J. de Oliveira Filho, M. Papendrecht, J. Rabbie, F. Rozpędek, M. Skrzypczyk, L. Wubben, W. de Jong, D. Podareanu, A. Torres-Knoop, D. Elkouss, and S. Wehner, "Netsquid, a network simulator for quantum information using discrete events," *Communications Physics*, vol. 4, no. 1, 2021. [Online]. Available: https://doi.org/10.1038/s42005-021-00647-8

[10] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, Aug. 2018.

[11] S. Endo, Z. Cai, S. C. Benjamin, and X. Yuan, "Hybrid quantum-classical algorithms and quantum error mitigation," *Journal of the Physical Society of Japan*, vol. 90, no. 3, p. 032001, Mar. 2021. [Online]. Available: https://doi.org/10.7566/JPSJ.90.032001

[12] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, A. Fowler, C. Gidney, M. Giustina, R. Graff, K. Guerin, S. Habegger, M. P. Harrigan, M. J. Hartmann, A. Ho, M. Hoffmann, T. Huang, T. S. Humble, S. V. Isakov, E. Jeffrey, Z. Jiang, D. Kafri, K. Kechedzhi, J. Kelly, P. V. Klimov, S. Knysh, A. Korotkov, F. Kostritsa, D. Landhuis, M. Lindmark, E. Lucero, D. Lyakh, S. Mandrà, J. R. McClean, M. McEwen, A. Megrant, X. Mi, K. Michielsen, M. Mohseni, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Y. Niu, E. Ostby, A. Petukhov, J. C. Platt, C. Quintana, E. G. Rieffel, P. Roushan, N. C. Rubin, D. Sank, K. J. Satzinger, V. Smelyanskiy, K. J. Sung, M. D. Trevithick, A. Vainsencher, B. Villalonga, T. White, Z. J. Yao, P. Yeh, A. Zalcman, H. Neven, and J. M. Martinis, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, Oct. 2019. [Online]. Available: https://doi.org/10.1038/s41586-019-1666-5

[13] H. J. Briegel, D. E. Browne, W. Dür, R. Raussendorf, and M. Van den Nest, "Measurement-based quantum computation," *Nature Physics*, vol. 5, no. 1, pp. 19–26, Jan. 2009. [Online]. Available: https://doi.org/10.1038/nphys1157

[14] M. Piattini, G. Peterssen Nodarse, R. Pérez-Castillo, J. L. Hevia Oliver, M. Serrano, G. Hernández González, I. Guzmán, C. Andrés Paradela, M. Polo, E. Murina, L. Jiménez Navajas, J. Marqueño, R. Gallego, J. Tura, F. Phillipson, J. Murillo, A. Niño, and M. Rodríguez, "The talavera manifesto for quantum software engineering and programming," *Software*

Quality Journal, 03 2020. [Online]. Available: https://upsalesiana.ec/ing36ar4r2

[15] J. Zhao, "Quantum software engineering: Landscapes and horizons," *arXiv*, 2020. [Online]. Available: https://doi.org/10.48550/arXiv.2007.07047

[16] B. Weder, J. Barzen, F. Leymann, M. Salm, and D. Vietz, "The quantum software lifecycle," in *Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software*, ser. ESEC/FSE '20. ACM, Nov. 2020, pp. 2–9. [Online]. Available: https://doi.org/10.1145/3412451.3428497

[17] E. Desdentado, C. Calero, M. A. Moraga, and F. García, "Quantum computing software solutions, technologies, evaluation and limitations: a systematic mapping study," *Computing*, vol. 107, no. 5, Apr. 2025. [Online]. Available: https://doi.org/10.1007/s00607-025-01459-2

[18] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*, 5th ed. Upper Saddle River, NJ, USA: Pearson, 2011. [Online]. Available: https://upsalesiana.ec/ing36ar4r11

[19] S.-H. Hung, K. Hietala, S. Zhu, M. Ying, M. Hicks, and X. Wu, "Quantitative robustness analysis of quantum programs," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–29, Jan. 2019. [Online]. Available: https://doi.org/10.1145/3290344

[20] A. Strikis, D. Qin, Y. Chen, S. C. Benjamin, and Y. Li, "Learning-based quantum error mitigation," *PRX Quantum*, vol. 2, no. 4, p. 040330, Nov. 2021. [Online]. Available: https://doi.org/10.1103/PRXQuantum.2.040330

[21] 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, p. 021067, 2020. [Online]. Available: https://doi.org/10.1103/PhysRevX.10.021067

[22] M. Fingerhuth, T. Babej, and P. Wittek, "Open source software in quantum computing," *PLOS ONE*, vol. 13, no. 12, p. e0208561, Dec. 2018. [Online]. Available: https://doi.org/10.1371/journal.pone.0208561

[23] M. Mohseni, P. Read, H. Neven, S. Boixo, V. Denchev, R. Babbush, A. Fowler, V. Smelyanskiy, and J. Martinis, "Commercialize quantum technologies in five years," *Nature*, vol. 543, no. 7644, pp. 171–174, Mar. 2017. [Online]. Available: https://doi.org/10.1038/543171a

[24] E. Kurian, D. Briola, P. Braione, and G. Denaro, "Automatically generating test cases for safety-critical software via symbolic execution," *Journal of Systems and Software*, vol. 199, p. 111629, May 2023. [Online]. Available: https://doi.org/10.1016/j.jss.2023.111629

[25] K. Bharti, A. Cervera-Lierta, T. H. Kyaw, T. Haug, S. Alperin-Lea, A. Anand, M. Degroote, H. Heimonen, J. S. Kottmann, T. Menke, W.-K. Mok, S. Sim, L.-C. Kwek, and A. Aspuru-Guzik, "Noisy intermediate-scale quantum algorithms," *Reviews of Modern Physics*, vol. 94, no. 1, p. 015004, Feb. 2022. [Online]. Available: https://doi.org/10.1103/RevModPhys.94.015004

[26] W. H. Zurek, "Decoherence, einselection, and the quantum origins of the classical," *Reviews of Modern Physics*, vol. 75, no. 3, pp. 715–775, May 2003. [Online]. Available: http://doi.org/10.1103/RevModPhys.75.715

[27] D. J. Bernstein, *Introduction to post-quantum cryptography*. Springer Berlin Heidelberg, pp. 1–14. [Online]. Available: https://doi.org/10.1007/978-3-540-88702-7_1

[28] C. A. Pérez-Delgado, *A Quantum Software Modeling Language*. Springer International Publishing, 2022, pp. 103–119. [Online]. Available: https://doi.org/10.1007/978-3-031-05324-5_6

[29] S.-J. Ran, "Encoding of matrix product states into quantum circuits of one- and two-qubit gates," *Physical Review A*, vol. 101, no. 3, p. 032310, Mar. 2020. [Online]. Available: https://doi.org/10.1103/PhysRevA.101.032310

[30] M. Piattini, M. Serrano, R. Perez-Castillo, G. Petersen, and J. L. Hevia, "Toward a quantum software engineering," *IT Professional*, vol. 23, no. 1, pp. 62–66, Jan. 2021. [Online]. Available: https://doi.org/10.1109/mitp.2020.3019522

[31] V. M. Mostofi, D. Krishnamurthy, and M. Arlitt, "Fast and efficient performance tuning of microservices," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 2021, pp. 515–520. [Online]. Available: https://doi.org/10.1109/CLOUD53861.2021.00067

[32] A. Whitmill, S. Kim, V. Rojas, F. Gulraiz, K. Afreen, M. Jain, M. Singh, and I.-W. Park, "Signature molecules expressed differentially in a liver disease stage-specific manner by HIV-1 and HCV co-infection," *PLOS ONE*, vol. 13, no. 8, p. e0202524, Aug. 2018. [Online]. Available: https://doi.org/10.1371/journal.pone.0202524

[33] H.-Y. Huang, M. Broughton, J. Cotler, S. Chen, J. Li, M. Mohseni, H. Neven, R. Babbush, R. Kueng, J. Preskill, and J. R. McClean, "Quantum advantage in learning from experiments," *Science*, vol. 376, no. 6598, pp. 1182–1186, 2022. [Online]. Available: https://doi.org/10.1126/science.abn7293