UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
School of Professional & Executive Development

Quside

Quantum Software Development

Dr. José Ramón Martínez Saavedra

Barcelona, Thursday, March 26th, 2026

About us

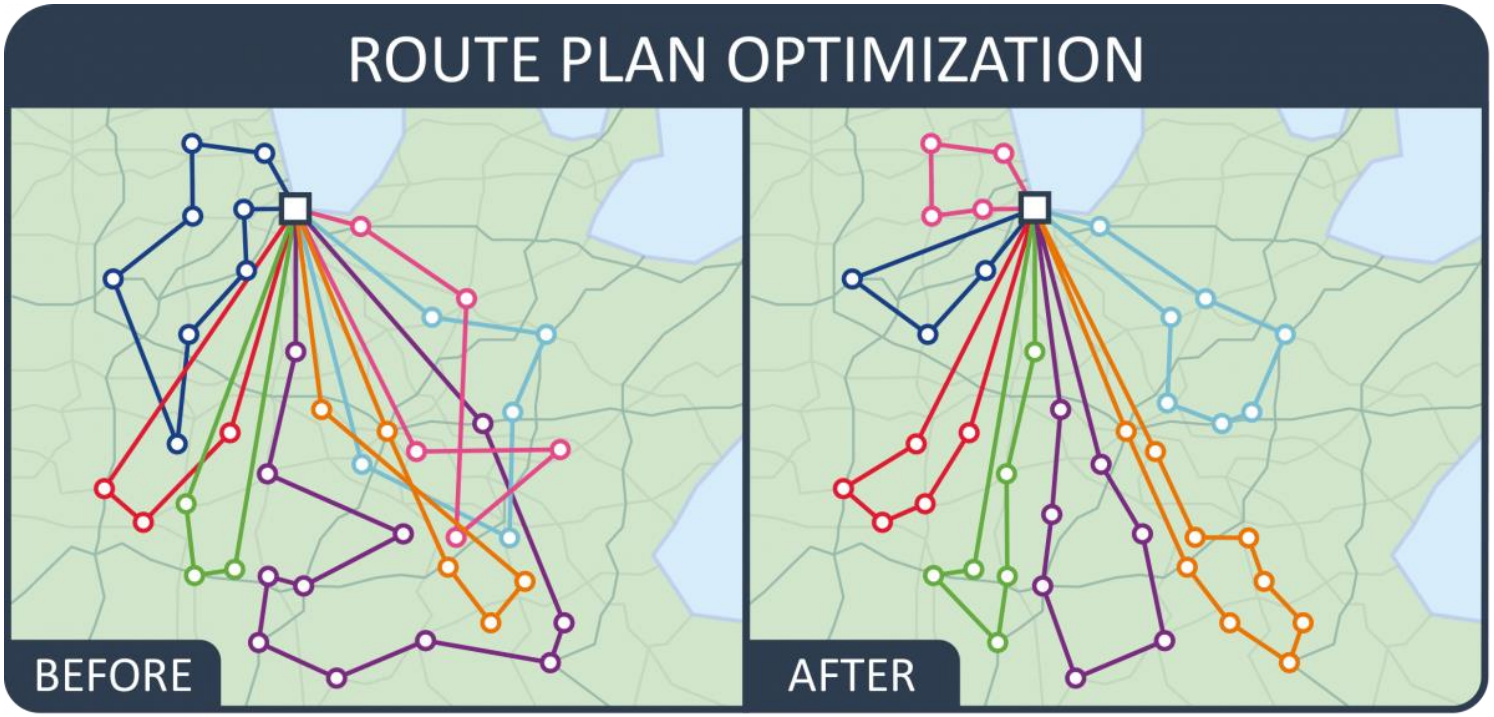
We build quantum technologies for safer connectivity and more powerful computation.

Based in the Barcelona area in the Mediterranean Technology Park.

Deep tech company spin-off from ICFO

Track record in quantum and photonics with pioneering scientific results and patents.

A typical day for a QC software developer



The QC Landscape in 2026 — What Changed

- **The quantum computing landscape has shifted dramatically**
  - • 2020: NISQ era. Fragmented SDKs (Qiskit Terra/Aer/Ignis/Aqua). <100 noisy qubits.
  - • 2023: Utility-scale experiments. IBM Eagle 127 qubits. SDK consolidation begins.
  - • 2024: Google Willow — first below-threshold quantum error correction (Physics World Breakthrough of the Year).
  - • 2025: Microsoft Majorana 1 — topological qubit claim. IBM Qiskit 1.0 ships. NVIDIA CUDA-Q emerges.
  - • 2026: Multi-provider marketplaces (Azure, Braket). QEC papers surge 3.3x. Error correction is now an engineering problem.
- Key message: QEC is no longer theoretical — it fundamentally changes what SW developers must plan for.

Why QC Software Is Different NOW vs. 5 Years Ago

- **Four fundamental shifts in quantum software development**

- **SDK Architecture**
  - • 2020: Monolithic frameworks (Qiskit Terra + Aer + Ignis + Aqua)
  - • 2026: Modular, primitives-based APIs (Sampler, Estimator)

- **Hardware Access**
  - • 2020: Single-vendor lock-in (IBM-only or Google-only)
  - • 2026: Multi-provider marketplaces (Azure: IonQ + Quantinuum + PASQAL + Rigetti)

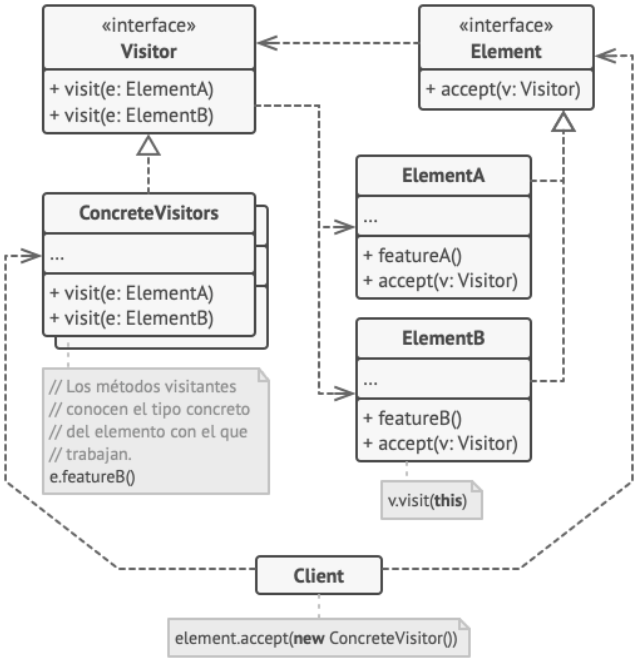
- **Testing & Verification**
  - • 2020: Basic statevector simulation
  - • 2026: QEC-aware resource estimation, classical shadows, error mitigation (Mitiq)

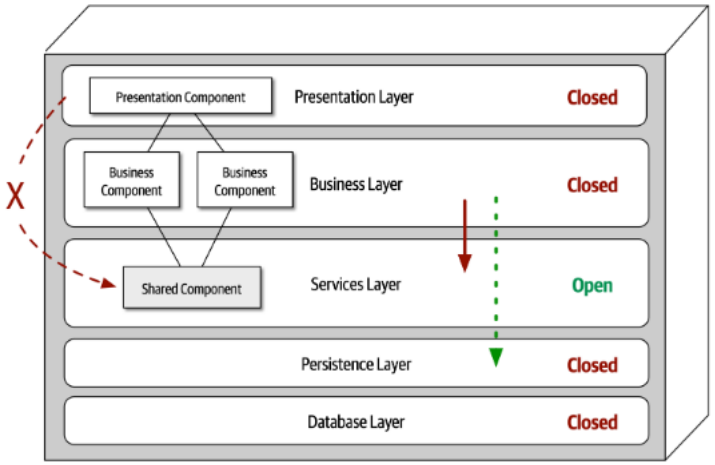
- **Simulation Infrastructure**
  - • 2020: CPU-only simulation
  - • 2026: GPU-accelerated simulation (NVIDIA CUDA-Q, cuQuantum) — 450x speedups

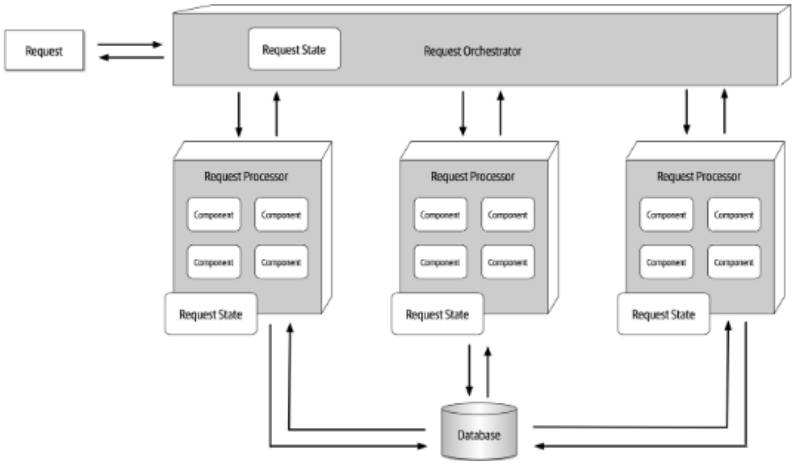
Abstraction layers, architectural constraints, and the QEC turning point

QC AS A HARDWARE ACCELERATOR

Abstraction is a fundamental capability of most classical SW designs





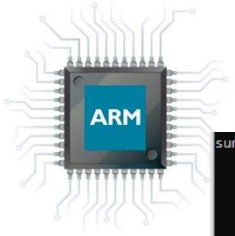


Most of the time, the programmer doesn't need to know details of the computer itself

```
.LFB246:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movsd   %xmm0, -24(%rbp)
        movsd   %xmm1, -32(%rbp)
        movsd   -24(%rbp), %xmm0
        addsd   -32(%rbp), %xmm0
        movsd   %xmm0, -8(%rbp)
        movsd   -8(%rbp), %xmm0
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

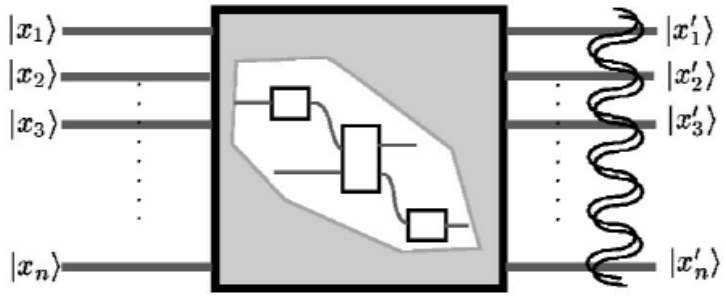
```
.LFB246:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movsd   %xmm0, -24(%rbp)
        movsd   %xmm1, -32(%rbp)
        movlpd  -24(%rbp), %xmm0
        addsd   -32(%rbp), %xmm0
        movsd   %xmm0, -8(%rbp)
        movlpd  -8(%rbp), %xmm0
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

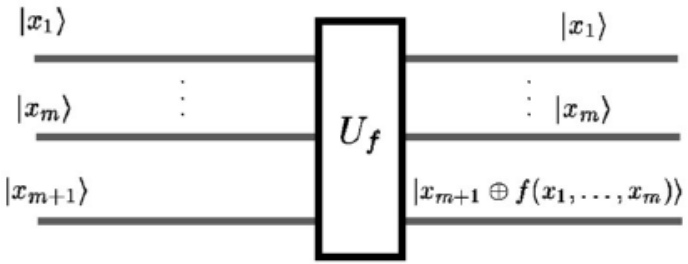
```
double sum(const double a, const double b)
{
        double c = a + b;
        return c;
}
```

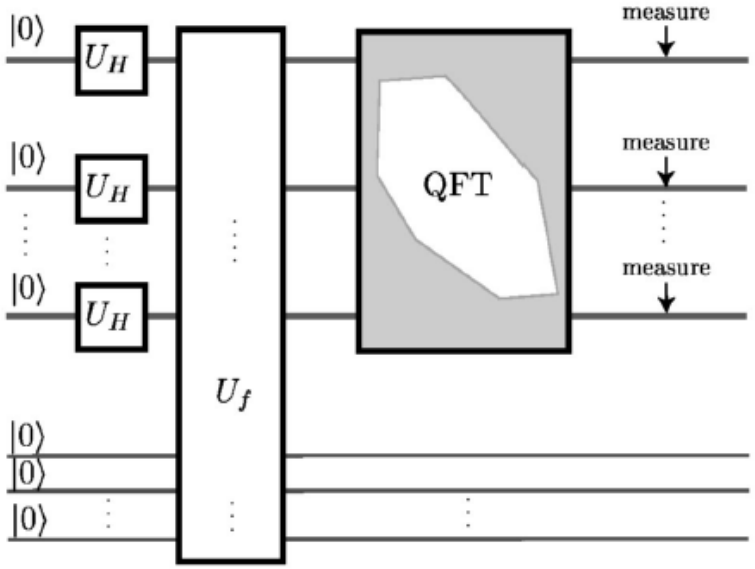



```
sum:
        @ args = 0, pretend = 0, frame = 24
        @ frame_needed = 1, uses_anonymous_args = 0
        push    {fp, lr}
        add     fp, sp, #4
        sub     sp, sp, #24
        str     r0, [fp, #-20]
        str     r1, [fp, #-16]
        str     r2, [fp, #-28]
        str     r3, [fp, #-24]
        sub     r3, fp, #28
        ldmia   r3, {r2-r3}
        sub     r1, fp, #20
        ldmia   r1, {r0-r1}
        bl      __aeabi_dadd
        mov     r2, r0
        mov     r3, r1
        str     r2, [fp, #-12]
        str     r3, [fp, #-8]
        sub     r3, fp, #12
        ldmia   r3, {r2-r3}
        mov     r0, r2
        mov     r1, r3
        sub     sp, fp, #4
        @ sp needed
        pop     {fp, pc}
        .size   sum, .-sum
        .align  2
        .global main
        .syntax unified
        .arm
        .fpu softvfp
        .type   main, %function
```

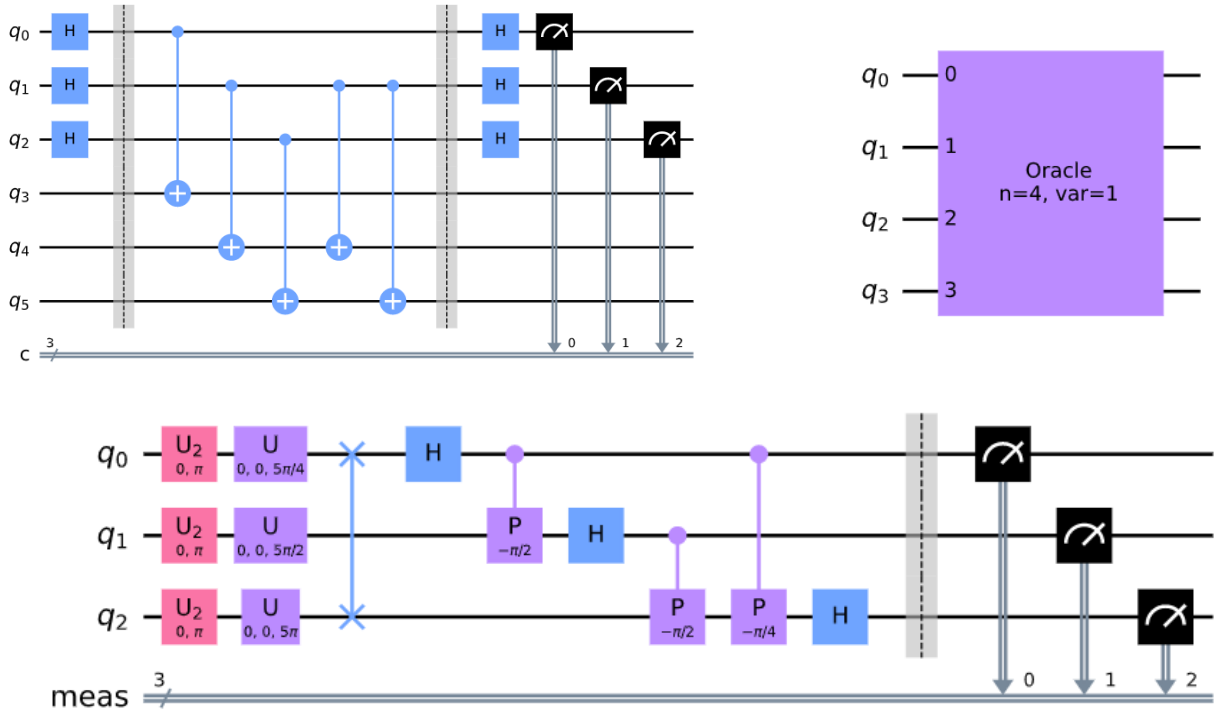
In an ideal QC, we could also perform as many abstractions as we want







Most QC implementations nowadays lack HW abstraction capabilities

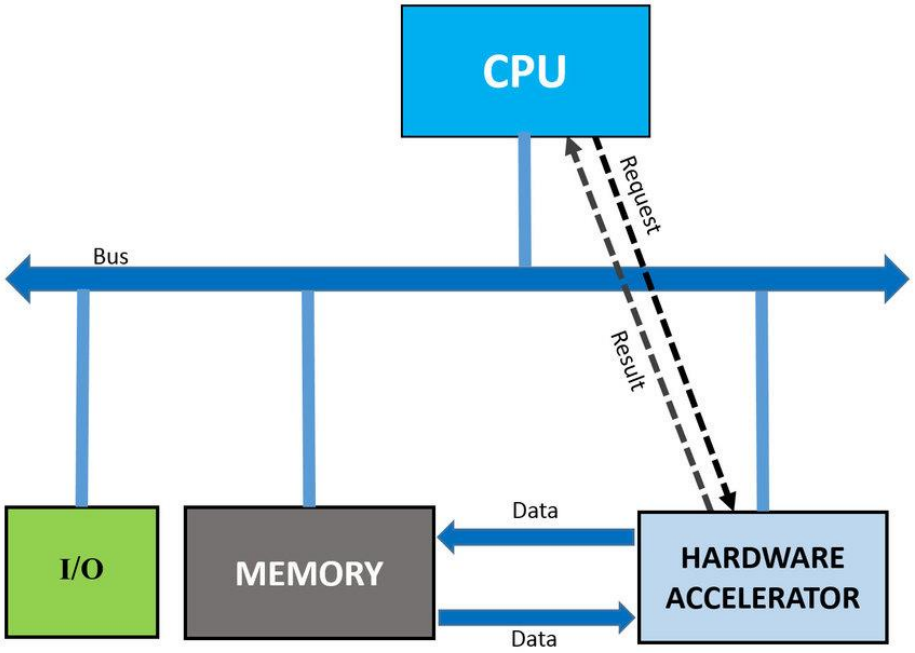


Low HW capabilities (# qubits, interconnections, coherence time)

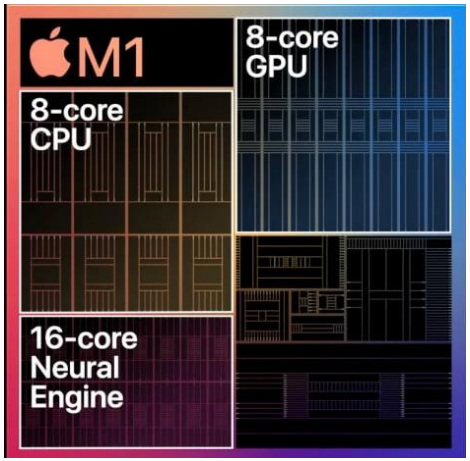
Embedded systems are closer to current QCs more than CPUs, from a resources perspective

Operation frequency	3.60 GHz	20 MHz
SRAM	16 MB	2 kB
Number of pins	1151	28
Number of cores	8	1

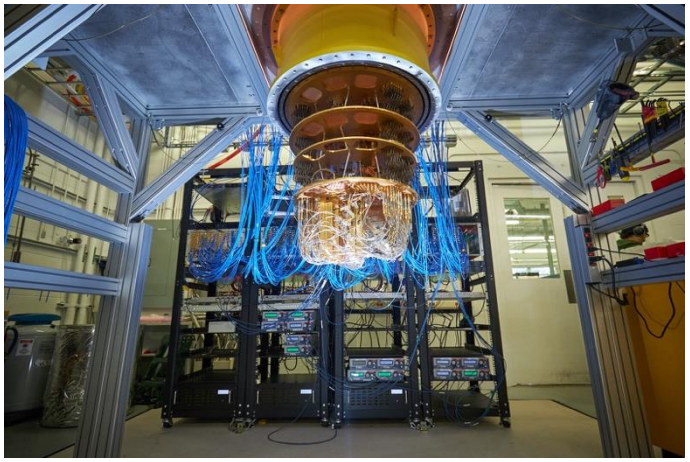
Accelerator devices are closer to current QCs more than CPUs, from a functional perspective



Apple M1



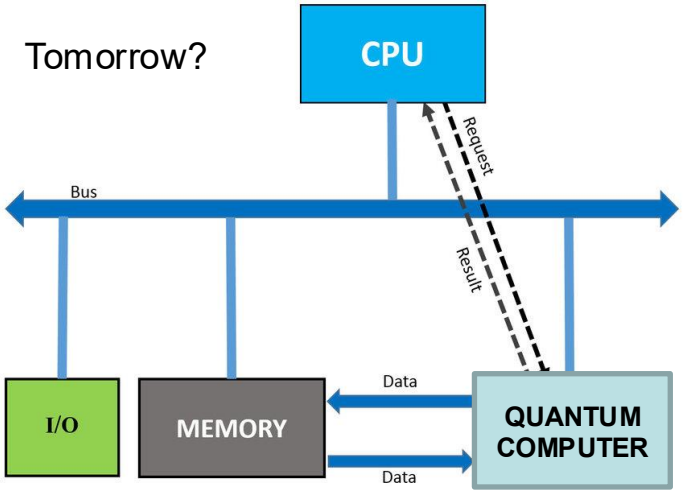
QCs aspire to be ultra-powerful HW-accelerators… in the future



Today:

Promise of Value

Tomorrow?



Limitations of QC restrict the architecture capabilities of useful quantum SW

Architectural characteristic	Today	Tomorrow
Availability	Limited (cloud)	Yes
Performance	Demonstrated for specific problems	Problem-dependent, but extraordinary
Robustness	Emerging (QEC demos)	Very high
Correctness	Very low	High
Privacy	?	?
Localization	?	?

The QEC Context for SW Developers

- **What QEC means for quantum software developers**

- **Logical vs. Physical Qubits**
  - • A logical qubit = many physical qubits working together for error protection
  - • Surface code: ~1000 physical qubits per logical qubit (current estimate)
  - • qLDPC codes (IBM, 2025): ~100 physical per logical — 10x improvement

- **Google Willow Result (Dec 2024)**
  - • First time: increasing code distance DECREASED logical error rate
  - • Distance-7 surface code: 0.143% error per cycle
  - • Logical memory lifetime exceeded best physical qubit by 2.4x

- **What this means for resource estimation**
  - • Your 9-qubit TSP needs ~9,000 physical qubits with surface codes
  - • Or ~900 with next-gen qLDPC codes
  - • Resource estimation is now a critical part of QSDL feasibility study

- The gap between logical and physical qubits is THE bottleneck for practical quantum SW.

From feasibility to quality management — a structured approach

QUANTUM SOFTWARE DEVELOPMENT LIFECYCLE

Disclaimer: most of the information here can be found on this paper

QDLC - The Quantum Development Life Cycle

Nivedita Dey, *QRDLab India, University of Calcutta,* Mrityunjay Ghosh, *QRDLab India, University of Calcutta,* Subhra Samir kundu, *QRDLab India* and Amlan Chakrabarti, *University of Calcutta*

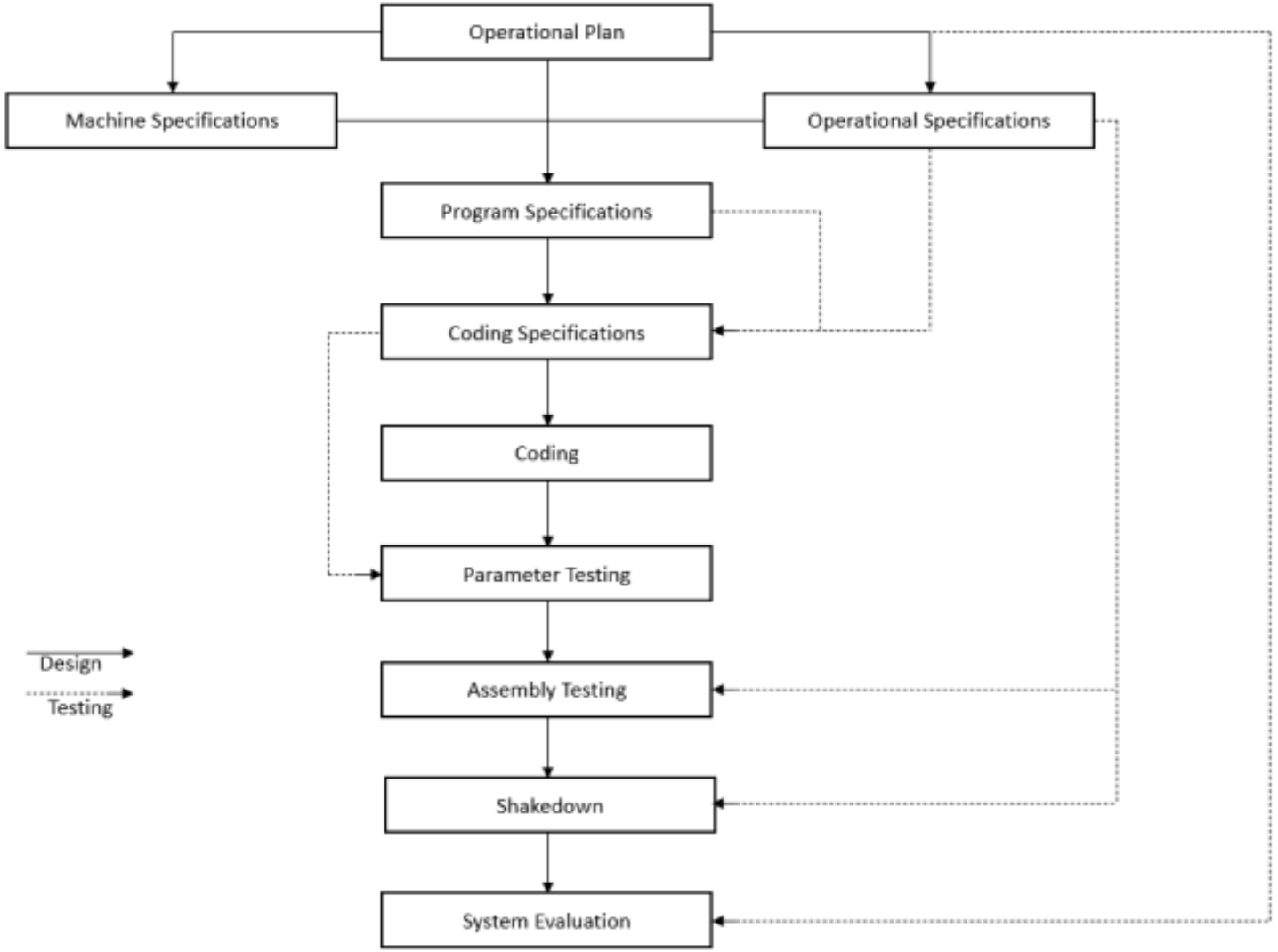
Abstract—The magnificence grandeur of quantum computing lies in the inherent nature of quantum particles to exhibit true parallelism, which can be realized by indubitably fascinating theories of quantum physics. The possibilities opened by quantum computation (QC) is no where analogous to any classical simulation as quantum computers can efficiently simulate the complex dynamics of strongly correlated inter-facial systems. But, unfolding mysteries and leading to revolutionary breakthroughs in quantum computing are often challenged by lack of research and development potential in developing qubits with longer coherence interval, scaling qubit count, incorporating quantum error correction to name a few. Putting the first footstep into explorative quantum research by researchers and developers is also inherently ambiguous – due to lack of definitive steps in building up a quantum enabled customized computing stack. Difference in behavioral pattern of underlying system, early-stage noisy device, implementation barriers and performance metric cause hindrance in full adoption of existing classical SDLC suites for quantum product development. This in turn, necessitates to devise systematic and cost-effective techniques to quantum software development through a Quantum Development Life Cycle (QDLC) model, specifying the distinguished features and functionalities of quantum feasibility study, quantum requirement specification, quantum system design, quantum software coding and implementation, quantum testing and quantum software quality management.

Index Terms—Quantum Computing, Quantum Development Life Cycle, Quantum SDLC, QDLC, Quantum software development, Quantum waterfall model.

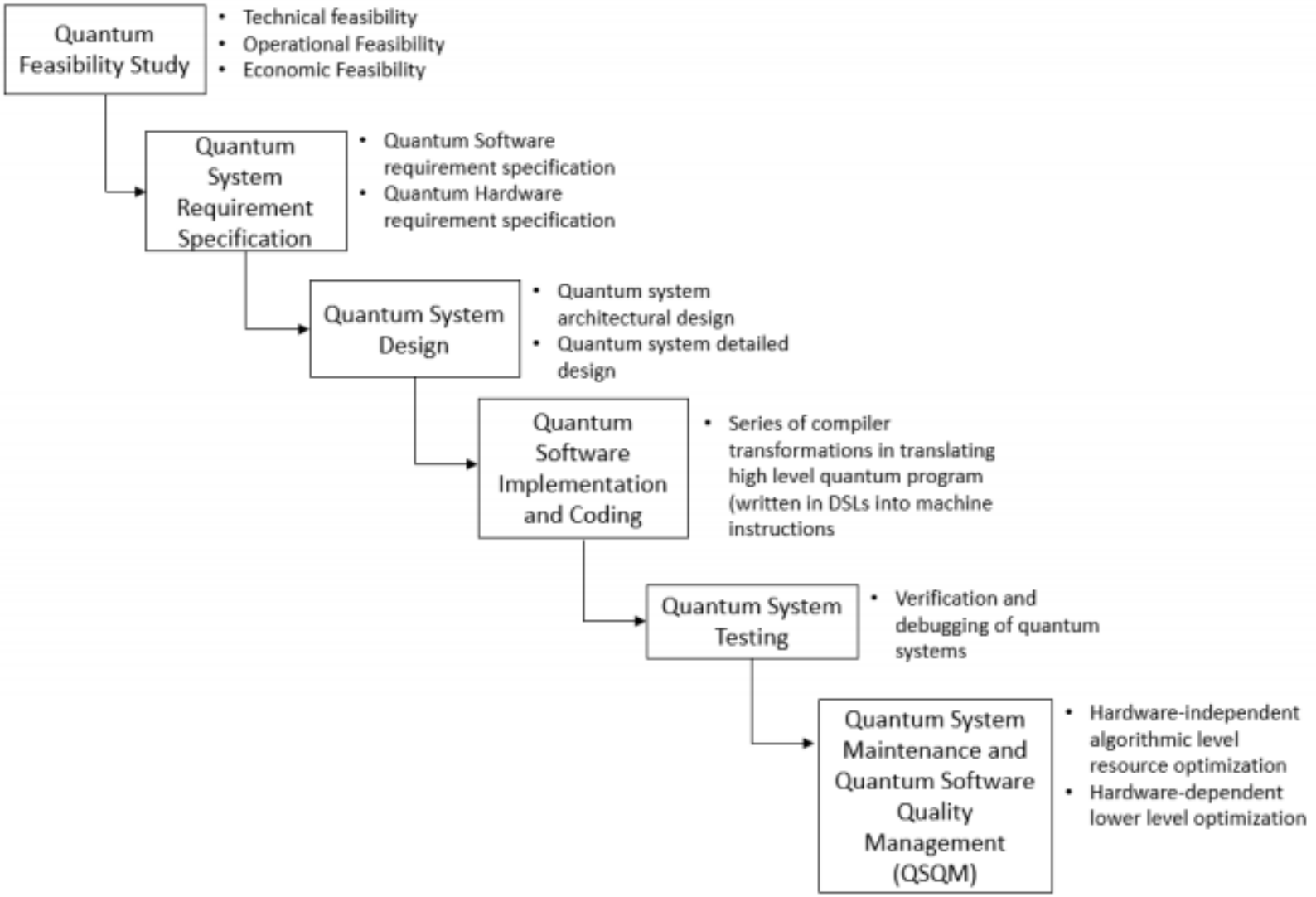
arXiv:2010.08053 (Oct. 2020)

[Foundational reference, 2020. We update with 2024-2026 developments throughout this class.]

Classical SW Development Lifecycle



Adapting SW design to Quantum Computing: the Quantum Software Development Lifecycle



QSDL: Feasibility Study

Operational Feasibility

Is the desired workload viable on a Quantum Computer?

Do algorithms for the current workload exist?

Do we have the required skillset to handle the quantum SW project?

Economic Feasibility

Does it really provide an improvement? (Time / Accuracy / Enabler → Money?)

Is the desired quantum SW project profitable?

Short-term: early-adopters, corporate readiness

Long-term: Promise of Value

Technical Feasibility

Can current quantum HW run the proposed workload?

QSDL: Feasibility — Hardware Landscape 2026

- **Three tiers of quantum hardware in 2026**
- **Tier 1: NISQ Devices (100-1000 noisy qubits)**
  - • IBM Heron (156q), Google Willow (105q), IonQ Forte (36q), Quantinuum H2 (56q)
  - • Use case: variational algorithms, small-scale optimization, proof of concept
  - • Limitation: noise limits circuit depth to ~100-300 gate layers
- **Tier 2: Early Fault-Tolerant (demonstration scale)**
  - • Google Willow QEC experiments, IBM Kookaburra (2026), Microsoft Majorana 1
  - • Use case: QEC research, logical qubit demonstrations
  - • Limitation: few logical qubits, high overhead
- **Tier 3: Quantum Annealers (5000+ qubits)**
  - • D-Wave Advantage2: 5000+ qubits, Pegasus/Zephyr topology
  - • Use case: QUBO optimization (logistics, scheduling, finance)
  - • Limitation: no universal computation, specific problem types only

Feasibility depends on matching your problem to the right tier.

QSDL: Requirement specification

Software requirements

- Quantum tools
  - Quantum programming languages & algorithm libraries
  - More on this later
- Integrator plugins
  - Adaptors to introduce the quantum component in the whole workloads
- Logical circuit synthesizer
  - Optimally convert the workload into a quantum circuit
- Classical validation
  - Tests to verify the correct implementation of the quantum workload

QSDL: Requirement specification

Hardware requirements (critical for operational and functional feasibility)

Qubit count & interconnectivity

Quantum volume

$$V_Q = \max_{n<N} \left\{ \min \left[n, \frac{1}{n\epsilon_{\text{eff}}(n)} \right]^2 \right\}$$

Physical machine description

Underlying physical process of the HW

Classical HW processor

Allows using the QC as a HW accelerator

QSDL: Quantum System Testing

Can we verify, *without interfering*, the result of a quantum computation?

Checking the results?

- Classical systems cannot verify the results (computing deficiency)
- Results of a QC program are inherently random

Reconstructing the final quantum state?

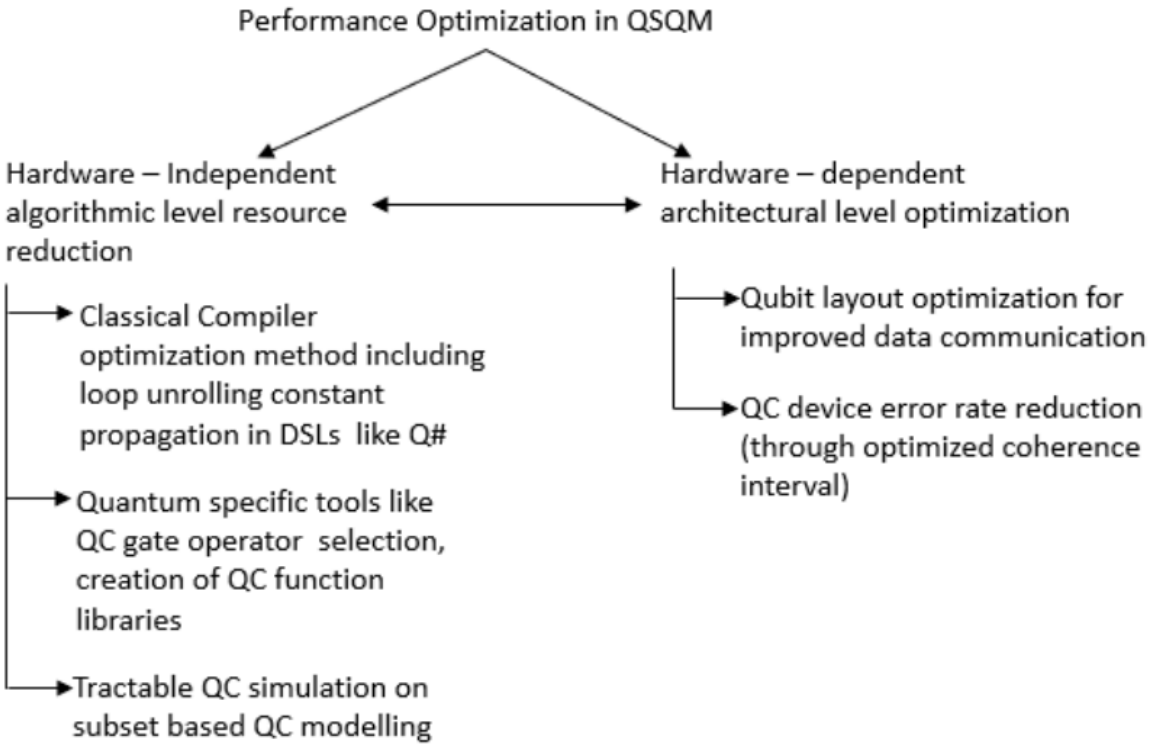
- Quantum state tomography
- May not be viable for large qubits

Specific testing procedures?

- Only available for a certain set of problems
- Require access to the inner quantum modules

QSDL: Quantum Software Quality Management

Optimization of the resources employed by the quantum workload



Four characteristics unique to quantum programming:

1. Commuting / Non-commuting Operations

Order of operations matters for optimization.

Changing order → trade-offs or errors.

2. Clean & Borrowed Qubits

Intermediate qubits must be returned to their original state.

Resource management: not all qubits are isolated.

3. Adjoint Operators

Required to restore system state (e.g., returning borrowed qubits).

Useful for optimization: operation + adjoint = identity.

4. Controlled Operators

Quantum 'if' statement — both branches execute in superposition.

Reordering these operations may enable further optimizations.

These characteristics are language-agnostic — they apply to ALL quantum SDKs.

Quantum SW Design

Quantum modules required

Control relationships between classical and quantum modules

Interfaces among modules (C-C, C-Q, Q-Q)

Algorithms

Data Structures

Quantum Modules

Collectivity of related entities (cohesion)

Functional Logical Temporal

Quantum temporal cohesion

Module may act on superposition states

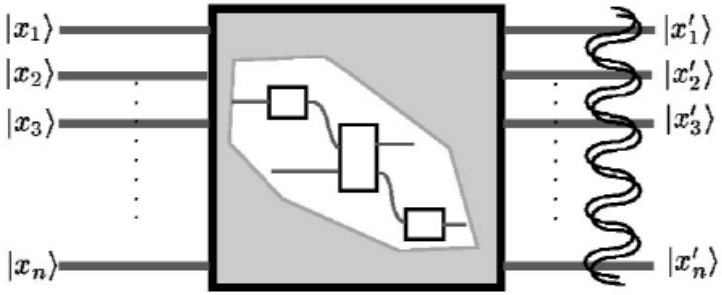
Quantum spatial cohesion

Interaction between close qubits

Interact between them (coupling)

Classical Quantum

Algorithms are the foundational part of any program, either classical or quantum



Classification of (classical) algorithms

Problem: given a, n, compute a^{2n}

Algorithmic technique	Example
Brute force	$a^{2n} = a \cdot a \cdot \cdots (2n \text{ times}) \cdots \cdot a$
Divide-and-conquer	$a^{2n} = a^n \cdot a^n$
Decrease-and-conquer	$a^{2n} = a^2 \cdot a^{2(n-1)}$
Transform-and-conquer	$\log(a^{2n}) = 2n \cdot \log a$

Quantum algorithms excel in different workloads

Oracle-based algorithms

Hidden subgroup problem	Factorization (Shor)
Amplitude amplification	Unstructured search (Grover)

Linear algebra problems

Quantum Fourier Transform	Fourier transformation in Hilbert space
Phase estimation	Eigenvalue decomposition
HHL algorithm	Solving linear systems of equations

Simulation of quantum processes

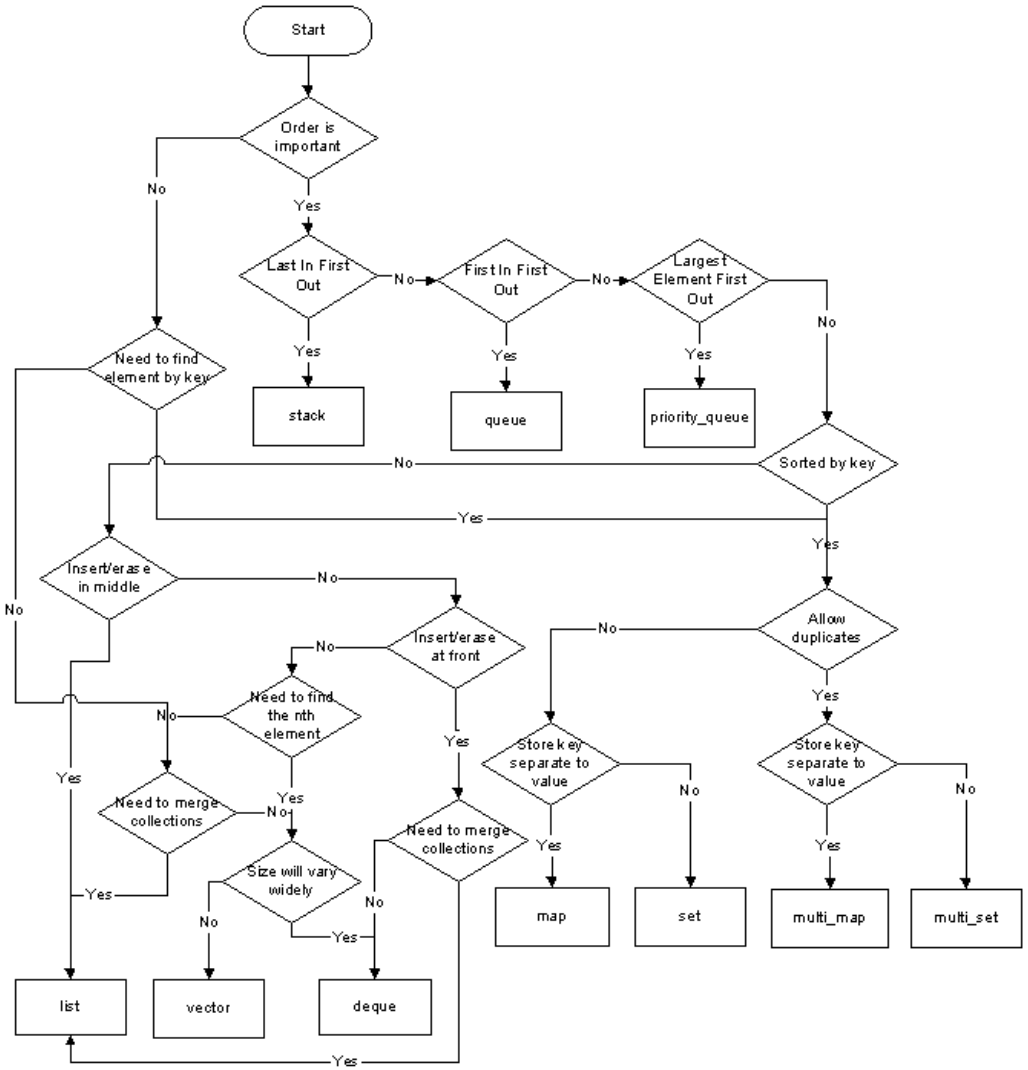
Quantum annealing	High-speed optimization
Quantum simulators	Specific quantum process simulation

Each of these algorithms are decomposed in fundamental, model-dependent, quantum subroutines

Gate-based systems

Initialization of Registers	Setting up and initializing the quantum registers and classical registers required for computation.
Uniform Superposition	Applying Hadamard transform to achieve an uniform superposition of quantum states.
Entanglement	Entangling multiple qubits to form an entangled quantum state.
Creation of function table	Analyzing quantum algorithm in a query model in order to compute boolean functions, where inputs are given in a black box aiming to compute the function value for an arbitrary input string with fewer number of queries.
Oracle Setup	Setting up an oracle, which us a black box operation and can be used in computation which is obtained as input from another quantum algorithm.
Uncompute Task	Applying reversible computing logic to remove entanglement from resulting state.
Phase Shift	In order to identify and mark a single quantum state from the superposed set of quantum states, controlled phase shift operation is performed using unitary quantum gates.
Amplitude Amplification	A tool of choice for quantum algorithm designers such that the success probability of query algorithms can be increased.
Speedup via verification	Performing quantum verification and achieving substantial speedup.
Classical Measurement	Splitting up the quantum and classical states by performing a classical measurement.

Data structures are as important as algorithms for any program



Hybrid Architectures: Classical-Quantum Integration 2026

- **Three hybrid classical-quantum integration patterns in 2026**

- **Pattern 1: Variational Loop**
  - • Classical optimizer ↔ Quantum circuit (QAOA, VQE)
  - • The quantum computer evaluates, the classical computer optimizes
  - • Dominant pattern for NISQ-era applications

- **Pattern 2: Cloud Multi-Provider Orchestration**
  - • Azure Quantum / Amazon Braket: submit jobs to IonQ, Quantinuum, Rigetti, etc.
  - • Same code, different backends. Best-fit hardware selection.
  - • Managed job scheduling, queuing, and result retrieval.

- **Pattern 3: GPU-QPU Co-Processing (CUDA-Q)**
  - • NVIDIA DGX Quantum: GPU + QPU on same system
  - • GPU handles classical pre-processing, simulation, and optimization loops
  - • QPU handles the quantum kernel. Seamless C++/Python integration.
  - • PsiQuantum achieved 450x speedup with CUDA-Q multi-GPU simulation.

Abstraction Levels in Quantum Programming

- **Five levels of abstraction — and which tools operate at each:**

- **Pulse Level (lowest)**
- Direct hardware control signals. Tools: Qiskit Pulse, Qblox

- **Gate Level**
- Quantum logic gates (H, CNOT, T, etc.). Tools: Cirq, Q#

- **Circuit Level**
- Composed gate sequences. Tools: Qiskit, Cirq, PennyLane

- **Algorithm Level**
- High-level quantum algorithms (QAOA, VQE, Grover). Tools: PennyLane, Qiskit

- **Application Level (highest)**
- Problem-oriented (optimize, simulate, factor). Tools: Classiq, D-Wave Ocean

- Trend: frameworks moving UP the stack. Developers want application-level, not pulse-level.

The Big Six: Side-by-Side Comparison

Framework	Language	Abstraction	HW Targets	Key Strength
Qiskit 1.0 (IBM)	Python	Gate/Circuit (Primitives API)	IBM QPUs (127-156 qubits)	Largest ecosystem, runtime optimization
Q# (Microsoft)	Q# / Python	Gate/Algorithm (standalone)	IonQ, Quantinuum, PASQAL, Rigetti	Resource Estimator, multi-provider
Cirq (Google)	Python	Gate/Circuit (HW-native)	Google QPUs (Willow 105q)	Low-level control, QEC research
PennyLane (Xanadu)	Python	Algorithm (differentiable)	All major HW (plugin system)	Auto-differentiation, ML integration
Amazon Braket	Python	Circuit (multi-provider)	IonQ, Rigetti, IQM, QuEra	AWS integration, managed infra
CUDA-Q (NVIDIA)	C++ / Python	Kernel (GPU-QPU hybrid)	All major HW (via plugins)	GPU acceleration, hybrid programming

Q# & Azure Quantum Marketplace

- **Q#: standalone quantum language (no longer needs C# host)**

- • Open-source, hardware-agnostic quantum programming language
- • VS Code extension with IntelliSense, debugging, resource estimation
- • Runs locally or submits to Azure Quantum cloud

- **Azure Quantum Marketplace**
- • Multi-provider hardware access from a single workspace:
- • IonQ (trapped ion, 36q fully connected)
- • Quantinuum (trapped ion, high fidelity, mid-circuit measurement)
- • PASQAL (neutral atom, room temperature)
- • Rigetti (superconducting, fast gate times)

- **Killer Feature: Resource Estimator**
- • Estimate physical resources BEFORE running on hardware
- • Supports 6 qubit types, 2 QEC schemes (surface code, floquet code)
- • Critical for QSDL feasibility studies

- **Majorana 1 (Feb 2025): topological qubits → Azure Quantum roadmap**

Cirq & Google Quantum AI

- **Cirq: low-level circuit manipulation for quantum research**

- • Python framework optimized for NISQ circuits on Google hardware
- • Hardware-native gateset: direct access to Google's qubit topology
- • Fine-grained control over qubit placement, scheduling, noise

- **Google Willow (December 2024)**
- • 105-qubit superconducting chip with real-time QEC
- • Below-threshold error correction: logical errors decrease with more qubits
- • 0.143% error per cycle (distance-7 surface code)
- • Random Circuit Sampling: benchmark performance beyond classical reach

- **Ecosystem**
- • TensorFlow Quantum (TFQ): quantum machine learning
- • Cirq-FT: fault-tolerant circuit primitives
- • OpenFermion: quantum chemistry simulations

- **Best for: QEC research, hardware-native optimization, low-level control**

PennyLane & Amazon Braket

- **PennyLane (Xanadu): differentiable quantum computing**

- • Treats quantum circuits as differentiable functions
- • Automatic gradient computation (parameter-shift rule, backprop, adjoint)
- • Native integration with PyTorch, JAX, TensorFlow
- • Plugin system: runs on Qiskit, Cirq, Braket, all major hardware
- • Catalyst compiler: JIT compilation for efficient execution
- • Best for: variational algorithms, quantum ML, hybrid optimization

- **Amazon Braket: AWS multi-provider platform**

- • Unified SDK for multiple hardware backends:
- • IonQ (trapped ion), Rigetti (superconducting), IQM, QuEra (neutral atom)
- • AWS Ocelot: cat qubits (announced Feb 2025)
- • Managed Jupyter notebooks, job management, S3 result storage
- • Pay-per-shot pricing model
- • Integrated with PennyLane, Qiskit, CUDA-Q
- • Best for: cloud-native workflows, multi-provider evaluation

CUDA-Q & GPU-Accelerated Quantum Computing

- **NVIDIA CUDA-Q: hybrid classical-quantum GPU ecosystem**

- **Programming Model**
- • Single-source C++/Python: __qpu__ annotations for quantum kernels
- • Seamless composition with CUDA, OpenMP, OpenACC
- • GPU accelerates simulation + classical portions of hybrid algorithms

- **Key Results**
- • cuQuantum: GPU-accelerated circuit simulation during development
- • PsiQuantum integration: 450x speedup on multi-GPU nodes
- • Classiq integration: 67 min → 2.5 min for 31-qubit circuit (A100 GPU)

- **DGX Quantum**
- • GPU + QPU on same system for ultra-low latency hybrid workloads
- • ORNL integration with Frontier supercomputer (2026)

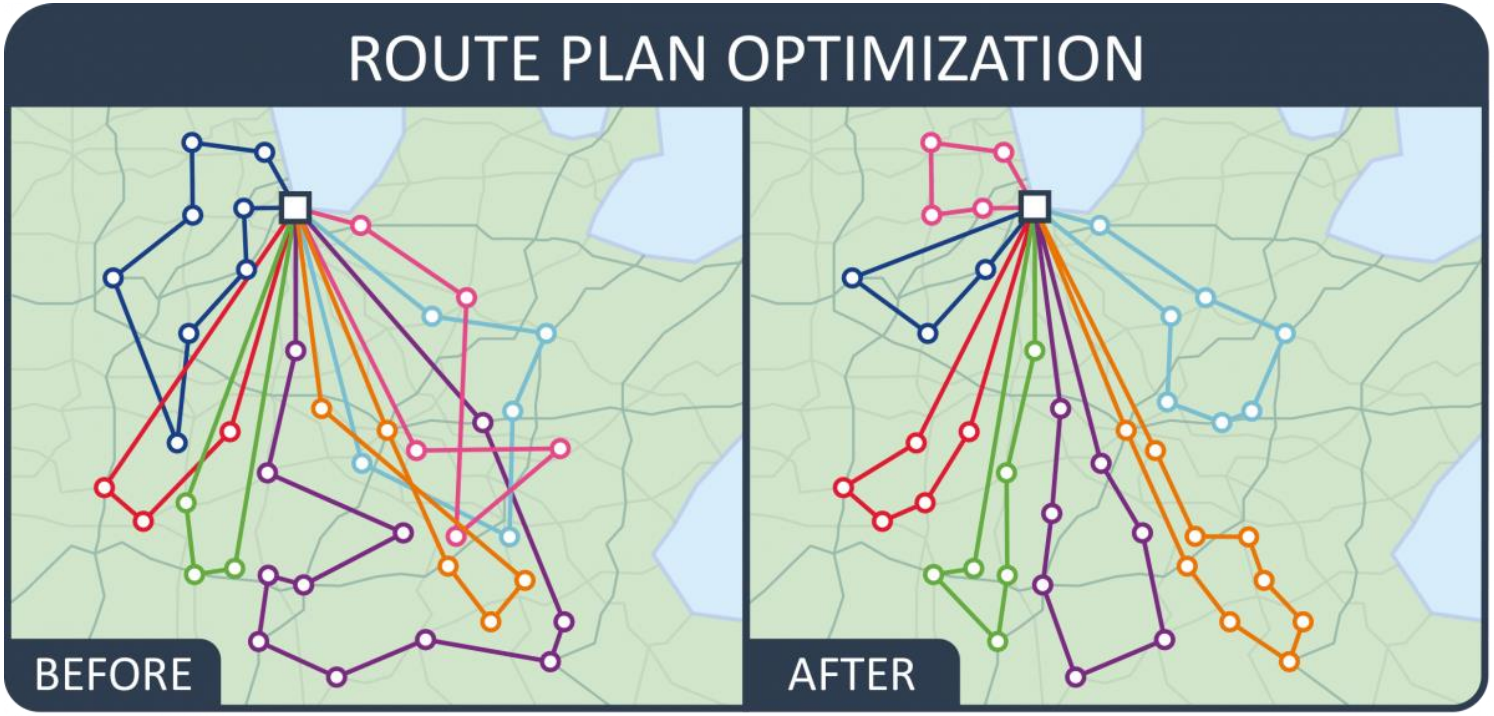
- **Also notable: Classiq (AI-assisted quantum development)**
- • High-level quantum modeling (Qmod language), $173M funded
- • v1.0 released Feb 2026. Enterprise-grade platform.

When to Use What: A Decision Framework

Choosing the right quantum tool for your problem

- **Optimization (QUBO, routing, scheduling):**
- • D-Wave Ocean → quantum annealing (production-ready for QUBO)
- • Qiskit QAOA / PennyLane QAOA → gate-based (research, scaling potential)
- **Quantum Chemistry (molecular simulation):**
- • PennyLane + OpenFermion → VQE with gradients
- • Q# → resource estimation for fault-tolerant algorithms
- **Machine Learning (QML):**
- • PennyLane → differentiable circuits + PyTorch/JAX integration
- • TensorFlow Quantum (Cirq) → TF ecosystem
- **Hardware research / QEC:**
- • Cirq → hardware-native gates, Google hardware access
- **Multi-provider evaluation / cloud deployment:**
- • Amazon Braket or Azure Quantum → single API, multiple backends
- **Heavy simulation during development:**
- • CUDA-Q → GPU-accelerated simulation, scales to 30+ qubits

Problem statement



QSDL Step 1: Feasibility Study for VRP

- **Applying the QSDL Feasibility Framework to Vehicle Routing**

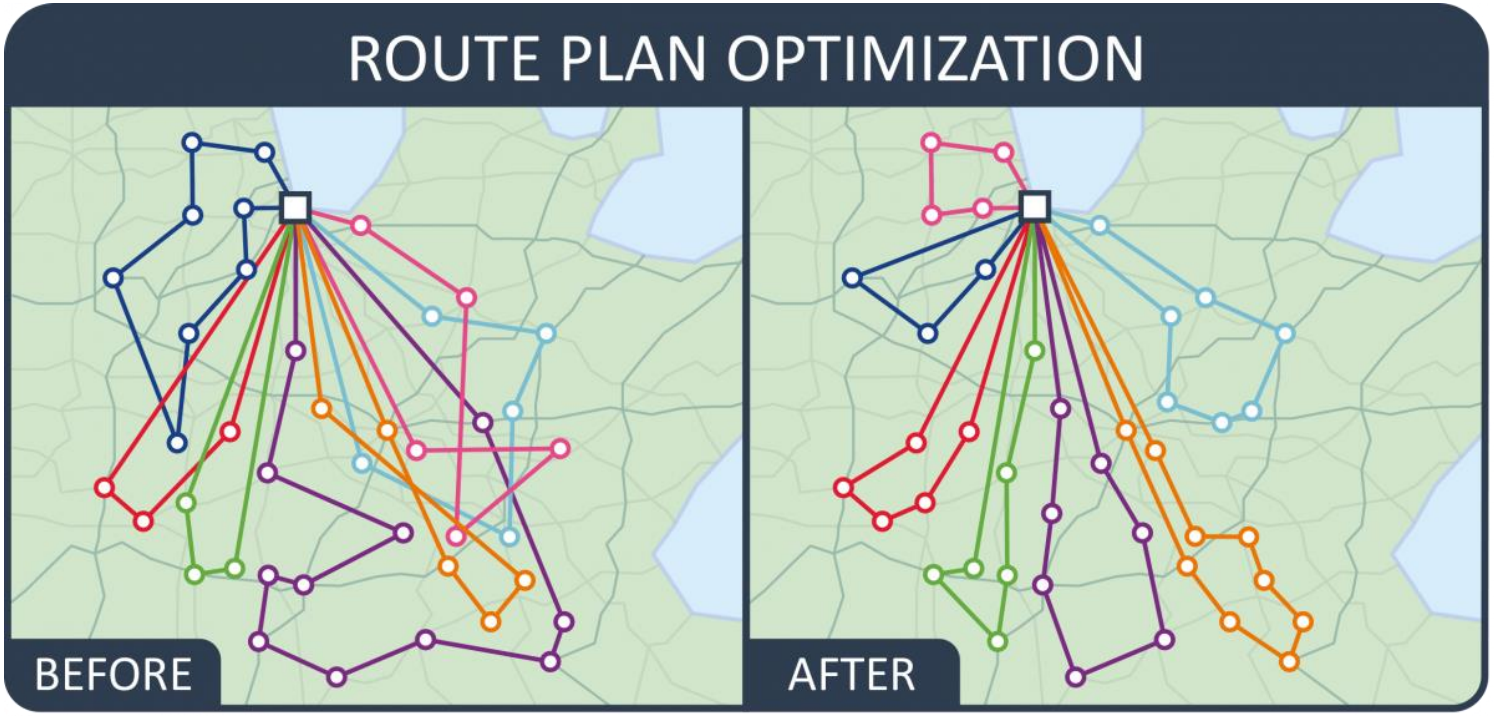
- **Operational Feasibility**
  - • Is VRP a suitable quantum workload? YES — it maps to QUBO/Ising naturally
  - • Algorithms exist: QAOA (gate-based), quantum annealing (D-Wave)
  - • Team skills needed: QUBO formulation, quantum SDK programming, classical optimization

- **Technical Feasibility**
  - • 3 cities → 9 qubits (simulator). 10 cities → 100 qubits (current HW edge).
  - • 20+ cities → requires QEC. Not feasible on today's hardware for production.
  - • Hybrid approach: quantum for the hard combinatorial core, classical for pre/post-processing.

- **Economic Feasibility**
  - • Small instances: classical solvers (OR-Tools, Gurobi) are faster and cheaper
  - • Value proposition: quantum advantage expected for large, constrained instances (50+ cities)
  - • Current ROI: negative for production. Positive for R&D and preparing for quantum readiness.

Problem formulation

Objective: Minimize total cost of the route distribution



Restrictions:

Every stop must be visited only once

Mathematical formulation

Objective: Minimize total cost of the route distribution

$$C_{\mathrm{T}} = \sum_{r=1}^{N_{\mathrm{R}}} C_r \qquad C_r = \sum_{i=1}^{N_{\mathrm{C}}} \sum_{j=1}^{N_C} d_{ij} \cdot l_{rij} \qquad l_{rij} = \sum_{k=1}^{S_r} x_{ri(k)} \cdot x_{rj(k+1)}$$

C_{T}: total cost

C_r: cost of route r

N_{R}: number of routes

N_C: number of cities

d_{ij}: distance between cities i and j

l_{rij}: Boolean: route r links cities i and j

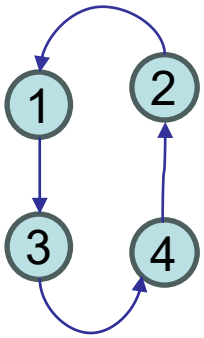
S_r: number of stops in route r

$x_{ri(k)}$: Boolean: route r visits city i in its k-th stop

Mathematical formulation

Objective: Minimize total cost of the route distribution

$$C_{\mathrm{T}} = \sum_{r=1}^{N_{\mathrm{R}}} C_r \qquad C_r = \sum_{i=1}^{N_{\mathrm{C}}} \sum_{j=1}^{N_C} d_{ij} \cdot l_{rij} \qquad l_{rij} = \sum_{k=1}^{S_r} x_{ri(k)} \cdot x_{rj(k+1)}$$



$x_{1,1,1}$

$x_{1,3,2}$

$x_{1,4,3}$

$x_{1,2,4}$

$l_{1,1,3}$

$l_{1,3,4}$

$l_{1,4,2}$

$$C_r = d_{13} + d_{34} + d_{42}$$

Mathematical formulation

Restrictions:

Every stop must be visited only once

$$\sum_{r=1}^{N_{\mathrm{R}}} \sum_{k=1}^{S_r} x_{ri(k)} = 1 \quad \forall i \in \{1 \dots N_{\mathrm{C}}\}$$

Every route r must have a k-th step

$$\sum_{i=1}^{N_{\mathrm{C}}} x_{ri(k)} = 1 \quad \forall r \in \{1 \dots N_{\mathrm{R}}\}, \forall k \in \{1 \dots S_{\mathrm{r}}\}$$

Avoids the appearance of jumps (empty steps)

Avoids the same step appearing more than once in the loop

Mathematical formulation

Objective: Minimize total cost of the route distribution

$$C_{\mathrm{T}} = \sum_{r=1}^{N_{\mathrm{R}}} \left\{ \sum_{i=1}^{N_{\mathrm{C}}} \sum_{j=1}^{N_C} d_{ij} \cdot \left[\sum_{k=1}^{S_r} x_{ri(k)} \cdot x_{rj(k+1)} \right] \right\}$$

Restrictions:

$$\sum_{i=1}^{N_{\mathrm{C}}} \left(1 - \sum_{r=1}^{N_{\mathrm{R}}} \sum_{k=1}^{S_r} x_{ri(k)} \right)^2 + \sum_{r=1}^{N_{\mathrm{R}}} \sum_{k=1}^{S_r} \left(1 - \sum_{i=1}^{N_{\mathrm{C}}} x_{ri(k)} \right)^2$$

Hamiltonian formulation

$$H = \sum_{r=1}^{N_R} \left\{ \sum_{i=1}^{N_C} \sum_{j=1}^{N_C} d_{ij} \cdot \left[\sum_{k=1}^{S_r} x_{ri(k)} \cdot x_{rj(k+1)} \right] \right\}$$
$$+\lambda \cdot \left\{ \sum_{i=1}^{N_C} \left(1 - \sum_{r=1}^{N_R} \sum_{k=1}^{S_r} x_{ri(k)} \right)^2 + \sum_{r=1}^{N_R} \sum_{k=1}^{S_r} \left(1 - \sum_{i=1}^{N_C} x_{ri(k)} \right)^2 \right\}$$

Variables: $x_{ri(k)} \in \{0, 1\}$

Parameters: N_C, $\{d_{ij}\}_{i,j=1\ldots N_C}$

Hyperparameters: N_R, $\{S_r\}_{r=1\ldots N_R}$

λ (penalty hyperparameter)

Resource estimation & feasibility study

Number of (logical) qubits:

$$\#\left(x_{ri(k)}\right) \sim O(N_{\mathrm{R}} \cdot N_{\mathrm{C}} \cdot \max\{S_r\})$$

NR	NC	max(Sr)	#Q
1	5	5	25
2	5	3	30
3	10	4	120
1	10	1	10
7	20	8	1120

Connectivity:

$x_{ri(k)} \cdot x_{rj(k+1)}$, connected via cost function
$O\left(N_{\mathrm{C}}^2 \cdot \max\{S_r\}\right)$ connections

Design & deployment

Problem parameters: N_C, $\{d_{ij}\}_{i,j=1\ldots N_C}$

Parametric study

Select hyperparameters: N_R, $\{S_r\}_{r=1\ldots N_R}$

λ (penalty hyperparameter)

Build the hamiltonian: H

Feed into the Quantum Machine of choice:

D-Wave: https://docs.dwavesys.com/docs/latest/c_gs_3.html

Qiskit: https://qiskit.org/textbook/ch-applications/qaoa.html

Check constraints $\sum_{i=1}^{N_C}\left(1-\sum_{r=1}^{N_R}\sum_{k=1}^{S_r}x_{ri(k)}\right)^2+\sum_{r=1}^{N_R}\sum_{k=1}^{S_r}\left(1-\sum_{i=1}^{N_C}x_{ri(k)}\right)^2=0$

Return solution

Implementation: Qiskit 1.0 Code Walkthrough

- **Solving TSP with Qiskit 1.0 Primitives (key code snippets)**

- **Step 1: Build QUBO from problem**
  - • tsp = Tsp(distance_matrix)
  - • qp = tsp.to_quadratic_program()
  - • qubo = QuadraticProgramToQubo().convert(qp)

- **Step 2: Solve with QAOA using new Primitives API**
  - • sampler = StatevectorSampler() # NEW in Qiskit 1.0
  - • qaoa = QAOA(sampler=sampler, optimizer=COBYLA(), reps=2)
  - • result = MinimumEigenOptimizer(qaoa).solve(qubo)

- **Step 3: Interpret results**
  - • route = tsp.interpret(result)
  - • # Returns city visit order: [0, 2, 1] → A → C → B → A

- Key change from pre-1.0: execute() → Primitives (Sampler/Estimator)
- Full runnable code in companion Jupyter notebook.

Implementation: Multi-Framework Comparison

- **Same TSP problem, three quantum approaches**

- **Qiskit 1.0 (QAOA)**
  - • Problem-specific ansatz. Uses Sampler primitive.
  - • Gradient-free optimizer (COBYLA). ~2 min on 9 qubits.

- **PennyLane (VQE)**
  - • Hardware-efficient ansatz. Auto-differentiation through circuit.
  - • Gradient-based optimizer (Adam). ~6 sec on 9 qubits.
  - • Strength: differentiable quantum computing, ML integration.

- **D-Wave Ocean (Simulated Annealing)**
  - • No circuit design needed. Just submit the QUBO.
  - • 1000 samples in ~50 ms. Near-100% optimal rate.
  - • Strength: simplicity, speed, no circuit optimization.

- All three found the optimal route. D-Wave fastest. See Jupyter notebook for full code.

Results & Discussion

- **Honest assessment: when does quantum help?**

- **Small instances (n < 10): Classical wins**
  - • Brute force or classical heuristics solve instantly
  - • Quantum overhead (circuit compilation, shots, optimization) not worth it

- **Medium instances (10-50): Hybrid potential**
  - • Classical solvers struggle with highly constrained variants
  - • Quantum annealers (D-Wave) show competitive results for QUBO problems
  - • QAOA/VQE need error-corrected hardware to scale beyond ~25 qubits

- **Large instances (n > 50): Quantum advantage expected — but not yet**
  - • Requires fault-tolerant QC with thousands of logical qubits
  - • Google Willow (2024) showed the path — but we're not there yet

- Further study: arXiv:2502.17725 — detailed VRP analysis on quantum hardware

Would you recommend quantum to a logistics client today?

QEC Breakthroughs: The Turning Point

- **Three breakthroughs that changed the game (2024-2025)**

- **Google Willow (December 2024)**
  - • 105-qubit chip: first below-threshold error correction demonstration
  - • Logical error rate DECREASES as code distance increases (distance-7 surface code)
  - • Logical qubit lifetime exceeded best physical qubit by 2.4x
  - • Physics World 2024 Breakthrough of the Year

- **IBM qLDPC Codes (2024-2025)**
  - • Quantum Low-Density Parity-Check codes: 10x reduction in qubit overhead vs. surface codes
  - • Kookaburra processor (2026): first practical QEC module in production roadmap

- **Microsoft Majorana 1 (February 2025)**
  - • 8-qubit topological processor using Majorana Zero Modes
  - • Claim: inherently error-resistant by design (information stored non-locally)
  - • Controversy: peer review ongoing. Digital control model (simpler than analog).

- Impact: Fault-tolerant QC is now an engineering problem, not a physics question.

Hardware Roadmaps: Where Are We Heading?

Provider	Technology	Current	Target	Timeline
IBM	Superconducting	156q (Heron r3)	100,000+ qubits	2033
Google	Superconducting	105q (Willow)	Error-corrected QC	2029
IonQ	Trapped ion	36q (#AQ 36)	1024+ #AQ	2028
Quantinuum	Trapped ion	56q (H2)	~10,000 qubits	2030s
Microsoft	Topological	8q (Majorana 1)	1M qubits on chip	TBD
D-Wave	Annealing	5000+q (Adv2)	Next-gen annealer	2026
PsiQuantum	Photonic	Pre-production	1M+ photonic qubits	2027+

What Changes for QC Software Developers?

Five shifts every quantum software developer must prepare for

- **1. Resource estimation becomes a critical skill**
  - • How many physical qubits for N logical qubits? What QEC code? What error budget?
  - • Tools: Azure Quantum Resource Estimator, Qiskit transpiler benchmarks

- **2. QEC-aware programming**
  - • Logical vs. physical qubit distinction. Syndrome measurement overhead.
  - • Code must account for T-gate costs, magic state distillation.

- **3. Transpilation and optimization matter more than ever**
  - • Circuit depth directly impacts error rates. Every gate counts.
  - • Qiskit 1.0 transpiler is 39x faster than 0.x — but optimization is the developer's job.

- **4. Multi-provider portability is real**
  - • Azure (IonQ, Quantinuum, PASQAL), Braket (IQM, Rigetti, QuEra). Write once, run anywhere.

- **5. Classical-quantum co-design is the norm**
  - • GPU pre-processing (CUDA-Q) + QPU execution + classical post-processing. Not just 'run a circuit'.

Recommended Resources & Further Reading

- **Updated resources for 2026**

- **Academic references**
  - • arXiv:2010.08053 — Quantum Software Development Lifecycle (foundational, 2020)
  - • arXiv:2502.17725 — Vehicle Routing Problem on quantum computers (2025)
  - • Nature Reviews Physics 2, 709-722 (2020) — Quantum programming languages

- **Learning platforms**
  - • IBM Quantum Learning: learning.quantum.ibm.com
  - • PennyLane Demos: pennylane.ai/qml/demos
  - • Microsoft Quantum: quantum.microsoft.com (Quantum Katas, Q# playground)
  - • Google Quantum AI: quantumai.google

- **Tools & SDKs**
  - • Qiskit 1.0: qiskit.org (Primitives API, Runtime)
  - • PennyLane: pennylane.ai (differentiable quantum computing)
  - • D-Wave Ocean: docs.dwavesys.com (annealing)
  - • Amazon Braket: aws.amazon.com/braket (multi-provider)
  - • NVIDIA CUDA-Q: developer.nvidia.com/cuda-q (GPU acceleration)

Open Discussion & Q&A

- What surprised you today?
- What would you explore further?
- How does this connect to your work or research?

FEEDBACK IS REALLY APPRECIATED!

training@quside.com

WE ARE HIRING!

www.quside.com/careers