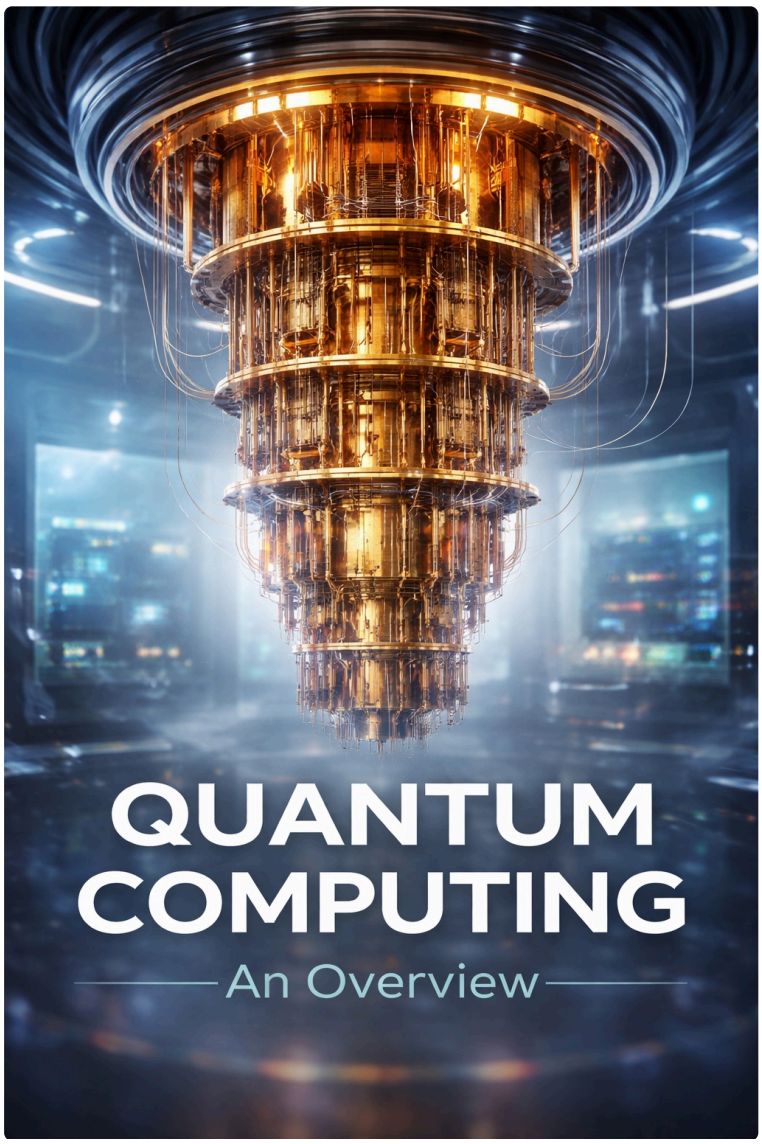
The author’s introduction

The author’s introduction, with documented scientific updates to the reading edition.

40 min read Dr. Charith Jayasekara

> **Reading edition · updated 6 November 2026.** Based on the author’s notes of 11 April 2026, retaining every section, table, seven original illustrations and bibliography. The RSA resource example in section 15 has been revised using 2025 and 2026 primary studies. Other hardware forecasts retain their source date. The seating example assumes labelled positions; unlabelled circular seating has a different count. See the guided foundation lessons for worked explanations and tested experiments.

Introduction to Quantum Computing



Dr. Charith Jayasekara

Last Update: 11th April 2026

By the end of going through this note, you should be able to:

- explain how quantum computing differs from classical computing in terms of state, logic, and measurement
- describe the roles of superposition, entanglement, and interference in quantum computation
- distinguish between NISQ systems and fault-tolerant quantum systems, and explain why that distinction matters
- explain how hybrid classical–quantum workflows operate, and where Quantum Software Engineering fits into the broader quantum computing ecosystem

Purpose

Quantum computing is emerging as one of the most important new computational paradigms of the post-Moore's-Law era. Its significance does not come from being a simple replacement for classical computing, but from introducing a fundamentally different way of representing, manipulating, and extracting information.

The central transition is from the familiar world of deterministic logic, where programs operate over definite states, to a world of probabilistic, state-based, and measurement-driven computation, where useful outcomes are engineered rather than directly enumerated.

This shift matters across computing. It affects how we think about algorithms, optimisation, simulation, hardware constraints, security, and the design of hybrid systems. In that sense, quantum computing is both a scientific and an engineering challenge. Where relevant, the term Quantum Software Engineering (QSE) is useful for describing the design, implementation, testing, and maintenance of computational workflows that involve quantum systems working alongside classical ones.

1. Why quantum computing matters now

For much of modern computing history, performance improvements came largely from advances in hardware. As transistors became smaller, denser, and more efficient, software benefited almost automatically. That long period of "free speed-up" is slowing because classical hardware scaling now faces real physical and engineering limits, including heat dissipation, fabrication complexity, and transistor miniaturisation boundaries (IBM, 2026).

This does not mean classical computing is obsolete. It means we can no longer assume that every hard computational problem will be solved simply by waiting for the next hardware generation. Quantum computing matters because it introduces a different computational model, one that may provide advantages for certain categories of problems such as combinatorial optimisation, structured search, simulation of molecules and materials, and some cryptographic tasks (Nielsen and Chuang, 2010). IBM's public roadmap also explicitly frames the current decade as one in which quantum systems are moving toward utility and, later, toward larger fault-tolerant systems (IBM, 2026).

In practical terms, this creates the need not only for quantum theory, but also for computational methods, programming models, circuit design tools, and hybrid workflows that allow quantum processors to be used meaningfully within broader computing systems. That is where the broader field of quantum computing intersects with Quantum Software Engineering.

2. Classical computation and quantum computation

Classical computing is built on the **bit**, where each bit takes one definite value: 0 or 1. At a programming level, this supports exact state transitions, deterministic branching, and the style of logic that appears in familiar constructs such as if, while, and function calls.



Figure: Representation of classical bit Vs Quantum bit

A useful analogy is a **light switch**. A switch is either off or on. A classical bit behaves the same way: it stores one definite alternative at a time.

Quantum computing is built on the **qubit**. A qubit can be in state 0, state 1, or a mathematically structured combination of both before measurement. Introductory quantum frameworks describe this through states, gates, and circuit transformations rather than ordinary variable assignment. A common visual model is the **Bloch sphere**, where a qubit can be represented as a point on the surface of a sphere rather than as a value confined to two corners (Google Quantum AI, n.d.; IBM Quantum, n.d.).

The most common analogy is a spinning coin. A coin lying on the table is clearly heads or tails, like a classical bit. A spinning coin is different: before it settles, it is not naturally described in the same way as a resting coin. This analogy is imperfect, but it helps convey the core idea that quantum systems are not merely holding hidden classical values; they are represented and manipulated differently.

The real conceptual shift is this:

- classical computing usually operates over **definite states**
- quantum computing operates over **probability amplitudes and measurement outcomes**

A concise way of expressing the difference is that classical computing programs certainty, while quantum computing engineers possibility.

3. The meaning of state

The idea of **state** is one of the best bridges from classical computing into quantum computing.

In classical programs, a variable such as x = 5 feels concrete. We can inspect it, print it, pass it to another function, and reason about it step by step. Classical computing assumes that, at least in principle, the machine's state can be

examined without changing it.

Quantum systems behave differently. A quantum state evolves under operations, but cannot be freely inspected without consequence. That means the state is not just data “sitting there” waiting to be read. It is part of an evolving computational process that must be transformed carefully until the correct moment for measurement.

A useful comparison is this:

- classical state is like a page in a notebook: we can read it without changing the writing
- quantum state is more like a soap bubble: touching it to inspect it can destroy the thing we wanted to study

That difference has enormous implications for how computations are designed and validated.

4. Measurement and why debugging is different

One of the most important principles in quantum computing is that **measurement changes the system**. In classical programming, observing an intermediate value does not normally disturb it. We can print variables, inspect memory, and step through debuggers freely.

In quantum computing, measurement generally collapses the quantum state into a definite classical outcome. This is why quantum languages and frameworks are built around circuit execution followed by measurement and analysis, rather than open-ended inspection of every intermediate state (OpenQASM, n.d.; Google Quantum AI, n.d.).

This has a direct computational consequence. In classical debugging, we might inspect values line by line:

print(x)

print(y)

print(z)

In quantum computing, we usually do something different:

- simulate the circuit
- reason mathematically about expected behaviour
- run the program many times
- compare the measured distribution with the expected one
- study how real hardware noise changes the outcome

This is one of the clearest places where **Quantum Software Engineering** becomes relevant: it is the practice of designing and validating computational workflows under these fundamentally different observability constraints.

A useful summary is:

We do not freely inspect quantum state(s) during execution.
We design, simulate, run, measure, and infer.

5. Superposition, entanglement, and interference

Three concepts are essential because they determine why quantum computation differs from classical computation in both power and difficulty.

5.1 Superposition

Superposition means a qubit can represent a structured combination of 0 and 1 before measurement. The popular phrase “0 and 1 at the same time” is acceptable at a very introductory level, but the more useful computing insight is that superposition gives us a richer state space over which algorithms can operate.

Superposition alone does not solve a problem. It only provides the raw representational capacity for a quantum algorithm to manipulate.

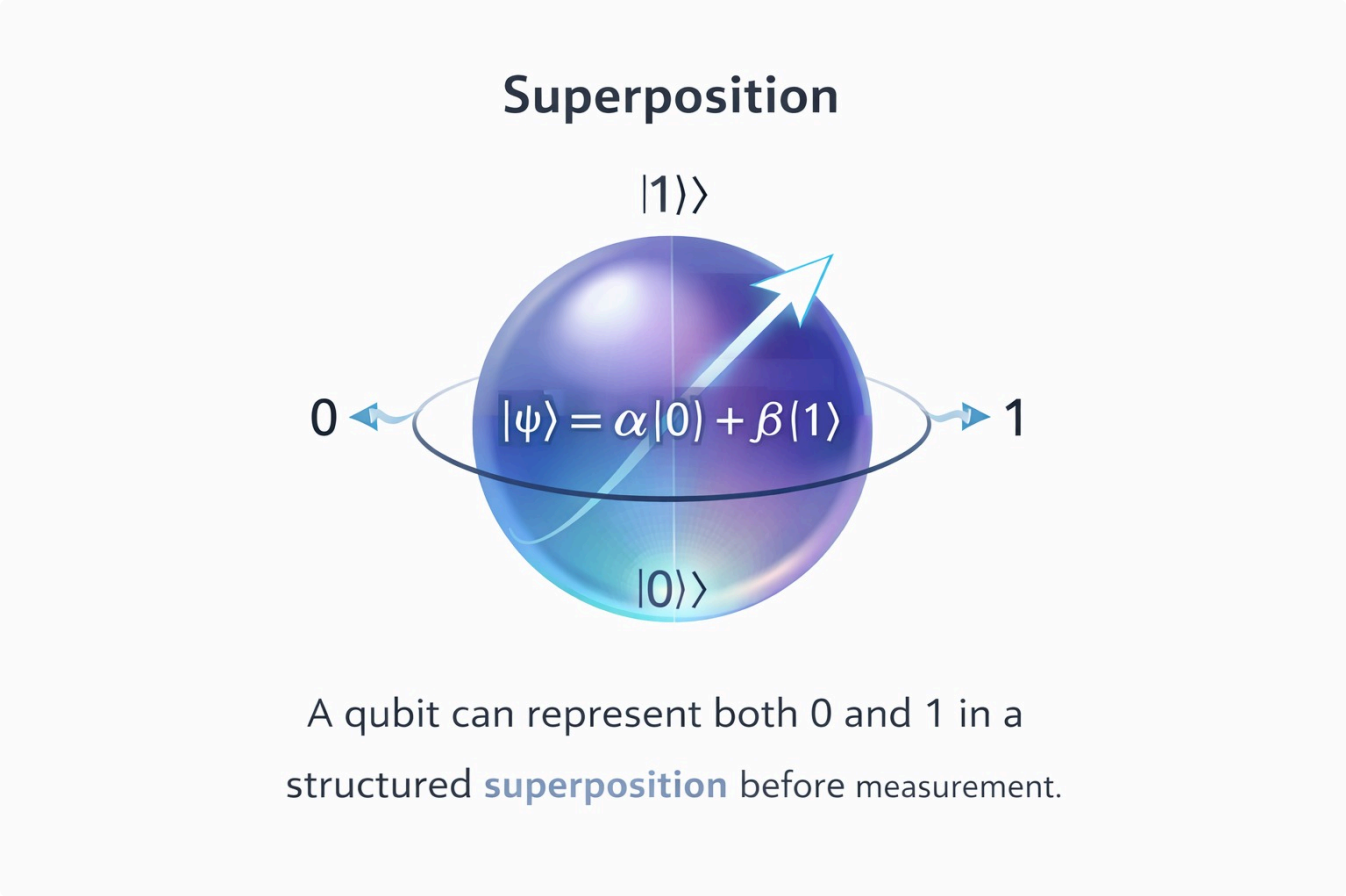


Figure : A qubit can represent both 0 and 1 in a structured superposition before measurement

5.2 Entanglement

Entanglement means that multiple qubits can become linked so that their states cannot be fully described independently.

A good analogy is not two separate coins tossed at the same time, but a pair of outcomes tied together (does not matter how far they are) by a hidden structural rule in the system. If one is measured, the other is constrained in a way that cannot be reproduced by simply treating them as separate classical variables.

Does this sound a bit weird?. But please note, many quantum algorithms depend on these non-classical correlations.

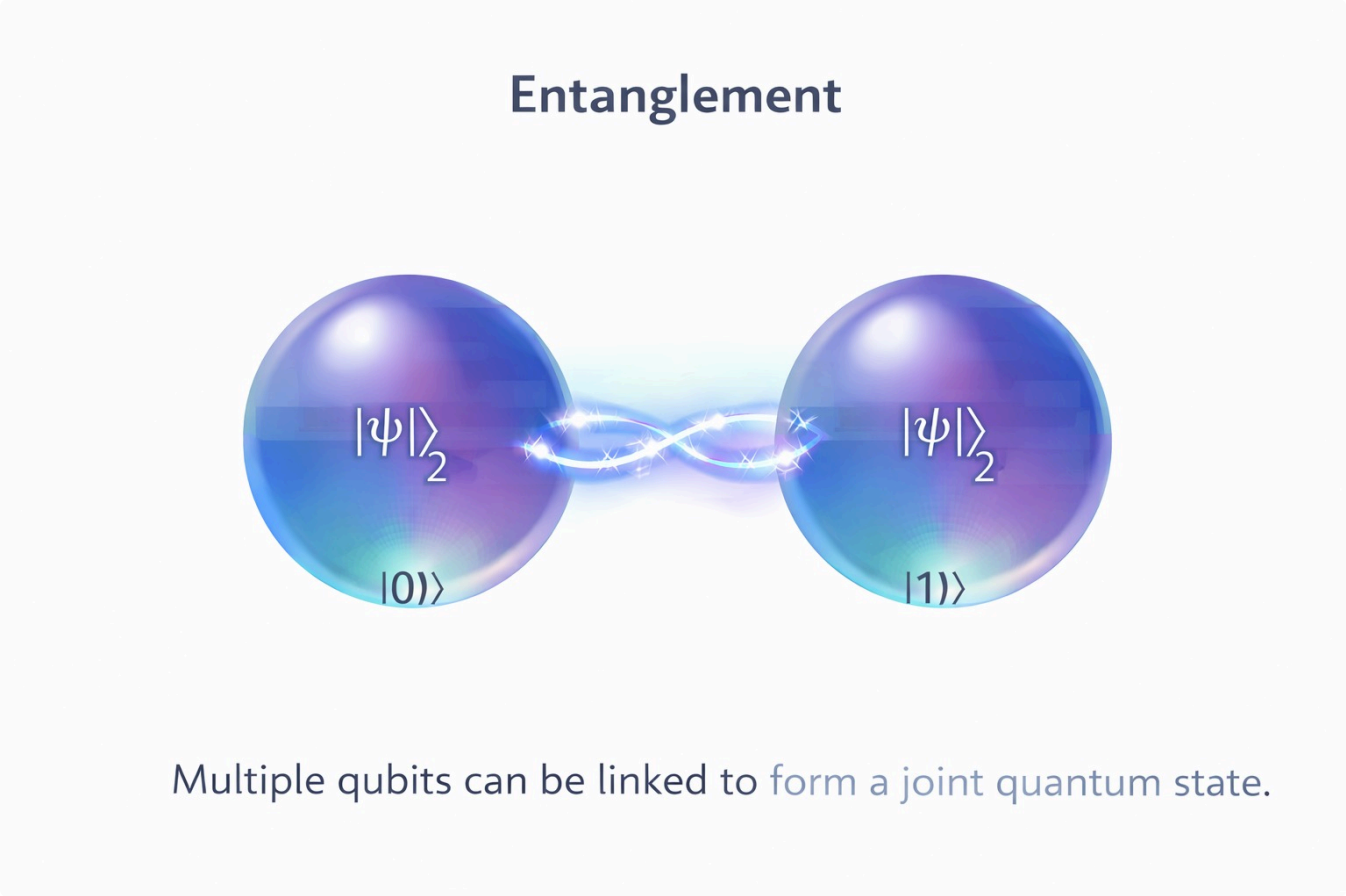


Figure : Multiple Qubits can be linked to form an entangled quantum state.

5.3 Interference

Interference is often the least intuitive but the most important for understanding quantum advantage.

A quantum computer does not simply “try all answers at once” and then magically reveal the best one. Rather, quantum algorithms are constructed so that computational paths corresponding to useful answers reinforce one another while computational paths corresponding to poor answers partially cancel out.

The water-wave analogy is helpful:

- when waves line up, they add
- when waves oppose, they diminish

Quantum algorithms exploit a mathematically similar phenomenon.

This leads to a very important computational insight:

Quantum algorithms do not just explore possibilities.
They shape the probability landscape so that desirable outcomes become more likely at measurement. (i.e Increasing the probability of selecting the best possible combination)

6. Why a quantum computer is not just a faster CPU

A quantum computer is **not** a general-purpose super-fast version of a laptop processor or cloud CPU.

Most everyday computing tasks, including web applications, operating systems, databases, file systems, user interfaces, and transaction processing, remain classical tasks and are expected to remain so. Quantum systems become interesting when the underlying problem has the right structure.

One good example is **combinatorial explosion**. Suppose a dinner event requires the best seating arrangement based on personality clashes, dietary constraints, family preferences, and social priorities. With only a few guests, a classical machine can test all possibilities or use ordinary optimisation. But as the number of guests grows, the number of arrangements grows extremely quickly. Similar growth appears in route planning, scheduling, portfolio construction, and logistics.

Combinatorial Explosion (Example: A seating arrangement)

Consider the problem of arranging people around a table. With 10 people, the total number of possible seating arrangements is 10! = 3,628,800. While this is already a large number, it is still within the range that classical computers can handle using efficient algorithms and optimisation techniques. However, if we increase the number of people to 15, the number of possible arrangements becomes 15! = 1,307,674,368,000, which is over 1.3 trillion combinations. This means that adding just 5 more people increases the number of possibilities by more than 360,000 times. This rapid growth is known as combinatorial explosion, where the number of possible solutions increases extremely quickly as the problem size grows. In such cases, even powerful classical computers may struggle, as evaluating every possibility can become extremely time-consuming or practically impossible. This is why problems of this nature, such as scheduling, route planning, and optimisation, are considered computationally challenging and are key areas where alternative approaches, including quantum methods, become highly relevant.

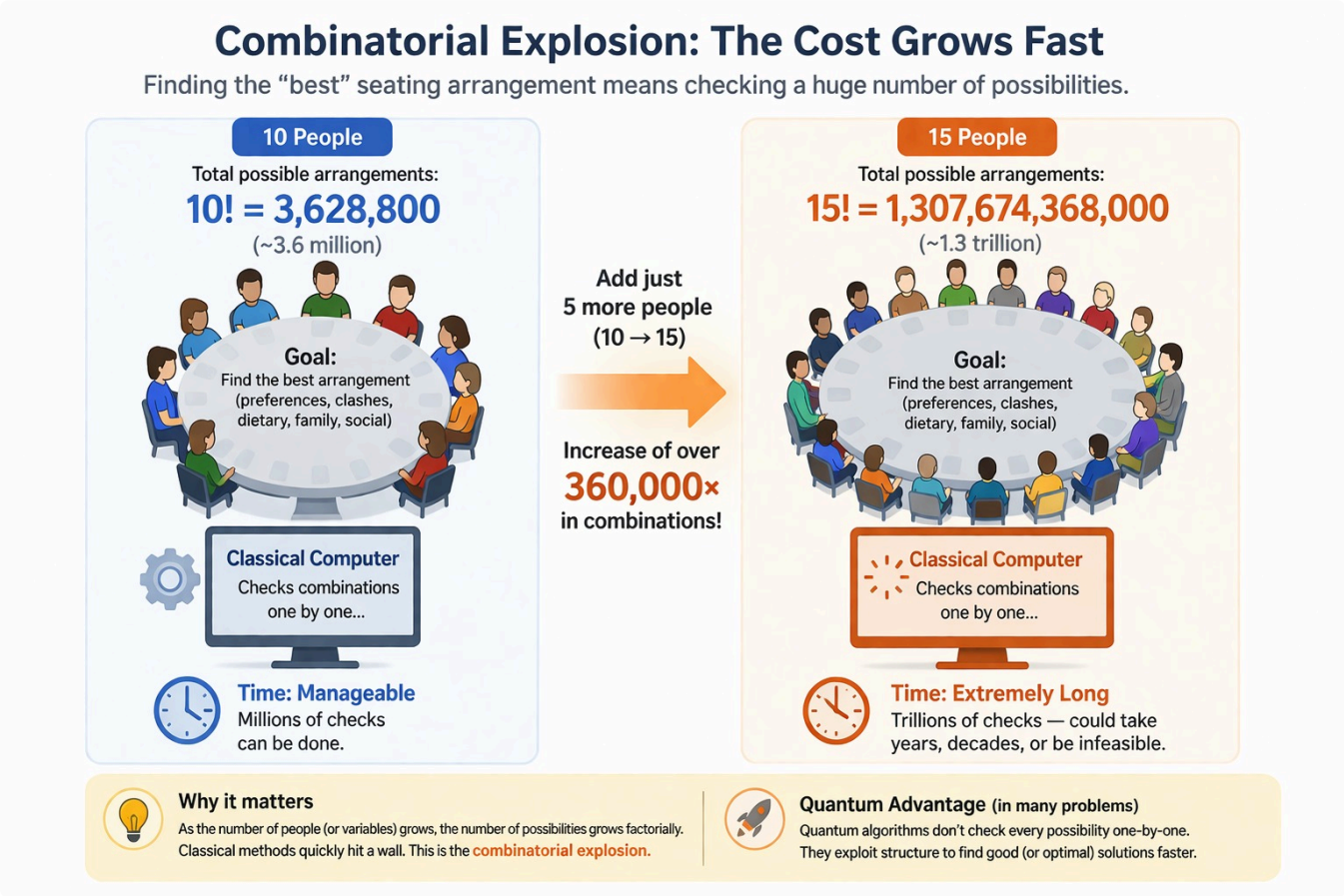


Figure : Combinatorial Explosion. The Computation cost grows fast.

Quantum algorithms can sometimes exploit this structure differently from classical methods. Three famous examples are:

- **Shor's algorithm** for factoring
- **Grover's algorithm** for search
- **Quantum Approximate Optimisation Algorithm (QAOA)** for optimisation

These algorithms are central to the educational and development ecosystems provided by modern quantum platforms (IBM Quantum, n.d.; Google Quantum AI, n.d.).

7. What work around quantum computing actually looks like

Although the broad field is quantum computing, much of the practical work happens through a sequence that feels familiar from computing and engineering.

7.1 Problem modelling

A real-world task such as route planning, scheduling, portfolio optimisation, chemistry simulation, cryptographic analysis, or error correction must first be expressed in a form that a quantum method can exploit.

The key question is not just:

Can this be coded?

It is:

Can this problem be represented in a way that a quantum algorithm can use effectively?

7.2 Algorithm selection

We then decide whether the task is better approached as:

- search
- optimisation
- simulation
- a variational method
- a cryptographic routine
- an error mitigation or correction strategy

7.3 Circuit design

The problem must then be translated into operations on qubits:

- initialise the state
- apply gates
- introduce entanglement if needed
- define when and how measurement will occur

At this point, the engineering feels very different from ordinary coding because we are designing transformations of state rather than just sequences of instructions.

7.4 Integration with classical code

A classical program typically handles:

- input collection
- parameter preparation
- circuit construction
- job submission
- result collection

- post-processing
- further decisions

This is why currently Python remains so prominent across the quantum ecosystem. IBM, AWS, Google, and Microsoft all support workflows in which classical orchestration remains central (AWS, n.d.; Microsoft, n.d.; Google Quantum AI, n.d.).

7.5 Testing and validation

Since outputs are probabilistic, correctness is often checked statistically. Instead of saying, “input A always produces output B,” we may say, “after 1,000 executions, the distribution should look approximately like this.”

This probabilistic style of validation is one of the places where **QSE** most clearly distinguishes itself from ordinary software development.

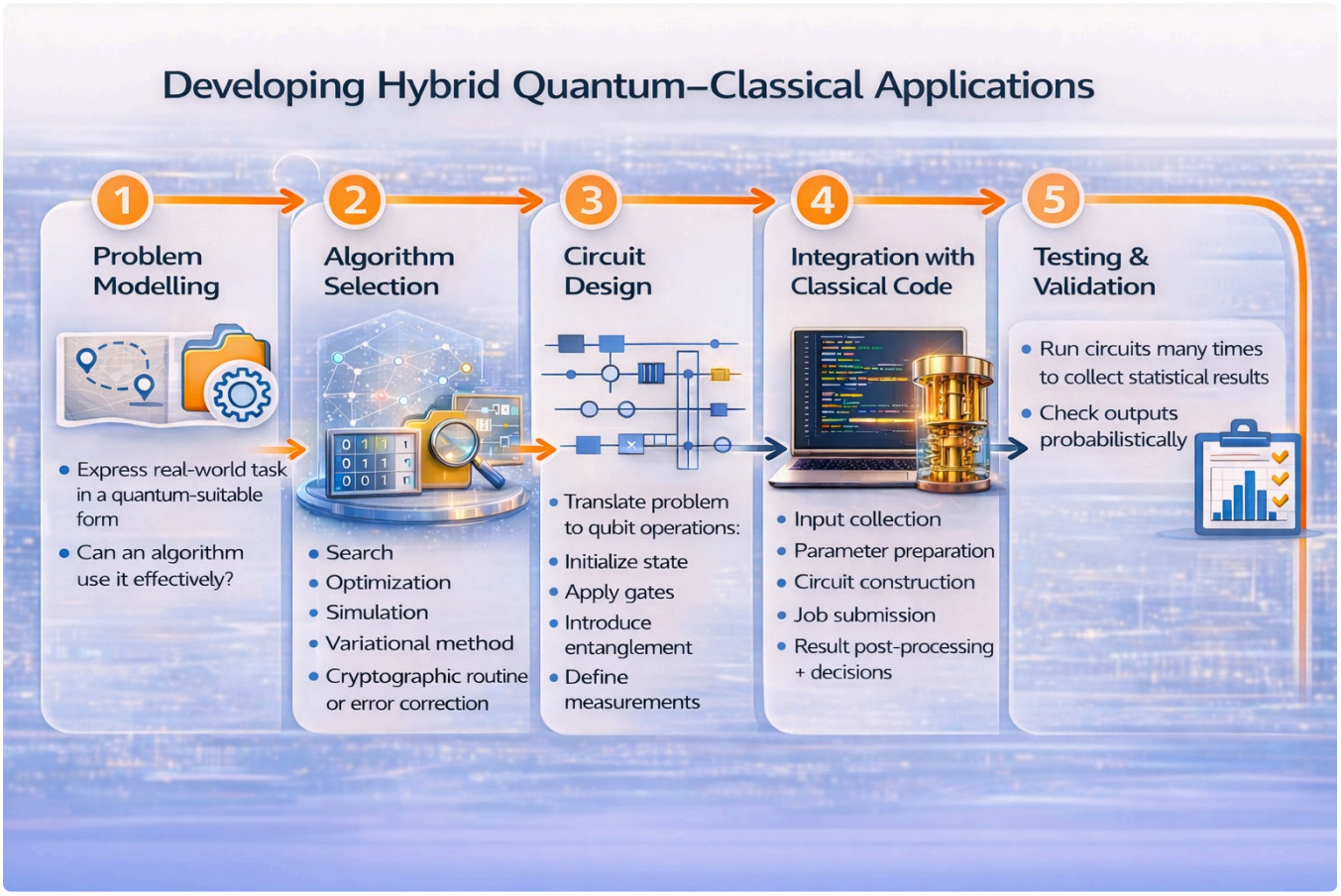


Figure : computing and engineering steps of developing a quantum application.

8. Hybrid classical–quantum architecture

One of the most important practical truths about current quantum computing is that it is not usually deployed as a “quantum-only application.” One of the modern standards is a **hybrid architecture**.

In a typical setup:

- a classical controller handles application logic, data preparation, networking, and orchestration
- a quantum processing unit receives a specific computational kernel, usually encoded as a circuit
- the QPU executes that circuit and returns measurement results
- the classical side analyses those results and may decide to modify parameters and run again

This is especially common in variational methods, where a classical optimiser and a quantum circuit work in a loop (AWS, n.d.; IBM Quantum, n.d.).

A useful analogy is:

- the classical computer acts as the **project manager**
- the quantum processor acts as a **specialist consultant**

The manager prepares the problem, asks a focused question, receives a result, and decides the next step. This is much more accurate than imagining the QPU replacing the entire application stack.

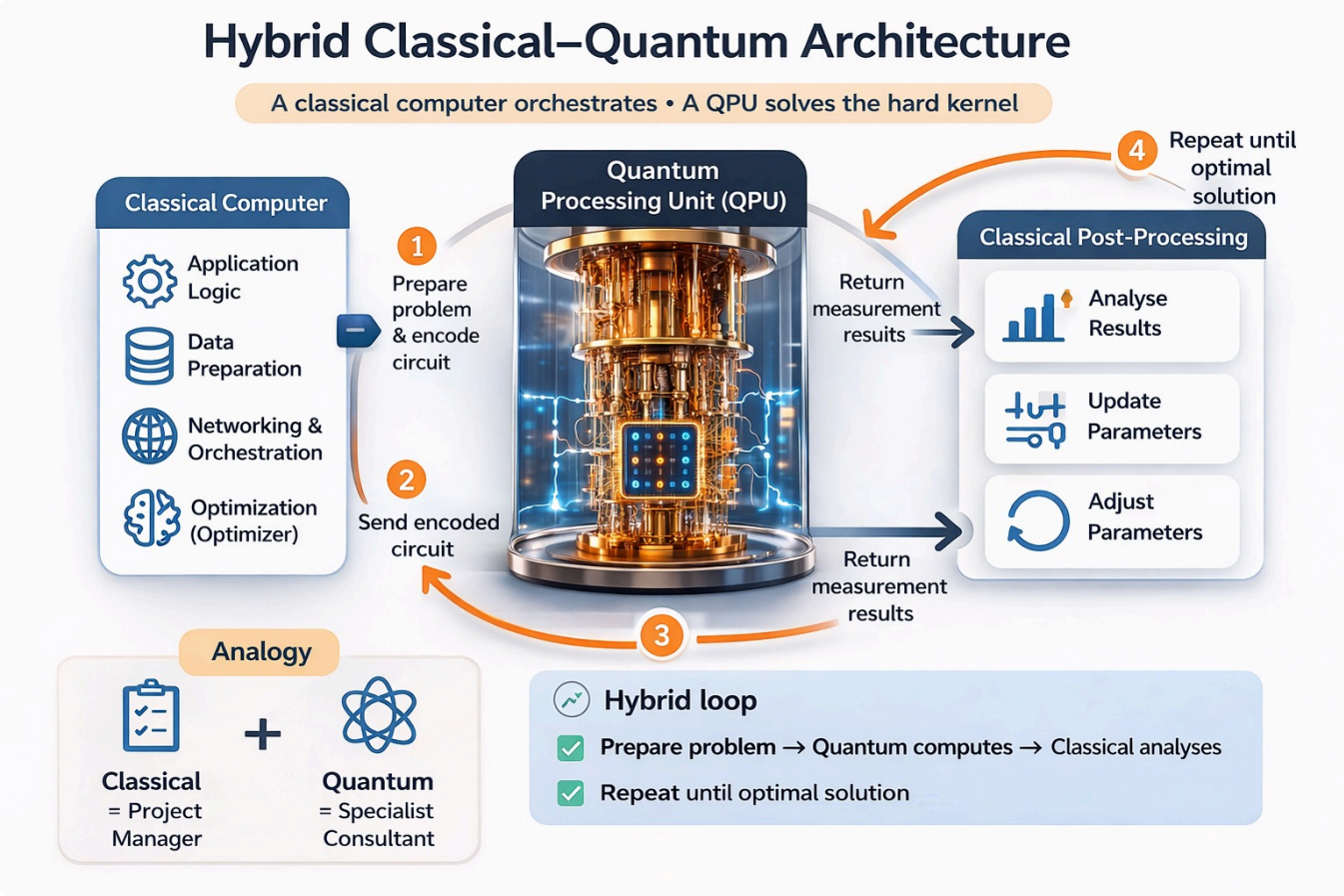


Figure : A possible hybrid classical - quantum architecture.

9. The quantum computing stack

Quantum computing is easier to understand if we separate it into layers.

Application layer

This is where real-world domains sit:

- Finance
- cybersecurity
- logistics
- chemistry
- materials science
- machine learning

- optimisation

Algorithm and circuit layer

This is where quantum logic is expressed:

- gate design
- circuit structure
- ansatz construction
- oracle design
- measurement strategy

Qiskit and Cirq are major tools here. IBM positions Qiskit as its core development ecosystem, while Google presents Cirq as a Python framework for creating and simulating circuits (IBM Quantum, n.d.; Google Quantum AI, n.d.).

Compiler and representation layer

This layer translates abstract circuit logic into executable forms. OpenQASM is an important example, providing a representation that matches the circuit model, measurement, and classical feed-forward behaviour needed in near-term workflows (OpenQASM, n.d.).

Hardware layer

This is where the physical devices themselves sit:

- superconducting circuits
- trapped ions
- photonic systems
- neutral atoms
- cryogenic environments
- lasers
- control electronics
- vacuum systems

Not everyone working in quantum computing operates at this layer, but everyone above it is constrained by it.

10. Example: classical versus quantum random number generation

A simple example helps connect the ideas to practice.

In a classical environment, random number generation is usually done algorithmically:

import random

print(random.randint(0, 1))

In a quantum setting, a minimal circuit can:

- prepare a qubit in state 0
- apply a Hadamard gate

- measure it

If repeated many times, the results are approximately balanced between 0 and 1.

The important point is not just that quantum randomness exists. It is that the workflow differs:

- define the task
- construct the circuit
- run it many times
- collect measurement counts
- interpret the distribution

This is a small but genuine example of quantum computing in action, and from a QSE perspective it is also a small but complete engineering cycle.

11. Example: optimisation with QAOA

Imagine a delivery company trying to minimise fuel cost while satisfying deadlines, vehicle capacities, traffic constraints, and warehouse sequencing. A classical system may use heuristics, approximation methods, or branch-and-bound strategies.

A quantum approach may instead map the problem into a structured optimisation form and use a hybrid algorithm such as **QAOA**.

In such a workflow:

- the classical side encodes the optimisation problem
- the quantum circuit evaluates candidate structures
- the classical optimiser updates circuit parameters
- the loop repeats until convergence or another stopping rule is reached

This example is useful because it shows that current quantum computing is not about replacing the whole application. It is about integrating a specialised quantum routine into a wider computational workflow.

12. The current state of quantum hardware

The hardware context is essential because it explains both the promise and the limitations of current quantum computing.

As of 2026, the field is best described as **cloud-accessible, utility-oriented, and still hardware-limited**. IBM's 2026 roadmap highlights goals such as running 7,500 gates on up to 360 qubits using the Nighthawk architecture, along with work on Kookaburra as a fault-tolerant module containing a logical processing unit and quantum memory (IBM, 2026). IBM also publicly states that it is targeting near-term quantum advantage by the end of 2026 and a large-scale fault-tolerant quantum computer by 2029.

This is important progress, but it is not the same as saying that universal, large-scale, fault-tolerant quantum computing is already available for ordinary commercial workloads.

A good historical analogy is that today's quantum hardware is closer to the **vacuum-tube era** of classical computing than to the smartphone era. It is powerful, exciting, remote, and still difficult to scale cleanly.

Cloud access is central to this stage of the field. Amazon Braket presents itself as a managed service for accessing quantum computers and simulators through AWS, while IBM provides cloud platform access to quantum hardware and

software tooling. This means many people will learn and experiment with quantum systems through remote platforms rather than by physically owning hardware (AWS, n.d.; IBM Quantum, n.d.).

13. The hardware "zoo"

Unlike classical computing, where CMOS-based approaches became dominant, quantum computing still has several competing hardware paradigms.

Table 1 : Quantum hardware technologies

Technology	Description	Pros	Cons
Superconducting Systems	Used by IBM and Google, these rely on superconducting electrical circuits operating at extremely low temperatures using cryogenic infrastructure.	Fast gate operations, strong industrial momentum	Fragility, heavy cooling requirements, sensitivity to noise
Trapped-Ion Systems	Used by IonQ, these rely on charged atoms controlled by electromagnetic fields and lasers.	Long coherence, high fidelity characteristics	Gate operations can be slower than superconducting systems
Photonic Systems	These use particles of light (photons) instead of matter-based qubits.	Room-temperature potential, strong networking advantages	Control and scaling challenges
Neutral-Atom Systems	These use lasers to arrange and manipulate atoms, and are available through cloud platforms such as AWS Braket.	Promising scalability, flexible geometric layouts	Still part of an evolving frontier

Key Insight

The key lesson is simple: there is no single settled "standard quantum chip" yet. Unlike classical computing, where a dominant hardware architecture emerged over time, quantum computing is still exploring multiple competing approaches. As a result, tools, workflows, and computational abstractions must be designed to be flexible, hardware-aware, and portable across different quantum technologies. This diversity is both a challenge and an opportunity, as it allows innovation across multiple directions while also requiring careful engineering to ensure that software can adapt to different underlying physical systems.

14. Noise and decoherence: why today's systems are called NISQ

A central phrase in modern quantum computing is **Noisy Intermediate-Scale Quantum (NISQ)**, meaning (Preskill, 2018).

The term captures the current stage of the field:

- machines exist and can be useful for research and experimentation
- they are still noisy
- they are not yet broadly fault tolerant

Decoherence refers to the loss of quantum information due to unwanted interaction with the environment. Heat, vibration, electromagnetic interference, imperfect controls, or simple device instability can corrupt a quantum state before the intended computation finishes.

A useful analogy is this:

- a classical bit is like writing on paper with ink
- a noisy qubit is like trying to write on the surface of water

This is not just a physics problem. It is a computing and engineering constraint. Noise changes how circuits must be designed, how long they can be, how outputs must be interpreted, and how validation is performed. In that sense, software does not sit “above” noise as if it were someone else’s problem. Noise is part of the computational environment itself.

Google’s Cirq documentation also explicitly frames the library as useful for noisy intermediate-scale quantum devices, where hardware details matter strongly to practical performance.

15. Physical qubits and logical qubits

One of the most important distinctions in current quantum computing is the difference between **physical qubits** and **logical qubits**.

A **physical qubit** is the actual hardware unit implemented on a device. It is real, but noisy.

A **logical qubit** is a more reliable qubit built by encoding information across many physical qubits so that errors can be detected and corrected.

This distinction matters because a machine may have many physical qubits but far fewer usable logical qubits. That is one reason why large physical-qubit headlines should not be interpreted as immediate signs of universal fault-tolerant computing or immediate cryptographic collapse. IBM’s roadmap explicitly treats encoded information, logical processing, and modular fault-tolerant systems as major future milestones rather than already-solved problems (IBM, 2026).

The broad computational lesson is that raw scale is not enough. Reliability, error correction, compilation, and hardware-aware design matter just as much.

Example: Physical vs Logical Qubits (Why Scale Alone Is Not Enough)

Physical-to-logical overhead is not a fixed multiplier. For RSA-2048, Gidney’s 2025 study estimates fewer than one million noisy physical qubits and less than a week of runtime, assuming a nearest-neighbour square grid, 0.1-percent gate error, one-microsecond surface-code cycles and ten-microsecond control reaction time. Read the 2025 resource study. Webster and colleagues’ Pinnacle study, revised May 2026, estimates fewer than one hundred thousand physical qubits using quantum LDPC codes under its stated error-rate and timing assumptions. Read the Pinnacle architecture study.

These are conditional resource estimates, not completed RSA-2048 factoring demonstrations or predictions of a delivery date. Compare architecture, connectivity, runtime, decoding and control requirements, physical error assumptions and target success probability before comparing headline counts. A smaller physical-qubit estimate can reflect a different code or space–time trade-off. The practical lesson is that raw qubit count alone does not establish attack capability: a sufficiently reliable full computation is required. See F22: noise and hardware limitations for a guided explanation.

16. Connectivity and crosstalk

Scaling a quantum computer is not simply a matter of adding more qubits.

Real quantum devices have **topology constraints**:

- not all qubits can interact directly

- some interactions are easier than others
- routing may require extra operations
- neighbouring qubits can interfere unintentionally

This last issue is often called **crosstalk**. It means that qubits can affect each other in unwanted ways, increasing error rates and complicating execution.

For computation, this matters because the same abstract circuit may have to be mapped differently depending on the hardware's connectivity graph. That often requires qubit remapping, additional swap operations, or changes to circuit depth. In practice, even high-level computational logic is constrained by low-level hardware geometry.

17. Why software and systems work matter even more in the noisy era

It can be tempting to think that meaningful progress must wait until the hardware is "good enough." In reality, the opposite is true.

Because current hardware is noisy, incomplete, and evolving, work on programming models, compilers, circuit optimisation, benchmarking, simulation, hybrid orchestration, testing, and error mitigation is especially important right now.

This is exactly where **Quantum Software Engineering** becomes central. QSE is not the whole field of quantum computing, but it is a crucial part of how the field becomes usable. It includes work on programming frameworks, hardware abstraction, compiler design, validation tools, circuit rewriting, runtime orchestration, and integration into broader computing systems.

In other words, the messiness of present hardware increases the value of good engineering. It does not reduce it.

18. Security, cryptography, and the post-quantum transition

One of the most socially significant impacts of quantum computing is in cryptography.

Shor's algorithm is famous because a sufficiently powerful fault-tolerant quantum computer could use it to threaten widely deployed public-key systems based on factoring and related assumptions. That is why governments, standards bodies, and industry have been pushing toward **post-quantum cryptography (PQC)**.

The broader computing lesson is powerful:

The right response to disruptive technology is not just fear, but redesign.

Quantum computing creates risk for current cryptographic systems, but it also creates an opportunity and responsibility to build the next generation of secure infrastructure.

19. Platforms and tools

Several platforms are especially important for learning and development.

- **Qiskit and IBM Quantum** provide an ecosystem for constructing circuits, using simulators, and running jobs on cloud-based quantum devices. (ibm.com)
- **Cirq** is Google Quantum AI's Python framework for constructing and simulating quantum circuits. Google describes it as a library for writing, manipulating, and optimising circuits, and its tutorials move from simple examples to more advanced workflows. (Google Quantum AI)

- **Q#** is Microsoft's high-level open-source quantum programming language within the Azure Quantum ecosystem. (Azure Quantum)
- **Amazon Braket** provides managed access to simulators and multiple hardware families through AWS. (AWS Documentation)
- **OpenQASM** remains important as a machine-oriented representation for expressing quantum circuits and forms of classical control within circuit-based execution models. (OpenQASM)

20. How to prepare

The best preparation path remains grounded in core computing.

- **Programming** matters, and Python is a particularly useful entry point because it sits at the centre of many quantum workflows across IBM, AWS, Google, and Microsoft ecosystems.
- **Mathematics** matters, especially linear algebra and probability, because quantum states, gates, and measurements are naturally expressed through vectors, matrices, and distributions.
- **Systems thinking** matters because quantum routines are rarely isolated. They must be embedded in larger workflows involving data handling, cloud execution, measurement analysis, and maintainability.

Mindset matters because the most important intellectual shift is this:

- classical computing encourages us to think in exact branches over exact states
- quantum computing requires us to think in transformations, interference, measurement, and guided probability

A concise way to express the shift is:

In the classical era, we programmed certainty.
In the quantum era, we engineer possibility.

21. Final synthesis

Quantum computing is not merely "classical computing, but faster." It is a different model of information processing built around state preparation, transformation, entanglement, interference, and measurement. Its value lies not in replacing all of classical computing, but in complementing it where the problem structure allows quantum methods to provide leverage.

At the same time, the field is still constrained by noisy hardware, imperfect scaling, connectivity limits, and the gap between physical and logical qubits. This is why hybrid systems, cloud access, careful validation, and engineering discipline matter so much.

A strong final definition is therefore:

Quantum computing is the study and application of computational systems that use quantum-mechanical principles to represent and process information, currently, often in hybrid partnership with classical systems.

And where implementation, orchestration, optimisation, and validation of those workflows become the focus, **Quantum Software Engineering** is the engineering discipline that helps make quantum computing usable in practice.

Glossary of Terms

Algorithm
A step-by-step computational method for solving a problem. In quantum computing, algorithms are designed to

manipulate quantum states so that useful outcomes become more likely when measured.

Amplitude
A mathematical value associated with a quantum state. Amplitudes determine the probability of measuring a particular outcome, but are not themselves probabilities. Their signs and phases allow interference to occur.

Ansatz
A structured trial circuit used in many hybrid quantum algorithms, especially variational methods such as QAOA and VQE. It is essentially a parameterised circuit whose settings are adjusted to find a good solution to a problem.

Bit
The basic unit of classical information. A bit can take only one of two definite values: 0 or 1.

Bloch Sphere
A visual representation of the state of a single qubit. Instead of thinking of a qubit as only 0 or 1, the Bloch sphere helps show that a qubit can exist in many possible states before measurement.

Circuit Depth
The number of layers of operations in a quantum circuit. Deeper circuits usually take longer to run and are more vulnerable to noise and decoherence.

Classical–Quantum Hybrid System
A computing system in which a classical computer and a quantum processor work together. The classical side typically handles orchestration, data preparation, and post-processing, while the quantum side performs a specialised computational task.

Compiler
A tool that translates code or abstract computational instructions into a lower-level form that hardware can execute. In quantum computing, compilers often map circuits onto real hardware constraints such as connectivity and gate availability.

Connectivity
The pattern describing which qubits on a quantum device can directly interact with each other. Limited connectivity can require additional operations to implement a circuit.

Crosstalk
Unwanted interaction between qubits or control lines on a quantum device. Crosstalk can introduce errors and reduce the reliability of computations.

Decoherence
The loss of quantum information due to unwanted interaction with the environment. Decoherence is one of the main reasons current quantum hardware is noisy and difficult to scale reliably.

Entanglement
A quantum phenomenon in which the states of two or more qubits become linked so that they must be described together. Measuring one qubit can constrain the outcome of another.

Fault-Tolerant Quantum Computing
A stage of quantum computing in which errors can be detected and corrected reliably enough for long and complex computations to be performed. This is beyond the current NISQ era and is a major long-term goal of the field.

Gate
A basic operation applied to one or more qubits, similar in spirit to a logic operation in classical computing. Quantum gates change the state of qubits in controlled ways.

Hadamard Gate
A very important single-qubit gate that places a qubit into an equal superposition of 0 and 1 when applied to a qubit initially in state 0. It is often one of the first gates introduced in beginner quantum circuits.

Hybrid Algorithm
An algorithm that combines classical and quantum computation in a loop. The classical system typically updates parameters or processes results, while the quantum system evaluates part of the problem.

Interference
The process by which quantum amplitudes combine so that some outcomes are reinforced and others are cancelled. Interference is a key reason quantum algorithms can outperform classical ones for certain tasks.

Logical Qubit
A more reliable qubit built by encoding information across multiple physical qubits so that errors can be detected and corrected.

Measurement
The process of reading out a quantum state and obtaining a classical result. Measurement usually collapses the quantum state into a definite outcome.

NISQ
Short for Noisy Intermediate-Scale Quantum. This describes the current era of quantum computing, where devices have useful numbers of qubits but still suffer from significant noise and limited error correction.

OpenQASM
A language and representation format used to describe quantum circuits and some related classical control operations in a hardware-oriented way.

Oracle
A problem-specific subroutine used in some quantum algorithms, especially search algorithms. It is often used to "mark" candidate solutions without explicitly revealing them.

Physical Qubit
An actual qubit on a hardware device. Physical qubits are real but noisy, and many may be required to build a smaller number of reliable logical qubits.

Probability Distribution
A description of how likely different outcomes are. In quantum computing, repeated measurements of a circuit often produce a distribution of results rather than a single fixed answer.

QAOA
Short for Quantum Approximate Optimisation Algorithm. A hybrid quantum-classical algorithm designed for solving optimisation problems by repeatedly tuning a parameterised quantum circuit.

QPU
Short for Quantum Processing Unit. This is the part of a system that performs the quantum computation, similar in role to how a CPU performs classical computation.

Qubit
The basic unit of quantum information. A qubit can exist in state 0, state 1, or a superposition of both until it is measured.

Quantum Software Engineering (QSE)
The discipline concerned with designing, implementing, testing, and maintaining software and workflows that involve quantum computation, usually in conjunction with classical systems.

Simulation
Running a model of a quantum circuit on a classical computer instead of on real quantum hardware. Simulators are very useful for learning, testing, and debugging.

State
The mathematical description of a system at a particular point in time. In classical computing, state can usually be read

without changing it; in quantum computing, the act of measurement can alter it.

Superposition
A property of quantum systems in which a qubit can be in a structured combination of 0 and 1 before measurement.

Swap Operation
An operation used to exchange the states of two qubits. Swap operations are often inserted when hardware connectivity does not allow a desired direct interaction.

Variational Algorithm
A hybrid quantum-classical algorithm in which a parameterised quantum circuit is repeatedly adjusted by a classical optimiser to improve performance on a target task.

VQE
Short for Variational Quantum Eigensolver. A hybrid algorithm commonly used in quantum chemistry and materials science to estimate low-energy states of quantum systems.

References

Amazon Web Services (AWS) (n.d.) What is Amazon Braket? Available at: https://docs.aws.amazon.com/braket/latest/developerguide/what-is-braket.html

Google Quantum AI (n.d.) Cirq basics and documentation. Available at: https://quantumai.google/cirq/start/basics

IBM (2026) IBM Quantum Roadmap 2026. Available at: https://www.ibm.com/roadmaps/quantum/2026

IBM Quantum (n.d.) Quantum hardware and roadmap overview. Available at: https://www.ibm.com/quantum/hardware

IonQ (n.d.) Our trapped ion technology. Available at: https://www.ionq.com/technology

Microsoft (n.d.) Introduction to the quantum programming language Q#. Available at: https://learn.microsoft.com/en-us/azure/quantum/qsharp-overview

Nielsen, M.A. and Chuang, I.L. (2010) Quantum Computation and Quantum Information. Cambridge: Cambridge University Press.

National Institute of Standards and Technology (NIST) (n.d.) Post-quantum cryptography. Available at: https://www.nist.gov/pqc

OpenQASM (n.d.) Introduction — OpenQASM live specification. Available at: https://openqasm.com/intro.html

Preskill, J. (2018) 'Quantum computing in the NISQ era and beyond', Quantum, 2, p. 79. Available at: https://doi.org/10.22331/q-2018-08-06-79

Yanofsky, N.S. and Mannucci, M.A. (2008) Quantum Computing for Computer Scientists. Cambridge: Cambridge University Press.

Books for further learning

Johnston, E.R., Harrigan, N. and Gimeno-Segovia, M. (2019) Programming Quantum Computers: Essential Algorithms and Code Samples. Sebastopol, CA: O'Reilly Media.

Nielsen, M.A. and Chuang, I.L. (2010) Quantum Computation and Quantum Information. Cambridge: Cambridge University Press.

Sutor, R.S. (2019) Dancing with Qubits: How Quantum Computing Works and How It Can Change the World. Birmingham: Packt.

Yanofsky, N.S. and Mannucci, M.A. (2008) Quantum Computing for Computer Scientists. Cambridge: Cambridge University Press.

Useful resources

- IBM Quantum / Qiskit documentation. (ibm.com)
- Google Quantum AI / Cirq documentation. (Google Quantum AI)
- Microsoft Azure Quantum / Q# documentation. (Azure Quantum)
- Amazon Braket documentation. (AWS Documentation)
- OpenQASM specification. (OpenQASM)
- Monash FIT Post Quantum Cryptography Project. (PQC-MonashFIT)