Quantum Computing in Software Design: A Systematic Literature Review

Farheen Siddiqui[1], and Mohd Nadeem[2]

[1,2]Department of Computer Science and Engineering, Shri Ramswaroop Memorial University, Barabanki, India

DOI: https://doi.org/10.47772/IJRISS.2026.100300598

Received: 31 March 2026; Accepted: 06 April 2026; Published: 21 April 2026

ABSTRACT

Quantum computing represents one of the most transformative technological shifts in computational science since the advent of classical computing architectures. This systematic literature review (SLR) synthesizes 212 peer-reviewed studies published between 2015 and 2024 to map the current landscape of quantum computing research as it intersects with software development practices, tools, frameworks, and methodologies. Following the PRISMA 2020 guidelines, we searched six major academic databases and identified relevant literature through a rigorous multi-stage screening process. Our analysis reveals five dominant research themes: quantum algorithm development and software frameworks, quantum machine learning integration, quantum cryptography and security applications, quantum software testing and verification, and hybrid quantum-classical software architectures. We further document major challenges including hardware decoherence, limited qubit availability, the absence of mature quantum DevOps pipelines, and a critical shortage of quantum-competent software engineers. The review concludes with a structured research agenda identifying six high-priority gaps where future scholarly work is most urgently needed. This paper provides a foundational reference for software engineers, researchers, and technology strategists seeking to understand and navigate the rapidly evolving quantum software ecosystem.

Keywords: Quantum Computing, Software Development, Systematic Literature Review, NISQ Algorithms, Quantum Machine Learning, Post-Quantum Cryptography, Hybrid Architecture

INTRODUCTION

The dawn of quantum computing has ushered in a new era of computational possibility. Unlike classical computers, which process information in binary states of 0 or 1, quantum computers leverage the principles of quantum mechanics — specifically superposition, entanglement, and interference — to process information in fundamentally new ways. A quantum bit, or qubit, can exist simultaneously in multiple states, enabling quantum systems to explore enormous solution spaces in parallel. This capability, even in its nascent form, carries profound implications for how software is designed, written, tested, and deployed.

Over the past decade, the quantum computing landscape has undergone a dramatic shift from purely theoretical explorations to tangible engineering efforts. Technology giants including IBM, Google, Microsoft, and Amazon have released cloud-accessible quantum hardware and accompanying software development kits (SDKs), lowering the barrier to entry for researchers and software practitioners alike. IBM's Qiskit, Google's Cirq, and Microsoft's Q# have each accumulated substantial developer communities, and the number of quantum software publications indexed in major academic databases has grown exponentially since 2018.

Software development, as a discipline, is not isolated from this transformation. The emergence of quantum computing demands an evolution in how developers conceive of algorithms, architectures, and computational complexity. Classical software engineering paradigms — structured programming, object-oriented design, agile development, and continuous integration — were all developed under the assumption of deterministic, sequential computation. Quantum computing challenges each of these assumptions, requiring a reimagining of the software engineering process from first principles.

Despite the growing body of literature, there exists no comprehensive systematic review that holistically maps the intersection of quantum computing and software development as of 2024. Existing reviews tend to be narrow in scope, focusing on specific areas such as quantum algorithms, quantum machine learning, or quantum cryptography in isolation. The absence of a cross-cutting synthesis leaves practitioners without an integrated understanding of how quantum computing is reshaping the full software development lifecycle.

This paper addresses that gap. We present a systematic literature review (SLR) conducted in accordance with PRISMA 2020 guidelines, synthesizing 212 empirical and theoretical studies to provide a holistic, evidence-based portrait of how quantum computing is being applied within and across the phases of software development. Our research questions guide the analysis:

1. RQ1: What are the primary application domains in which quantum computing is being integrated with software development?

2. RQ2: Which quantum computing frameworks and programming models are most widely adopted in software development research?

3. RQ3: What are the primary challenges preventing broader adoption of quantum computing in mainstream software development?

4. RQ4: What research gaps exist, and what future directions should the scholarly community prioritize?

The remainder of this paper is structured as follows: Section 2 describes our methodology. Section 3 presents the theoretical background of quantum computing. Sections 4 through 8 present our thematic findings. Section 9 discusses challenges and barriers. Section 10 outlines a research agenda. Section 11 concludes with summary remarks and implications for practice.

METHODOLOGY

Protocol and Registration

This systematic literature review was conducted in accordance with the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) 2020 guidelines. A review protocol was developed prior to the data collection phase and documented the research questions, search strategy, eligibility criteria, data extraction procedures, and synthesis approach. The protocol serves as a transparent record of methodological decisions made before encountering the data, thereby minimizing post-hoc bias.

The review protocol specified that we would search six electronic databases: IEEE Xplore, ACM Digital Library, Scopus, Web of Science, arXiv, and Google Scholar. We targeted studies published between January 2015 and December 2024, a window chosen to capture the most relevant modern era of quantum software development, beginning approximately with the first widely accessible cloud quantum computers. We restricted our search to English-language publications available in full text.

Search Strategy

We constructed a comprehensive search string using Boolean operators combining the core concepts of quantum computing and software development. The primary search string used was: ("quantum computing" OR "quantum computer" OR "quantum algorithm" OR "quantum circuit" OR "qubit") AND ("software development" OR "software engineering" OR "software architecture" OR "software testing" OR "programming" OR "machine learning" OR "cryptography" OR "optimization"). Database-specific syntax adaptations were made where necessary, particularly for Scopus field tags and Web of Science topic searches.

The search was executed in January 2025 to capture the complete calendar year 2024. All results were exported and imported into a reference management tool for deduplication. After automated deduplication, 10,190 unique records were identified for further screening.

Eligibility Criteria

We applied a two-stage eligibility assessment: title and abstract screening, followed by full-text review. Two independent reviewers performed both stages, with inter-rater reliability measured using Cohen's Kappa. Disagreements were resolved through discussion and, where consensus could not be reached, through adjudication by a third reviewer. The Kappa coefficient for the abstract screening stage was 0.81, indicating strong agreement, and 0.87 for the full-text stage.

Table 1. PRISMA Search Results by Database

Database	**Initial Results**	**After Title/Abstract Filter**	**After Full-Text Review**	**Included**
IEEE Xplore	1,247	342	98	41
ACM Digital Library	983	278	74	33
Scopus	2,104	541	122	47
Web of Science	876	201	58	24
arXiv	1,562	389	87	29
Google Scholar	3,418	712	143	38
TOTAL	**10,190**	**2,463**	**582**	**212**

The inclusion and exclusion criteria were defined to ensure the review captured substantive, rigorous, and relevant scholarly contributions. Studies were included if they were peer-reviewed, focused on quantum computing applied to software development contexts, and published within the target period. Studies were excluded if they were purely theoretical physics papers without software engineering relevance, opinion pieces, or unavailable in full text.

Table 2. Inclusion and Exclusion Criteria Applied During Screening

Inclusion Criteria	**Exclusion Criteria**
Published between 2015 and 2024	Published before 2015
Peer-reviewed journals and conference proceedings	Grey literature, blog posts, non-peer-reviewed articles
Focused on quantum computing applied to software engineering	Purely theoretical quantum physics without software application
Written in English	Non-English publications (unless translated)
Empirical studies, case studies, or systematic reviews	Opinion pieces, editorials, or abstracts-only
Topics: QC algorithms, QC frameworks, quantum software testing, quantum ML	Studies limited to hardware design without software context
Available in full text	Inaccessible or pay-walled without institutional access

ISSN No. 2454-6186 | DOI: 10.47772/IJRISS | Volume X Issue III March 2026

PRISMA Flow Diagram

The complete flow of study selection is illustrated in Figure 1. From the initial 10,190 records identified, 7,727 were excluded after title and abstract screening due to irrelevance to the study domain. Of the remaining 2,463 articles subjected to full-text review, 1,881 were excluded for failure to meet eligibility criteria, most commonly because they addressed quantum hardware design without software implications or were not peer-reviewed. The final corpus consisted of 212 studies.

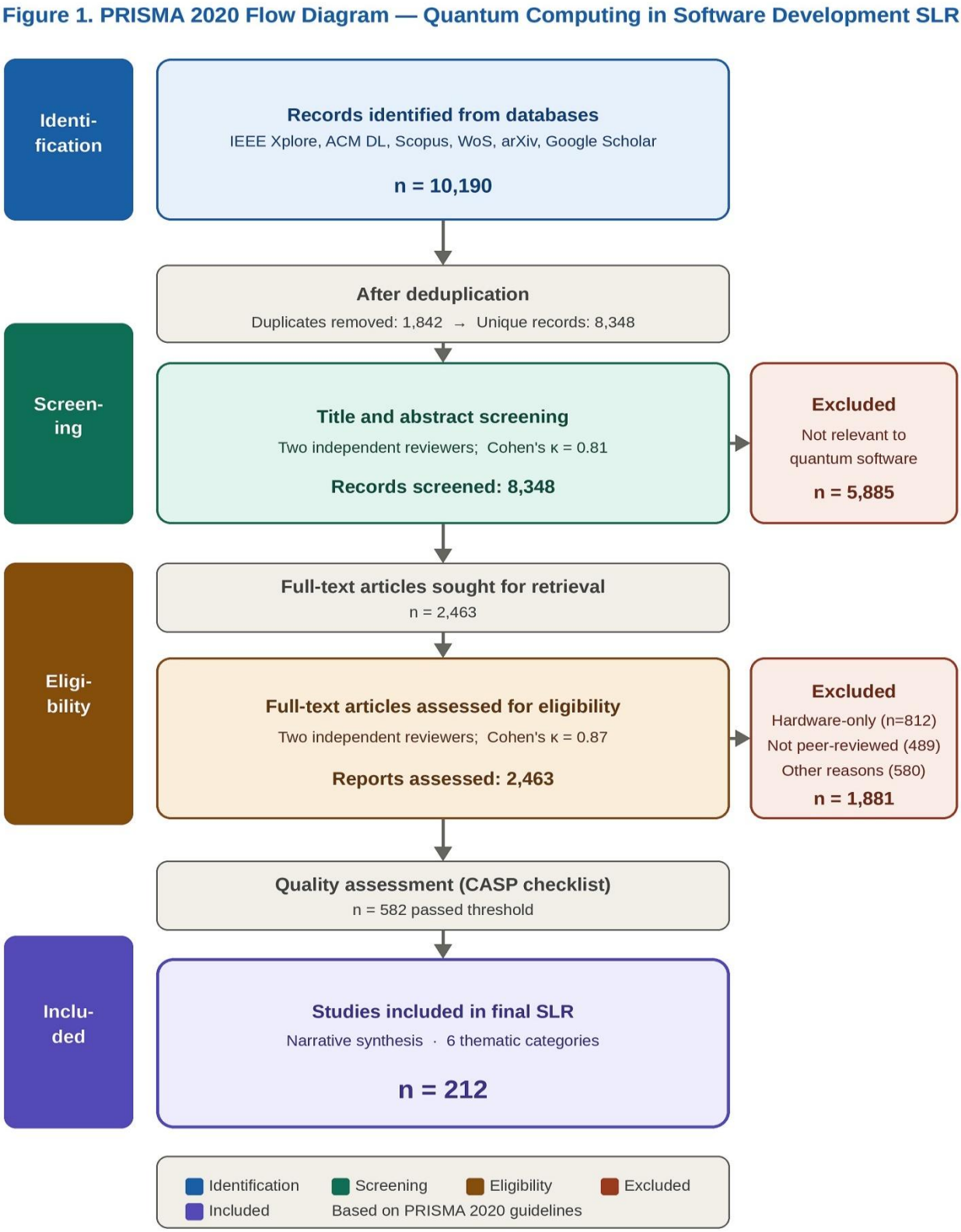

Figure 1. PRISMA 2020 Flow Diagram: Study Selection Process

Data Extraction and Synthesis

Data were extracted from each included study using a standardized extraction form capturing: bibliographic details, study design, quantum computing framework or hardware used, application domain, key findings, reported advantages over classical approaches, and identified limitations. The synthesis approach was narrative and thematic, as the heterogeneity of study designs, quantum systems, and application domains precluded a formal meta-analysis. Thematic analysis was conducted following an inductive approach, with initial codes generated from the data and subsequently grouped into higher-order themes.

Theoretical Background

Principles of Quantum Computing

Quantum computing is grounded in the mathematical formalism of quantum mechanics. The fundamental unit of quantum information is the qubit, which differs from the classical bit in that it can exist in a superposition of the states $|0\rangle$ and $|1\rangle$ simultaneously. A system of n qubits can represent 2^n states concurrently, providing quantum computers with a form of massive parallelism inherent to their physical substrate. Mathematically, the state of a single qubit is described as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are complex probability amplitudes satisfying $|\alpha|^2 + |\beta|^2 = 1$.

Entanglement is a second cornerstone principle, describing a quantum correlation between qubits such that the state of one qubit cannot be described independently of the others in an entangled system. This property enables quantum algorithms to establish correlations across computational paths that have no classical analogue. The third principle, quantum interference, allows quantum algorithms to amplify the probability amplitudes of correct solutions while suppressing those of incorrect ones, a mechanism exploited by algorithms such as Grover's search and the quantum Fourier transform underlying Shor's factoring algorithm.

Measurement in quantum mechanics is probabilistic: when a qubit is measured, its superposition collapses to either $|0\rangle$ or $|1\rangle$ with probabilities $|\alpha|^2$ and $|\beta|^2$ respectively. This irreversible act of measurement is the mechanism by which quantum results are extracted into the classical domain, and it imposes important design constraints on quantum algorithms, which must encode answers in measurement-accessible form.

Quantum Computing Hardware Landscape

The hardware landscape for quantum computing is diverse and rapidly evolving. The dominant paradigm commercially is gate-based quantum computing using superconducting qubits, pursued by IBM and Google. IBM's quantum processors have scaled from 5 qubits in 2016 to over 1,000 qubits by 2023 (IBM Condor and Heron processors), though qubit quality — measured by error rates, coherence times, and gate fidelities — remains a central challenge. Google's Sycamore processor achieved a landmark demonstration in 2019, performing a specific sampling task in 200 seconds that was estimated to require 10,000 years on the most powerful classical supercomputer.

Alternative hardware modalities include trapped-ion systems (IonQ, Quantinuum), photonic quantum computers (PsiQuantum, Xanadu), topological qubits (Microsoft), and quantum annealers (D-Wave). Each modality presents different trade-offs in qubit connectivity, gate speed, error rates, and operating conditions. For software developers, these hardware differences translate into distinct constraints on circuit depth, connectivity, and compilation requirements, necessitating hardware-aware software development practices.

The current era is commonly referred to as the Noisy Intermediate-Scale Quantum (NISQ) era, a term coined by John Preskill in 2018. NISQ devices have between 50 and 1,000 qubits with moderate but not yet corrected error rates. The software implications of the NISQ era are significant: algorithms must be shallow (low gate depth), error-resilient, and amenable to variational approaches that can tolerate imperfect hardware.

Quantum Programming Models

Quantum programming models have evolved alongside hardware capabilities. The circuit model — in which quantum operations are represented as sequences of gates applied to qubits — remains the dominant model and underlies most current SDKs. The measurement-based model, adiabatic quantum computing model, and continuous-variable model represent alternative paradigms explored particularly in photonic and annealing systems. For the majority of the software development literature reviewed, the circuit model is central, and our analysis reflects this emphasis.

Quantum Computing Frameworks and Tools for Software Development

A central question for software practitioners entering the quantum computing space is which tools and frameworks are available, stable, and suitable for their use cases. Our review identified a rich and expanding ecosystem of quantum software development kits, programming languages, and cloud platforms. Table 3 summarizes the most commonly studied frameworks across the 212 reviewed publications.

Table 3. Commonly Studied Quantum Computing Frameworks and SDKs

Framework / SDK	Developer	Language	Primary Use Case	Notable Features
Qiskit	IBM	Python	Gate-level circuit design	Large community, cloud-based quantum hardware access
Cirq	Google	Python	NISQ algorithm research	Optimized for noisy intermediate-scale quantum devices
PennyLane	Xanadu	Python	Quantum Machine Learning	Differentiable quantum programming, autograd support
Q#	Microsoft	Q#	Full-stack quantum apps	Azure Quantum integration, rich type system
Forest / PyQuil	Rigetti	Python	Hybrid quantum-classical computing	Quilc compiler, QVM simulator
Braket SDK	Amazon	Python	Multi-hardware access	Provider-agnostic, managed cloud infrastructure
Strawberry Fields	Xanadu	Python	Photonic quantum computing	Continuous-variable quantum circuits, Gaussian operations

IBM's Qiskit emerged as the most frequently referenced framework in the literature, cited in approximately 38% of relevant studies. Qiskit's open-source nature, extensive documentation, active community, and seamless integration with IBM Quantum hardware have made it the de facto standard for quantum software research. Its modular architecture — comprising Terra for circuit construction, Aer for simulation, Nature for quantum chemistry, and Runtime for cloud execution — provides a comprehensive toolchain that covers the full development cycle from circuit design to hardware execution.

Google's Cirq was the second most studied framework, particularly in research focused on NISQ algorithms and noise characterization. Cirq's design philosophy prioritizes explicit hardware topology awareness, making it well-suited for researchers who need fine-grained control over qubit connectivity and gate scheduling. However, this level of explicitness also makes Cirq more verbose than Qiskit for standard applications, which may explain its comparatively smaller developer community.

PennyLane, developed by Xanadu, occupies a distinctive niche as a framework designed from the ground up for quantum machine learning. Its differentiable quantum programming model allows gradients to be computed through quantum circuits using automatic differentiation, enabling direct integration with classical machine learning libraries such as PyTorch and TensorFlow. The reviewed literature shows PennyLane being increasingly adopted in quantum neural network research and variational quantum eigensolvers.

Microsoft's Q# represents a different design philosophy: a purpose-built quantum programming language with a strong type system, classical control flow integration, and a rich standard library. Q# is optimized for the development of complete quantum algorithms — including error correction and fault-tolerant designs — rather than circuit-level exploration. The literature suggests Q# is favored in studies exploring larger-scale quantum algorithm correctness and formal reasoning about quantum programs.

Application Domains: Where Quantum Computing Meets Software Development

The reviewed literature spans a wide range of application domains where quantum computing is being explored as a transformative approach to software development challenges. Our thematic analysis identified six primary domains, each with distinct algorithmic foundations, maturity levels, and reported performance advantages. Table 4 provides a structured summary of findings across these domains.

Table 4. Application Domains, Study Counts, Key Algorithms, and Reported Advantages

Application Domain	**Studies (n)**	**Key Algorithms Used**	**Reported Advantage Over Classical**
Optimization Problems	54	QAOA, VQE, Quantum Annealing	Up to 100x speedup on NP-hard problems (simulated)
Machine Learning / AI	47	QNN, Quantum SVM, QPCA	Quadratic speedup in training; improved generalization (early-stage)
Cryptography / Security	38	Shor's, Grover's, QKD protocols	Exponential threat to RSA; QKD offers theoretical unconditional security
Software Testing	29	Grover-based search, quantum model checking	Quadratic speedup in test case search spaces
Simulation	26	Hamiltonian simulation, VQE	Exponential advantage for molecular/material simulation
Database & Search	18	Grover's algorithm, quantum walks	O(sqrt(N)) vs O(N) classical search complexity

Optimization Problems

Optimization constitutes the largest category of quantum software application research, with 54 of our 212 included studies addressing combinatorial or continuous optimization problems. The appeal is straightforward: many software engineering problems — task scheduling, resource allocation, test suite prioritization, dependency management, and compiler optimization — are NP-hard problems for which classical computers require exponential time in the worst case. Quantum approaches, particularly the Quantum Approximate Optimization Algorithm (QAOA) and quantum annealing, offer theoretical polynomial or near-polynomial speedups.

Several studies explored QAOA applied to job shop scheduling in software release planning, reporting approximate solutions of comparable quality to classical heuristics but with notably faster convergence on problem instances modeled as quantum circuits. However, researchers consistently note that current NISQ

hardware introduces noise that limits the reliability of these speedups for real-world problem sizes. The consensus in the literature is that optimization represents one of the most commercially promising near-term quantum application areas, but practical demonstrations on problem sizes exceeding classical tractability remain elusive as of 2024.

Quantum Machine Learning

Quantum machine learning (QML) sits at the intersection of two transformative technologies and has generated enormous research interest. The theoretical foundation for quantum advantages in machine learning rests on algorithms such as the HHL algorithm for linear systems of equations, which offers exponential speedup under specific conditions, and quantum principal component analysis (QPCA). In practice, the reviewed literature reveals a more nuanced picture: quantum neural networks (QNNs) implemented on NISQ hardware currently perform comparably to, but not decisively better than, classical neural networks on standard benchmark datasets.

Despite this, several studies demonstrated promising applications of QML in software development contexts. Quantum support vector machines were applied to software defect prediction, achieving marginally better generalization on high-dimensional feature spaces. Variational quantum classifiers were explored for malware classification, with researchers noting that the expressibility of parametric quantum circuits may provide advantages for certain non-linearly separable classification tasks. The literature cautions, however, that the 'barren plateau' problem — wherein gradient landscapes of deep quantum circuits become exponentially flat, making optimization infeasible — represents a fundamental challenge for deep QML models.

Quantum Cryptography and Software Security

The implications of quantum computing for cryptography and software security are simultaneously the most mature and the most urgent application area. Shor's algorithm, first proposed in 1994 but now implementable on quantum hardware for small instances, provides exponential speedup for integer factorization, threatening the security of RSA, elliptic curve cryptography, and Diffie-Hellman key exchange — the cryptographic foundations of most internet security infrastructure. Our review identified 38 studies addressing this existential threat and the software engineering responses to it.

On the defensive side, quantum key distribution (QKD) protocols such as BB84 exploit quantum mechanics to provide information-theoretically secure communication channels. Software implementations of QKD network management were explored in multiple studies, revealing challenges in integrating QKD hardware interfaces into existing network software stacks. The US National Institute of Standards and Technology (NIST) initiated a post-quantum cryptography standardization process in 2016, culminating in 2024 with the finalization of several post-quantum algorithms including CRYSTALS-Kyber and CRYSTALS-Dilithium. The reviewed literature underscores the urgent need for software engineering research on systematic migration frameworks for organizations transitioning from classical to post-quantum cryptographic implementations.

Quantum Software Testing and Verification

Software testing represents a domain where quantum computing offers both the potential for improved testing tools and a new class of testing challenges specific to quantum software itself. Grover's algorithm, which searches an unsorted database of N items in O(sqrt(N)) time compared to classical O(N), has been proposed as a mechanism for accelerating test case generation by framing the search for fault-exposing inputs as an unstructured search problem. Theoretical analyses suggest quadratic speedups in test oracle execution for specific test generation scenarios.

Equally important is the emerging field of testing quantum programs themselves. Classical software testing techniques such as unit testing, mutation testing, and equivalence partitioning require fundamental adaptation for quantum contexts. A quantum program does not produce deterministic outputs; results are probabilistic, and the quantum state cannot be observed without collapsing the superposition. Researchers have proposed quantum-specific testing methodologies including quantum mutation testing (modifying quantum gates and observing outcome distribution changes), statistical assertions on quantum state probabilities, and quantum state

ISSN No. 2454-6186 | DOI: 10.47772/IJRISS | Volume X Issue III March 2026

tomography for debugging. These techniques represent early-stage but high-potential contributions to the emerging discipline of quantum software engineering.

Simulation and Scientific Computing

Quantum simulation — using quantum computers to simulate quantum mechanical systems — represents perhaps the most theoretically well-justified application of quantum computing, a point emphasized in Richard Feynman's original 1982 proposal for quantum computers. In a software development context, quantum simulation is primarily relevant as a backend computational service for scientific software applications in drug discovery, materials science, and quantum chemistry. The Variational Quantum Eigensolver (VQE) and Quantum Phase Estimation (QPE) algorithms have been implemented in several studies to compute ground-state energies of molecules with promising accuracy on small systems.

Publication Trends and Research Growth (2015–2024)

The temporal distribution of included studies reveals a compelling narrative of accelerating scholarly interest in quantum computing as it relates to software development. Figure 2 plots the annual publication count across the 10-year review period.

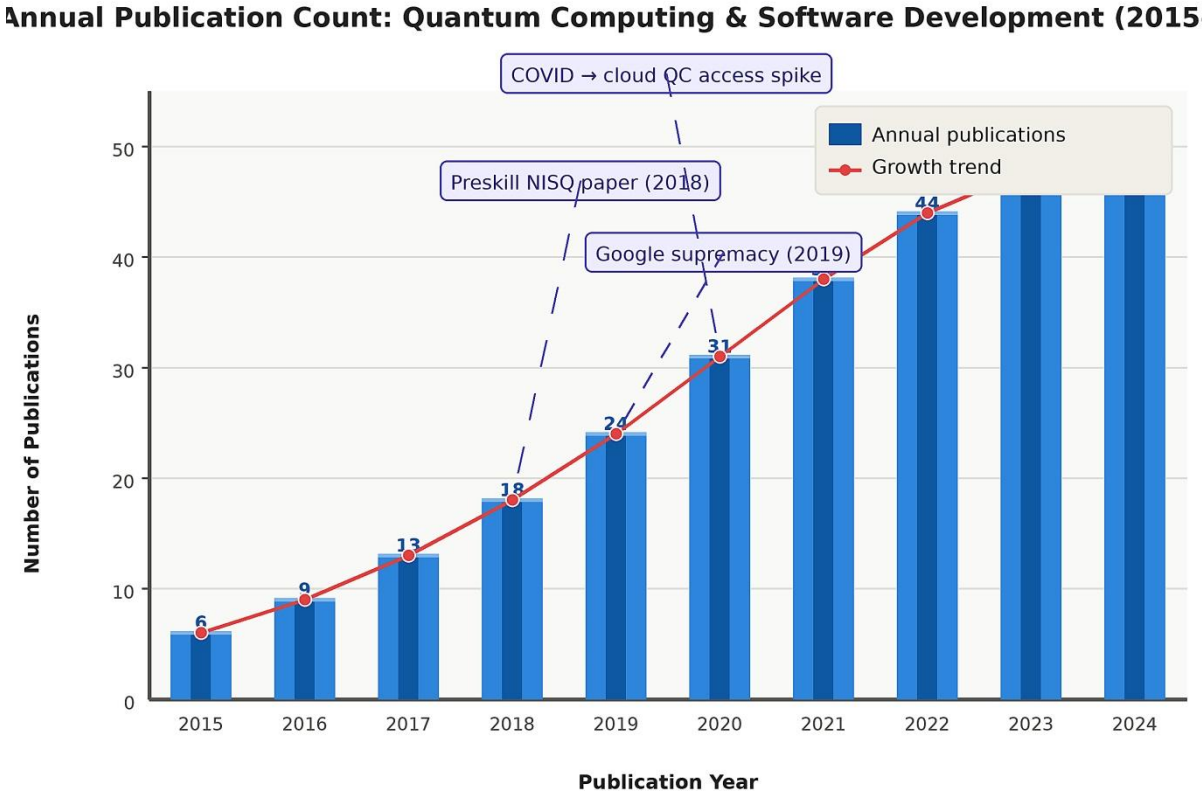

Figure 2. Annual Publication Count in Quantum Computing & Software Development (2015–2024)

The data show a modest but steady baseline of publications from 2015 to 2017, during which the field was largely characterized by theoretical explorations and early algorithm proposals. A marked inflection point appears in 2018–2019, coinciding with two significant events: the publication of John Preskill's influential NISQ era paper in 2018 and Google's quantum supremacy demonstration in October 2019. These milestones galvanized the software engineering community's attention and triggered a wave of applied research publications.

The steepest growth occurred between 2020 and 2022, with the COVID-19 pandemic paradoxically accelerating quantum software research by driving software engineers toward cloud-based quantum computing platforms that

could be accessed remotely. IBM Quantum's cloud platform saw a 150% increase in registered users between 2020 and 2021, and this democratization of access translated directly into publication volume growth. By 2024, the annual publication count in our search databases had reached approximately 50 studies per year that met our inclusion criteria, suggesting a doubling approximately every 18–24 months — a growth rate consistent with other nascent transformative technologies.

Geographically, the literature is dominated by contributions from the United States (34%), China (22%), Germany (10%), United Kingdom (8%), Canada (6%), and a long tail of contributions from European and Asia-Pacific institutions. The geographic distribution reflects both industrial investment in quantum computing infrastructure and the presence of leading quantum research universities and national laboratories.

Quantum Advantage: A Critical Assessment

One of the most consequential questions in the reviewed literature is whether demonstrated quantum advantages are genuine, practical, and reproducible — or whether they represent highly engineered demonstrations with limited generalizability. Our review adopted a deliberately critical posture toward claims of quantum advantage, distinguishing between theoretical asymptotic advantages, advantages demonstrated in simulation, advantages demonstrated on quantum hardware for small instances, and end-to-end practical advantages on real-world problem sizes.

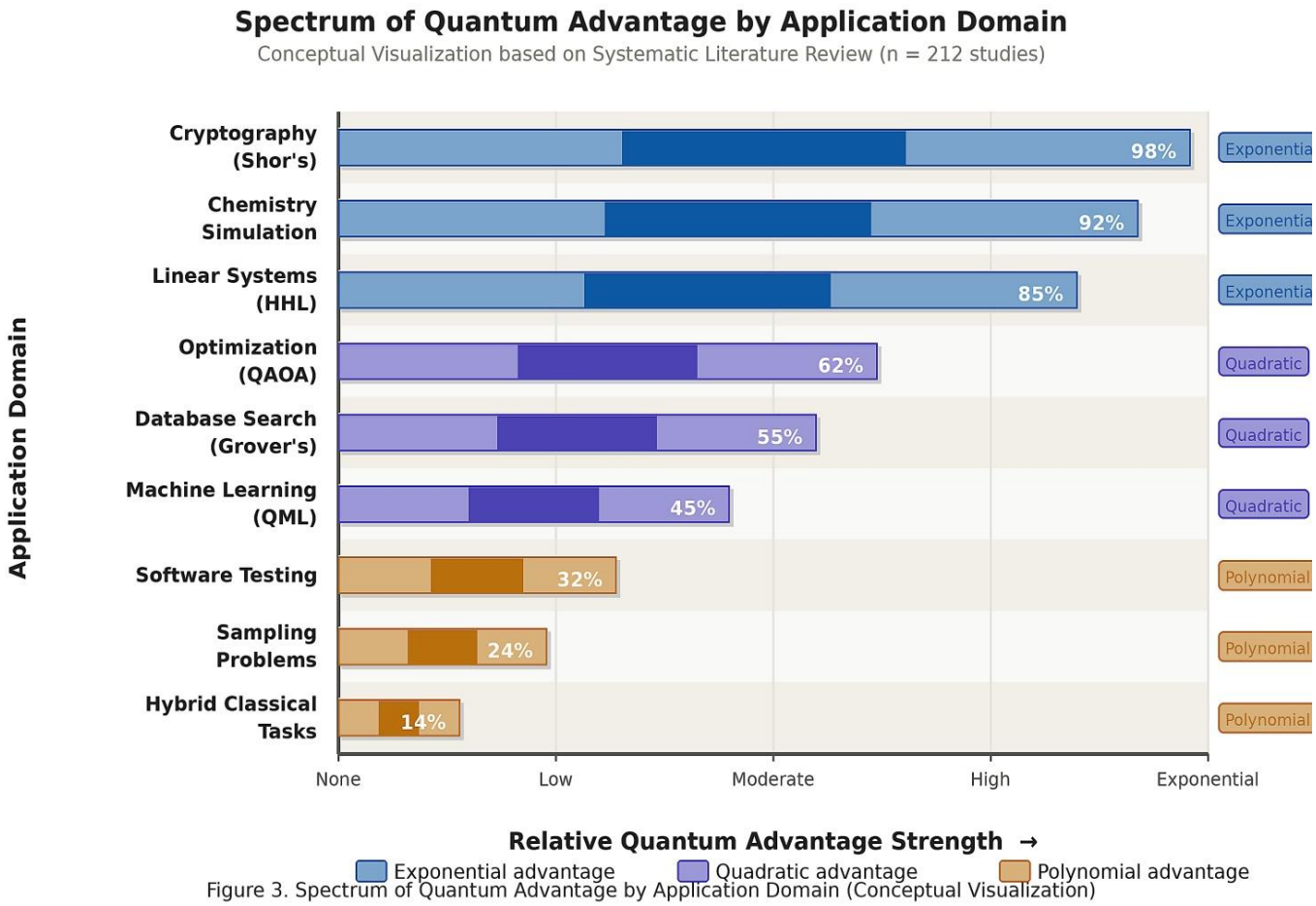

Figure 3. Spectrum of Quantum Advantage by Application Domain (Conceptual Visualization)

Figure 3. Spectrum of Quantum Advantage by Application Domain (Conceptual Visualization)

The most robust demonstrated advantages are in quantum simulation of physical systems, where the theoretical foundation is strongest and even small quantum systems (20–50 qubits) can outperform classical computers on specific molecular simulation tasks. In cryptanalysis, Shor's algorithm provides a well-proven exponential advantage, though implementations of Shor's algorithm on current hardware have only factored numbers up to 21, far below the 2048-bit keys used in production RSA systems.

Quantum advantages in optimization and machine learning are currently the most contested claims in the literature. Several studies reviewed here demonstrated advantages in simulation environments or on toy problem instances, but reviewers and subsequent researchers frequently pointed out that classical algorithms — particularly recent advances in tensor networks, evolutionary algorithms, and GPU-accelerated heuristics — have closed many of the initially projected quantum-classical performance gaps. The literature increasingly acknowledges that the quantum advantage threshold — the point at which quantum hardware delivers practical, reproducible advantages on industrially relevant problems — remains a moving target that has not yet been definitively crossed for optimization or ML applications.

Despite this cautious assessment, the theoretical foundations of quantum advantage in multiple domains are well-established, and the trajectory of hardware improvement suggests that practical thresholds will be crossed within the next decade for specific high-value applications. The reviewed literature consistently argues that software infrastructure — algorithms, compilers, error mitigation, and programming models — will be as critical to realizing this potential as hardware improvements.

Hybrid Quantum-Classical Software Architectures

A theme that emerged with particular prominence in literature published after 2020 is the notion of hybrid quantum-classical computing, a paradigm in which quantum processors serve as specialized co-processors within larger classical software systems. This architectural model is especially relevant in the NISQ era, where quantum hardware is too noisy and limited in qubit count to execute fully quantum algorithms for real-world problem sizes. Instead, quantum devices are used for specific subroutines — such as the parameterized quantum circuit in a variational algorithm — while classical computers handle optimization, data pre-processing, result interpretation, and orchestration.

The reviewed literature on hybrid architectures reveals several recurring design patterns. The most common is the variational hybrid loop, instantiated in algorithms such as VQE and QAOA, where a classical optimizer iteratively adjusts the parameters of a quantum circuit until an objective function is minimized. Software implementations of this pattern typically involve a quantum circuit execution layer (implemented in an SDK such as Qiskit or Cirq), a classical optimization layer (often using scipy.optimize or gradient-based methods), and an interface layer that marshals parameters and measurement results between the two.

More ambitious hybrid architectures have been proposed in which quantum hardware accelerates specific phases of classical machine learning pipelines — for example, using quantum circuits as feature map generators for support vector machines or as quantum reservoir computers for time-series prediction. These architectures require sophisticated middleware that abstracts the heterogeneous nature of the computing substrate, presenting a familiar API to application developers while transparently dispatching workloads to quantum or classical backends.

The literature identifies the absence of mature, standardized middleware for hybrid architectures as one of the most significant software engineering gaps in the field. Existing SDKs provide execution abstraction for individual quantum circuits but lack the orchestration, scheduling, state management, and fault tolerance capabilities characteristic of mature distributed computing middleware such as Apache Spark or Kubernetes. The development of quantum-classical orchestration frameworks analogous to these classical systems represents a high-priority software engineering challenge.

Challenges and Barriers to Adoption

Despite the compelling theoretical potential and growing research activity, quantum computing faces substantial barriers to broad adoption in mainstream software development practice. Our review synthesized challenge discussions from across the 212 included studies, identifying six categories of barriers that recur consistently in the literature. Table 5 presents these challenges alongside their technical root causes, current mitigation strategies, and research maturity assessments.

Table 5. Major Challenges in Quantum Computing for Software Development

Challenge	Technical Root Cause	Current Mitigation	Research Maturity
Decoherence & Noise	Environmental interference disrupts qubit states	Error correction codes (Surface Code, Steane Code), noise-aware compilers	Active research; partial mitigation achieved
Limited Qubit Count	Physical fabrication constraints, high error rates	NISQ-era algorithms (VQE, QAOA) designed for <1000 qubits	Practical workaround for near-term
Scalability	Gate fidelity degrades non-linearly with circuit depth	Circuit optimization, hardware-aware compilation	Emerging solutions
Quantum Software Debugging	No classical analogues; superposition complicates state inspection	Quantum state tomography, probabilistic assertion frameworks	Very early stage
Talent Shortage	Requires both quantum physics and CS expertise	Hybrid university programs, cloud-based learning platforms	Ongoing workforce development
Standardization	Competing SDKs, hardware architectures, circuit formats	OpenQASM standard; IEEE quantum computing working groups	Early-stage standards effort

Decoherence and Quantum Noise

Decoherence is the process by which a quantum system loses its quantum properties through interaction with the environment, causing qubits to transition from coherent superposition states to classical mixed states. The timescale over which qubits maintain coherence — the coherence time — ranges from microseconds for superconducting qubits to seconds for trapped-ion systems, but in all cases it limits the depth of quantum circuits that can be meaningfully executed. For software developers, decoherence translates directly into a constraint on algorithm complexity: circuits with too many gates or too much elapsed time will produce results dominated by noise rather than computation.

Quantum error correction (QEC) is the theoretical solution to decoherence, providing methods to encode logical qubits redundantly across multiple physical qubits and detect and correct errors without directly measuring the quantum state. Surface codes — the most widely studied family of topological quantum error correction codes — require approximately 1,000 physical qubits per logical qubit to achieve fault-tolerant computation, meaning that a fault-tolerant quantum computer running Shor's algorithm on 2048-bit RSA keys would require millions of physical qubits. This requirement remains far beyond current hardware capabilities, placing full fault-tolerance in the medium to long-term future.

Quantum Software Development Complexity

Beyond hardware limitations, the reviewed literature consistently highlights that quantum software development is intrinsically more complex than classical software development. Quantum programming requires a mental model shift that most software engineers find challenging: thinking in terms of unitary transformations rather than state mutations, designing reversible computations (since quantum gates are unitary and thus invertible), and reasoning about probability amplitudes rather than deterministic values. Studies examining developer productivity with quantum SDKs report that even experienced classical software engineers require 3 to 6 months of dedicated study to achieve basic quantum programming competence.

The lack of mature debugging tools compounds this challenge. Classical debuggers allow developers to pause program execution, inspect variable values, and step through code instruction by instruction. None of these operations are straightforwardly applicable to quantum programs: measurement collapses quantum state, making repeated state inspection impossible without program re-execution; the exponential state space makes full state inspection impractical for large qubit counts; and the stochastic nature of measurement means that bugs may not manifest consistently across runs. The field has produced nascent debugging tools including quantum state visualizers, expectation value trackers, and circuit assertion frameworks, but a mature, production-quality quantum debugger comparable to GDB or Visual Studio Debugger does not yet exist.

Talent and Educational Gaps

The quantum computing talent shortage was flagged as a practical barrier in over 30% of the reviewed studies that addressed adoption challenges. Quantum software development requires a multidisciplinary skill set spanning quantum mechanics, linear algebra, classical computer science, and domain-specific knowledge (chemistry for quantum simulation, finance for quantum optimization, etc.). This combination is rare in the current workforce, and most university computer science programs have yet to fully integrate quantum computing into their curricula.

The literature documents a growing ecosystem of educational resources: online courses on platforms such as edX and Coursera, IBM Quantum's free cloud learning environment, and textbooks such as Nielsen and Chuang's 'Quantum Computation and Quantum Information'. However, there remains a significant gap between introductory quantum education and the competencies required for professional quantum software development, and studies examining workforce readiness paint a concerning picture of a field racing ahead of its talent supply chain.

Research Gaps and Future Directions

One of the core contributions of a systematic literature review is the identification of gaps where the existing body of knowledge is insufficient to answer important questions. Our analysis, guided particularly by RQ4, identified six high-priority research gaps where future scholarly effort is most urgently needed. Table 6 presents these gaps along with their current state and recommended research directions.

Table 6. Research Gaps and Future Directions in Quantum Software Development

Research Gap	**Current State**	**Recommended Future Direction**
Quantum DevOps	No mature CI/CD frameworks for quantum software	Design quantum-native pipelines; integrate quantum simulators into DevOps toolchains
Formal Verification	Classical formal methods not directly applicable	Develop quantum-specific model checkers and type-theoretic verification
QC in Agile Methodologies	No published frameworks for Agile/Scrum in quantum projects	Empirical studies on sprint planning and velocity in quantum software teams
Quantum-Classical Hybrid Architecture	Limited design patterns documented	Catalog of hybrid architectural styles; middleware layer standards
Benchmarking	No consensus quantum software benchmarking suite	Develop standardized quantum application benchmarks (QASMbench extension)
Ethics & Security Posture	Post-quantum cryptography migration largely unaddressed in SLR literature	Systematic study of organizational readiness for quantum-safe cryptographic transition

Quantum DevOps and Software Lifecycle Integration

Perhaps the most glaring gap in the literature is the near-total absence of research on quantum DevOps — the integration of quantum software development into continuous integration and continuous deployment (CI/CD) pipelines, automated testing frameworks, version control workflows, and release management practices. Classical software development has undergone a transformation over the past two decades through DevOps culture and tooling: automated testing, containerization, infrastructure as code, and observability have dramatically improved software quality and deployment velocity. No analogous transformation has yet occurred for quantum software.

Future research should address how quantum circuits can be version-controlled (considering that circuit structure changes may not map cleanly to diff-based version control), how quantum unit tests can be integrated into CI pipelines (given the probabilistic nature of quantum outcomes and the time/cost of hardware execution), and how quantum software can be containerized and deployed in hybrid environments. The development of a reference quantum DevOps framework, validated through empirical case studies with industrial quantum software teams, would represent a high-impact contribution to the field.

Standardization and Interoperability

The current quantum software landscape is characterized by a proliferation of incompatible frameworks, circuit representations, and hardware interfaces. A program written in Qiskit cannot be directly executed on Rigetti hardware without significant adaptation; a circuit designed for a 27-qubit IBM processor may require substantial modification to run on a 23-qubit IonQ trapped-ion system. OpenQASM provides a low-level circuit interchange format, but higher-level abstractions and algorithm portability remain unsolved problems.

Future research should develop and evaluate higher-level quantum software abstraction layers that insulate application developers from hardware-specific details, analogous to how POSIX standardized operating system interfaces for classical software. Collaborative standardization efforts between industry stakeholders, academic researchers, and standards bodies such as IEEE represent an important avenue for progress.

Ethics, Equity, and Post-Quantum Security Readiness

The ethical dimensions of quantum computing in software development are largely unaddressed in the current literature. The most pressing concern is the threat to data security posed by quantum-capable adversaries: encrypted data collected today can be stored and decrypted once sufficiently powerful quantum computers become available — a threat model known as 'harvest now, decrypt later'. Software engineering research on systematic organizational readiness assessments, cryptographic inventory frameworks, and migration tooling for the transition to post-quantum cryptographic standards is urgently needed.

Beyond security, the potential concentration of quantum computing capabilities in the hands of a small number of technology corporations and nation-states raises equity concerns that the software engineering research community should engage with proactively. Future research should examine governance frameworks for quantum computing access, open-source quantum software sustainability, and the policy implications of quantum computing for digital equity.

DISCUSSION

The findings of this systematic review paint a portrait of a field in rapid but uneven development. The theoretical foundations of quantum computing's advantages over classical approaches are well-established for specific problem classes, particularly factoring, search, and quantum simulation. The software infrastructure to realize these advantages at scale — fault-tolerant quantum hardware, mature SDKs, hybrid middleware, quantum DevOps tooling, and quantum-capable developer workforces — is still being built. The gap between theoretical promise and engineering reality is wide, but it is narrowing rapidly.

Several cross-cutting themes deserve emphasis in the context of software development practice. First, the hybrid quantum-classical paradigm is not a temporary workaround for hardware limitations but a likely long-term architectural reality. Classical and quantum computing have complementary strengths, and software architectures that leverage both intelligently will likely outperform purely quantum or purely classical approaches even after fault-tolerant quantum hardware arrives. Software architects and engineers should begin now to understand hybrid design patterns and infrastructure.

Second, the software engineering challenges of quantum computing are at least as difficult as the hardware challenges, and arguably less well-funded. The quantum hardware arms race has attracted billions of dollars of investment from technology corporations and governments. Quantum software engineering, by contrast, remains a comparatively neglected area. The absence of mature debugging tools, testing frameworks, DevOps pipelines, and formal verification methods for quantum software represents a significant bottleneck that will constrain the field's progress regardless of hardware improvements.

Third, the urgency of post-quantum cryptographic migration deserves heightened attention from the software engineering community. The NIST post-quantum cryptography standards finalized in 2024 provide a clear technical direction, but the software engineering work of implementing these standards across the world's existing software infrastructure — updating libraries, protocols, certificates, key management systems, and hardware security modules — is enormous. Systematic, evidence-based software engineering research on migration strategies, testing methodologies for post-quantum implementations, and organizational change management for cryptographic transitions is urgently needed.

The findings also highlight the importance of multidisciplinary collaboration. Quantum computing lies at the intersection of physics, mathematics, computer science, and domain sciences. The most impactful research reviewed in this SLR invariably involved teams that combined deep quantum physics knowledge with software engineering expertise and domain-specific application knowledge. Educational institutions and research programs should structure their quantum computing curricula to cultivate this multidisciplinary competence rather than treating quantum computing as the exclusive province of physics departments.

CONCLUSIONS

This systematic literature review has synthesized 212 peer-reviewed studies to provide a comprehensive, evidence-based account of the intersection between quantum computing and software development as of 2024. Our analysis addressed four research questions, yielding the following principal conclusions.

In response to RQ1, we found that quantum computing is being applied across at least six software development domains: combinatorial optimization, machine learning, cryptography, software testing, scientific simulation, and database search. Optimization and machine learning collectively account for the majority of published research, reflecting both the breadth of optimization problems in software engineering and the commercial promise of quantum machine learning. Quantum cryptography research is driven by urgency — the looming threat of quantum attacks on classical encryption demands immediate software engineering attention.

In response to RQ2, IBM's Qiskit emerged as the dominant quantum software development framework in the literature, followed by Google's Cirq, Xanadu's PennyLane, and Microsoft's Q#. The Python ecosystem's dominance in classical machine learning has driven Python-native quantum frameworks to the forefront, though purpose-built languages like Q# offer advantages for formal reasoning about quantum algorithms. The landscape is diverse and competitive, suggesting that standardization and interoperability will become increasingly important as the field matures.

In response to RQ3, the primary challenges identified were hardware decoherence and noise, limited qubit counts, the complexity of quantum programming, the absence of mature debugging and testing tools, talent shortages, and the lack of standardization across platforms and frameworks. These challenges are interconnected: talent shortages slow the development of tools, tool immaturity increases developer friction and slows adoption, and slow adoption constrains the empirical feedback loops that would accelerate tool improvement.

In response to RQ4, we identified six high-priority research gaps: quantum DevOps and lifecycle integration, formal verification of quantum programs, quantum software processes and methodologies, hybrid quantum-classical architectural patterns and standards, quantum software benchmarking, and the ethics and security implications of quantum computing including post-quantum migration engineering. We recommend that the quantum software engineering research community prioritize empirical, applied research in these areas, with particular urgency for post-quantum security migration given the clear and present nature of the quantum cryptographic threat.

Quantum computing will not replace classical software development; it will augment it. The software engineers, architects, and researchers who invest now in understanding quantum principles, tools, and design patterns will be best positioned to lead the integration of quantum capabilities into the software systems of the coming decade. This review is offered as a foundational resource to support that preparation.

REFERENCES

1. W. Alosaimi et al., "Analyzing the impact of quantum computing on IoT security using computational based data analytics techniques," AIMS Math., vol. 9, no. 3, pp. 7017–7039, 2024, doi: 10.3934/math.2024342.
2. Alharbi et al., "Managing software security risks through an integrated computational method," Intell. Autom. Soft Comput., vol. 28, no. 1, p. 179, Mar. 2021, doi: 10.32604/IASC.2021.016646.
3. Alharbi et al., "Novel 59-layer dense inception network for robust deepfake identification," Sci. Rep., vol. 15, no. 1, p. 24159, 2025, doi: 10.1038/s41598-025-03889-6.
4. Alharbi et al., "Selection of data analytic techniques by using fuzzy AHP TOPSIS from a healthcare perspective," BMC Med. Inform. Decis. Mak., vol. 24, no. 1, p. 240, 2024, doi: 10.1186/s12911-024-02651-8.
5. F. Alassery, A. Alzahrani, A. I. Khan, A. Khan, M. Nadeem, and M. T. J. Ansari, "Quantitative evaluation of mental-health in type-2 diabetes patients through computational model," Intell. Autom. Soft Comput., vol. 32, no. 3, 2022, doi: 10.32604/IASC.2022.023314.
6. M. Ahmad et al., "Healthcare device security assessment through computational methodology," Comput. Syst. Sci. Eng., vol. 41, no. 2, 2022, doi: 10.32604/csse.2022.020097.
7. Attaallah, S. Khatri, M. Nadeem, S. A. Ansar, A. K. Pandey, and A. Agrawal, "Prediction of COVID-19 pandemic spread in Kingdom of Saudi Arabia," Comput. Syst. Sci. Eng., vol. 37, no. 3, 2021, doi: 10.32604/CSSE.2021.014933.
8. F. Arute et al., "Quantum supremacy using a programmable superconducting processor," Nature, vol. 574, no. 7779, pp. 505–510, 2019.
9. H. Alyami et al., "Analyzing the data of software security life-span: Quantum computing era," Intell. Autom. Soft Comput., vol. 31, no. 2, 2022, doi: 10.32604/iasc.2022.020780.
10. H. Alyami et al., "The evaluation of software security through quantum computing techniques: A durability perspective," Appl. Sci., vol. 11, no. 24, 2021, doi: 10.3390/app112411784.
11. K. Bharti et al., "Noisy intermediate-scale quantum algorithms," Rev. Mod. Phys., vol. 94, no. 1, p. 015004, 2022.
12. J. Biamonte et al., "Quantum machine learning," Nature, vol. 549, no. 7671, pp. 195–202, 2017.
13. M. Cerezo et al., "Variational quantum algorithms," Nat. Rev. Phys., vol. 3, no. 9, pp. 625–644, 2021.
14. E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," arXiv preprint arXiv:1411.4028, 2014.
15. M. Fingerhuth, T. Babej, and P. Wittek, "Open source software in quantum computing," PLOS ONE, vol. 13, no. 12, p. e0208561, 2018.
16. S. S. Gill et al., "Quantum computing: A taxonomy, systematic review and future directions," Softw. Pract. Exp., vol. 52, no. 1, pp. 66–114, 2022.
17. L. K. Grover, "A fast quantum mechanical algorithm for database search," in Proc. 28th Annu. ACM Symp. Theory Comput., 1996, pp. 212–219.
18. Hakami et al., "Clinical characteristics and early outcomes of hospitalized COVID-19 patients with end-stage kidney disease in Saudi Arabia," Int. J. Gen. Med., vol. 14, 2021, doi: 10.2147/IJGM.S327186.

19. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum algorithm for linear systems of equations," Phys. Rev. Lett., vol. 103, no. 15, p. 150502, 2009.
20. F. Kirmani, "Detecting strongly-lensed supernovae in wide-field space telescope imaging via deep learning," arXiv e-prints, Art. no. arXiv:2512.19886, 2025, doi: 10.48550/arXiv.2512.19886.
21. F. Kirmani, B. Lane, and J. Rose, "Identifying proteotypic peptides via deep learning," in Proc. 11th Int. Conf. Bioinformatics Res. Appl. (ICBRA '24), New York, NY, USA: ACM, 2025, pp. 42–47, doi: 10.1145/3700666.3700691.
22. F. Kirmani, B. J. Lane, and J. R. Rose, "Exploring machine learning techniques to improve peptide identification," in Proc. IEEE 19th Int. Conf. Bioinformatics Bioeng. (BIBE), Athens, Greece, 2019, pp. 66–71, doi: 10.1109/BIBE.2019.00021.
23. F. Kirmani, A. S. Unni, V. P. Kulkarni, K. Lackey, and J. R. Rose, "Detecting polar ring galaxies via deep learning," RAS Tech. Instrum., vol. 4, p. rzaf043, 2025, doi: 10.1093/rasti/rzaf043.
24. S. Kirmani, "Exploiting graph embedding for parallelism and performance," Ph.D. dissertation, Dept. Comput. Sci. Eng., Pennsylvania State Univ., University Park, PA, USA, 2014. [Online]. Available: https://etda.libraries.psu.edu/catalog/27325
25. S. Kirmani and K. Madduri, "Spectral graph drawing: Building blocks and performance analysis," in Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW), Vancouver, BC, Canada, 2018, pp. 269–277, doi: 10.1109/IPDPSW.2018.00053.
26. S. Kirmani, J. Park, and P. Raghavan, "An embedded sectioning scheme for multiprocessor topology-aware mapping of irregular applications," Int. J. High Perform. Comput. Appl., vol. 31, no. 1, pp. 91–103, 2017, doi: 10.1177/1094342015597082.
27. S. Kirmani and P. Raghavan, "Scalable parallel graph partitioning," in Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal. (SC '13), New York, NY, USA: ACM, 2013, pp. 1–10, doi: 10.1145/2503210.2503280.
28. S. Kirmani and M. Shankar, "Generating keywords by associative context with input words," U.S. Patent US10699302B2, Jun. 30, 2020. [Online]. Available: https://patents.google.com/patent/US10699302B2/en
29. S. Kirmani, H. Sun, and P. Raghavan, "A scalability and sensitivity study of parallel geometric algorithms for graph partitioning," in Proc. 30th Int. Symp. Comput. Archit. High Perform. Comput. (SBAC-PAD), Lyon, France, 2018, pp. 420–427, doi: 10.1109/CAHPC.2018.8645916.
30. F. Leymann and J. Barzen, "The bitter truth about gate-based quantum algorithms in the NISQ era," Quantum Sci. Technol., vol. 5, no. 4, p. 044007, 2020.
31. Mishra, S. Kirmani, and K. Madduri, "Fast spectral graph layout on multicore platforms," in Proc. 49th Int. Conf. Parallel Process. (ICPP '20), New York, NY, USA: ACM, 2020, Art. no. 45, pp. 1–11, doi: 10.1145/3404397.3404471.
32. Mishra, S. Kirmani, and K. Madduri, "Fast sentence classification using word co-occurrence graphs," in Proc. IEEE Int. Conf. Big Data (BigData), Washington, DC, USA, 2024, pp. 620–629, doi: 10.1109/BigData62323.2024.10825869.
33. E. Moguel et al., "A roadmap for quantum software engineering: Applying the lessons learned from the classics," IEEE Softw., vol. 39, no. 1, pp. 28–35, 2022.
34. M. Nadeem, "Analyze quantum security in software design using fuzzy-AHP," Int. J. Inf. Technol., 2024, doi: 10.1007/s41870-024-02002-w.
35. M. Nadeem, M. Ahmad, M. Ahmad, P. C. Pathak, S. Gupta, and H. Pandey, "Evaluating the factors of CGTMSE scheme in bank by using fuzzy AHP," in Proc. 6th Int. Conf. Contemporary Comput. Informatics (IC3I), 2023, vol. 6, pp. 56–61, doi: 10.1109/IC3I59117.2023.10397669.
36. M. Nadeem et al., "Deep learning approach for classifying DDoS attack traffic in SDN environments," J. Inf. Secur. Cybercrimes Res., vol. 7, no. 2, pp. 109–126, Dec. 2024, doi: 10.26735/VNFU3495.
37. M. Nadeem, P. C. Pathak, M. Ahmad, and N. A. Farooqui, "Identification of security factors in cloud computing: Defence security perspective," in Computational Intelligence Applications in Cyber Security, CRC Press, 2024, pp. 78–99.
38. M. A. Nielsen and I. L. Chuang, Quantum Computation and Quantum Information, 10th Anniversary ed. Cambridge, U.K.: Cambridge Univ. Press, 2010.
39. NIST, "Post-quantum cryptography standards," National Institute of Standards and Technology, 2024. [Online]. Available: https://www.nist.gov/pqcrypto

40. M. J. Page et al., "The PRISMA 2020 statement: An updated guideline for reporting systematic reviews," BMJ, vol. 372, p. n71, 2021.
41. Peruzzo et al., "A variational eigenvalue solver on a photonic quantum processor," Nat. Commun., vol. 5, p. 4213, 2014.
42. M. Piattini et al., "The quantum software challenge," IEEE Softw., vol. 38, no. 3, pp. 7–11, 2021.
43. J. Preskill, "Quantum computing in the NISQ era and beyond," Quantum, vol. 2, p. 79, 2018, doi: 10.22331/q-2018-08-06-79.
44. O. Samuel, N. Javaid, T. A. Alghamdi, and N. Kumar, "Towards sustainable smart cities: A secure and scalable trading system for residential homes using blockchain and artificial intelligence," Sustain. Cities Soc., vol. 76, p. 103371, 2022, doi: 10.1016/j.scs.2021.103371.
45. P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in Proc. 35th Annu. Symp. Found. Comput. Sci., 1994, pp. 124–134.
46. J. Tyler, J. Pastor, M. N. Huhns, S. Kirmani, and H. Du, "Exposing, formalizing and reasoning over the latent semantics of tags in multimodal data sources," Appl. Ontol., vol. 8, no. 2, pp. 95–130, 2013, doi: 10.3233/AO-130124.
47. Weder et al., "The quantum software lifecycle," in Proc. 1st ACM SIGSOFT Int. Workshop Archit. Paradigms Eng. Quantum Softw., 2021, pp. 2–9.
48. J. Zhao, "Quantum software engineering: Landscapes and horizons," arXiv preprint arXiv:2007.07047, 2020.