

# A Reference Architecture for Quantum Computing as a Service

<sup>1</sup>Aakash Ahmad, <sup>2</sup>Ahmed B. Altamimi, <sup>3</sup>Jamal Aqib

<sup>1</sup>School of Computing and Communications. Lancaster University Leipzig, Germany

[a.ahmad13@lancaster.ac.uk](mailto:a.ahmad13@lancaster.ac.uk)

<sup>2</sup>Department of Computer Engineering, University of Ha'il, Saudi Arabia

[altamimi.a.uoh@gmail.com](mailto:altamimi.a.uoh@gmail.com)

<sup>3</sup>Department of Information and Computer Science, University of Ha'il, Saudi Arabia

[j.aqib@uoh.edu.sa](mailto:j.aqib@uoh.edu.sa)

## Abstract

Quantum computers (QCs) aim to disrupt the status-quo of computing – replacing traditional systems and platforms that are driven by digital circuits and modular software – with hardware and software that operates on the principle of quantum mechanics. QCs that rely on quantum mechanics can exploit quantum circuits (i.e., quantum bits for manipulating quantum gates) to achieve ‘quantum computational supremacy’ over traditional, i.e., digital computing systems. Despite being in a state of their infancy due to hardware limitations or lack of software ecosystem, QCs have started to demonstrate their data processing superiority in certain application areas including bio-inspired computing, cryptography, and tackling optimization problems. Currently, the issues that impede mass-scale adoption of quantum systems are rooted in the fact that building, maintaining, and/or programming QCs is a complex and radically distinct engineering paradigm when compared to challenges of classical computing and software engineering. Quantum service orientation is seen as a solution that synergises the research on service computing and quantum software engineering (QSE) to allow developers and users to build and utilise quantum software services based on pay-per-shot utility computing model. The pay-per-shot model represents a single execution of instruction on quantum processing unit and it allows vendors (e.g., Amazon Braket) to offer their QC platforms, simulators, software services etc. to enterprises and individuals who do not need to own or maintain quantum systems. Existing research lacks solutions in terms of empirically grounded processes, patterns, and guidelines to architect and implement quantum computing as a service. This research contributes by (i) developing a reference architecture for enabling quantum computing as a service, (ii) implementing microservices with the quantum-classic split pattern as an architectural use-case, and (iii) evaluating the reference architecture based on feedback by 22 practitioners. In the QSE context, the research focuses on unifying architectural methods and service-orientation patterns to promote reuse knowledge and best practices to tackle emerging and futuristic challenges of architecting and implementing Quantum Computing as a Service (QCaaS).

**Keywords:** Software Architecture - Quantum Software Engineering - Quantum Service Computing.

## 1 Introduction

The emergence of quantum computers (QCs) has started to gradually disrupt traditional computing technologies - systems driven by binary logic and digital circuits - with machines that rely on quantum circuits to achieve computational efficiency [1] [2]. QCs exploit the fundamental of quantum mechanics via programmable Quantum Bits (QuBits) that operationalise Quantum Gates (QuGates) to execute some calculations exponentially faster than any ‘traditional computer’ [3]. QCs represent a unification of hardware, i.e., quantum circuitry (QuBits mapped to QuGates) and software, i.e., quantum algorithms that manipulate the hardware, and network that can transmit and receive QuBit-encoded information [4]. In quantum-era computing, QCs are undergoing a continuous evolution from their inception to a gradual maturity. However, such systems have been successful in mimicking biological systems and chemical reactions, solving optimization problems, and empowering fundamental science

and existing technologies that are driven by quantum information processing [5]. To attain strategic benefits and developing commercial competencies associated with QCs, academic research [1] [2], industrial projects (e.g., Amazon Braket [6]), and technology funding consortiums (e.g., Quantum Flagship [7]) are competing in a so-called race towards building quantum economies. Global policymakers and state representatives at the most recent World Economic Forum (WEF) advocated for building quantum economies that currently represent public and private investment worth \$35.5 billion [8]. Despite the scientific benefits and commercial opportunities linked with QCs, a plethora of issues such as limited hardware, lack of software ecosystems, quantum noise, and scarcity of professional expertise in QSE domain hinders wide-scale adoption of quantum systems and technologies [9]. From users' perspective, enterprises and individuals lack access to quantum computing resources for tackling computationally hard tasks due to a multitude of challenges that may range from costs and economy of acquiring or maintaining QCs, quantum error rate, immature technology and/or lack of quantum software services [10]. From QSE perspective, engineers find themselves underprepared to tackle the complexities of quantum mechanics, handling QuBits/QuGates, knowledge of quantum programming, and workflows to develop software applications that can be executed on QC platforms [11, 12]. This leads to a situation that is referred to as the quantum divide, representing a strategic and computational disparity between entities or states who have access and the ones that lack access to quantum systems and technologies [10]. To minimise this divide, initiatives across the world such as the Quantum Flagship [7] and National Quantum Initiative [13] are focused on supporting the development of software ecosystems, networking technologies, and human expertise for the alleged quantum leap in computing [2].

**Service-orientation for QCs:** Service-oriented systems are viewed as the enablers of utility computing model, allowing pay-per-usage of software applications and hardware resources that are made available as a computing services, to be used by individuals and enterprises [14]. The providers of utility computing (i.e., service vendors) offer a variety of services to their customers (i.e., service users) that range from data storage, video streaming, and entertainment, to resource-sharing applications, representing a multi-billion dollar industry in service economies [15]. Central to the success of service-orientation is the concept of as-a-Service (aaS) or anything-as-a-Service (\*aaS) model that provides any-time, any-where distributed access to *infrastructures* (e.g., Microsoft Azure), *platforms* (e.g., Google App Engine) and *software* (e.g., Cisco WebEx) as a service [16]. Attuned to the practices of service-orientation is the concept of quantum service computing that allows quantum vendors to provide and quantum users to request quantum hardware and software resources available via network [11] To breach the quantum divide [10], quantum service-orientation can enable service requesters (QC users) to access resources offered by service providers (QC vendors). Figure 1 conceptualises a quantum service computing model where the users can utilise a multitude of resources such as quantum processing units, simulators, storage, algorithms, and software applications to tackle computationally challenging tasks via QCs. On the flip side, QC vendors see quantum service computing as an opportunistic business model to generate revenue streams by offering pay-per-shot resources or crowd-sourced testing of their under-developed quantum platforms [6, 12]. In QC context, a shot represents a single execution of quantum instruction on a quantum processor unit. Considering quantum service-orientation, as envisioned in Figure 1, there is a need for research and development, rooted in empirically grounded processes, patterns, and tools to architect and implement Quantum Computing as a Service (QCaaS).

**Research context and objectives:** Quantum software engineering is regarded as a recently emerged genre of software engineering (SE) that aims to apply the principles, processes, and practices of SE to systemise the development of quantum software systems and services [2] [17]. QSE empowers the role of software designers and developers who can exploit processes that support structured development, architectural models for design visualisation, and patterns as best practices to engineer software that can be executed on QCs [18]. QSE approaches can help developers to abstract the complexities of quantum mechanics such as mapping operations of QuBits and QuGates to software components that can be transformed to modules of quantum source code via model-driven engineering [19] The objective of this research is *'to support pattern-based architecting - enabling reuse of best practices and exploiting architectural modeling - to engineer software services in the context of QSE and for enabling QCaaS'*. By relying on QSE, we adopted a stepwise approach to architect and implement (proof-of-the-concept) QCaaS for quantum information processing. The proposed research aims to synergise the principles of QSE and practices of service-oriented computing to (i) develop a reference architecture that acts as a blue-print for (ii) implementing a prototype of quantum computing as a service. We engaged a total

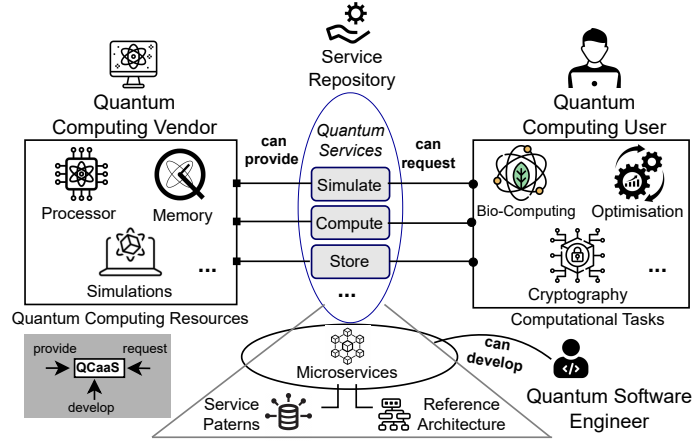


Figure 1: An Overview of Quantum Service Computing.

of 22 QSE professionals, from 12 countries across 6 continents, experienced in working with various QC platforms (e.g., Amazon Braket, Quantum Azure) to evaluate the suitability and usability of the reference architecture [20]. The reference architecture can enable engineers to abstract complexities of quantum source code into architectural components, apply reuse knowledge via quantum software patterns, and adopt best practices such as microservices architecture style to develop QCaaS solutions [12]. We developed a QCaaS solution by using microservices architecture, applied service orchestration and quantum-classic split patterns, and executed quantum service-orientation on the Microsoft Azure platform. Primary contributions of this research include:

- An empirically grounded reference architecture, rooted in the systematic mapping of published research [11] and IBM service development lifecycle [21], to provide a blue-print and point of reference to architect software-intensive systems and services for QCaaS.
- A proof-of-the-concept that demonstrates architecture-centric and pattern-driven implementation of quantum software services that can be executed on a quantum computing platform.
- Practitioners' evaluation of the reference architecture that provides recommendations and guidelines to design and develop solutions for QCaaS.

The study represents a pioneering effort in architecting QCaaS solutions and it requires empiricism, diverse usecase, and further experimentation as part of future research. This study can help academic researchers to understand the role of reference architectures, patterns, micro-servicing etc. and help them formulate new hypotheses for investigating emerging and futuristic challenges of QSE in the context of QCaaS. Practitioners can explore and extend the reference architecture (system blueprint) and reuse knowledge (patterns) that can be adopted to develop solutions for QCaaS.

Rest of this paper is organised as follows. Section 2 discusses background details on quantum systems and quantum service computing. Section 3 presents the research method to conduct this study. Section 4 details the design and interpretation of the reference architecture for quantum computing as a service. Section 5 presents a proof-of-the-concept implementation of the reference architecture. Section 6 discusses evaluation of the reference architecture. Section 7 presents the related work to rationalise the scope and contributions of the proposed research. Section 8 details threats to the validity of this research. Section 9 presents conclusions and dimensions of future work.

## 2 Context: Service Orientation for Quantum Computing

This section presents background on quantum computing (Section 2.1) and service-orientation for quantum systems (Section 2.2) to contextualise service-orientation for quantum computing. We use the illustrations in Figure 2 that shows building blocks and conceptualisation of quantum computing as a service. The concepts and terminologies introduced in this section are used throughout the paper.

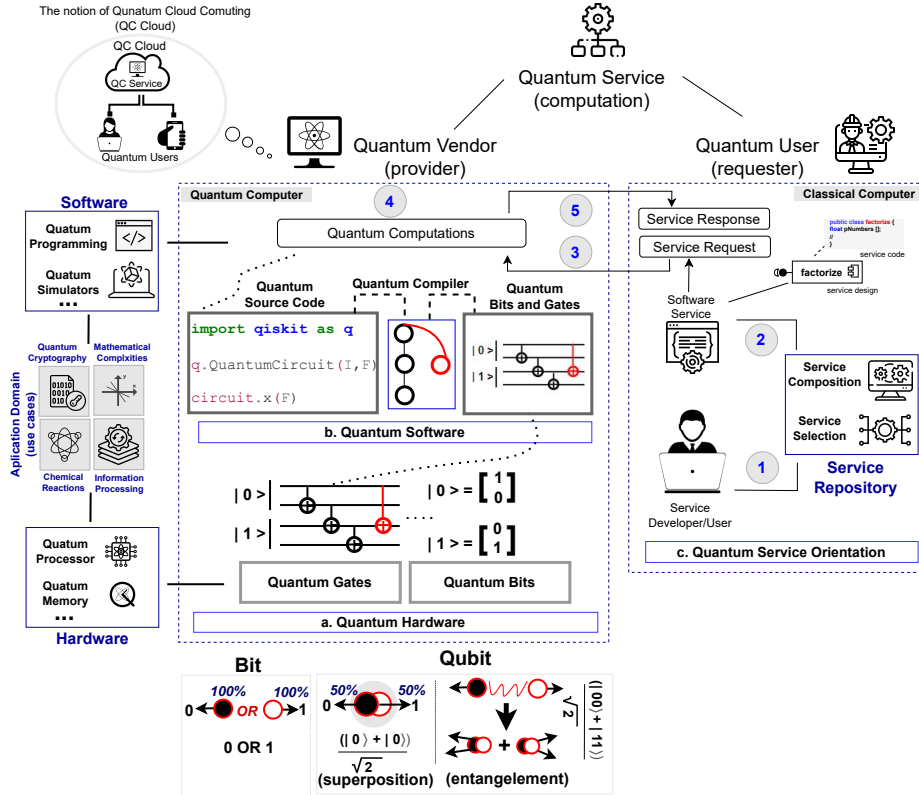


Figure 2: A General View of the Quantum Service Computing Systems.

## 2.1 Quantum Computing Systems

To gain strategic advantages in the quantum race, i.e., attaining technical and commercial benefits of quantum computing, research and development efforts are focused on engineering both the hardware and software systems that can be unified into a practical quantum computer [4] [1]. Application domains or practical use cases of QC systems such as quantum cryptography or bio-inspired computing can exploit quantum hardware resources by means of quantum software systems and applications that can manipulate the hardware [2]. We discuss the quantum computing system from both the hardware and software perspective as in Figure 2 a). Fundamental to achieving quantum computations are Quantum Bits (QuBits) that represent the basic unit of quantum information processing by manipulating Quantum Gates (QuGates) [3]. Traditional Binary Digits (Bits) in classical systems (i.e., digital computers) are represented as  $[1, 0]$  where 1 represents the computation state as ON and 0 represents the state as OFF to manipulate binary gates in digital circuits. In comparison, a QuBit represents a two-state quantum computer expressed as  $|0\rangle$  and  $|1\rangle$ . Specifically, the state of a single QuBit can be expressed as  $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$  and  $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ . When compared to a Bit, quantum superposition allows a QuBit to attain a liner combination of both states:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad (1)$$

In Figure 2 a), we distinguish between a Bit and Qubit. Unlike the Bit that can manipulate an electronic circuit as On and Off, a Qubit uses the properties of quantum mechanics to be a 1 or 0 or attain any value in between. Specifically, a Bit can take a value of '0' or '1' representing as either 'Off:0' or 'On:1' with 100% probability (left, Figure 2). In comparison, a Qubit can be in a state of  $|0\rangle$  or  $|1\rangle$  or in a superposition state with 50%  $|0\rangle$  and 50%  $|1\rangle$  (middle). In addition, two Qubits can be entangled (right), and entangled qubits are linked in a way that observing (i.e., measuring) one of the QuBit, can reveal the state of other QuBit. There is an abundance of literature and use cases that discuss theoretical aspects of quantum physics and its application to quantum systems, extended

details of QuBits and QuGates to develop and operate the QC systems are reported in studies like [22] and [3]. To utilise the QC resources such as quantum processor and memory, there is a need for control software that can program QuBits to manage QuGates of a QC system. Quantum software systems rely on quantum source code compilers that allow quantum algorithm designers and code developers to write, build, and execute software for quantum computers [23]. For example, a programmer can use a quantum programming language such as Q# (by Microsoft) or Qiskit (by Google) to use quantum compilers for enabling programable quantum computations [24]. By developing software systems that can manage and control quantum hardware, a number of applications such as quantum cryptography, bio-inspired computing, and quantum information processing can benefit from quantum supremacy in computing.

## 2.2 Service-Oriented Architecture for Quantum Computing

Service computing follows the Service Oriented Architecture (SOA) style that allows service users to discover and utilise a multitude of available software services that encapsulate computing resources and applications offered by service providers [14]. Figure 2 b) shows SOA-styled quantum servicing where a QC user (i.e., service requester) can utilise the QC resources offered by quantum vendors (i.e., service provider) by means of quantum services [11]. Despite the computational superiority of QC systems in tackling certain classes of complex problems, when compared to traditional computers, quantum systems are in a state of infancy due to continuous evolution of hardware, immature technology, and limitations of quantum algorithms. In most cases, QC systems of today are not capable of executing quantum algorithms wrapped with a large amount of data, inputs, and outputs [9]. More specifically, contextualising the computation illustrations in Figure 2, large volumes of data in quantum algorithms require more QuBits and complex QuGates that result in deep quantum circuiting and consequently increased errors referred to as noisy intermediate-scale quantum (NISQ) [25]. To address issues like NISQ and to make quantum computers more practical in handling a significant amount of processing, the classic-quantum split pattern splits quantum software (having algorithms and data) into a classical part and a quantum part [26]. In principle, the classic-quantum split slices the overall quantum software or application into classical modules (pre/post-processing) and quantum modules (quantum computation) that result in hybrid applications [26]. One of the prime examples of the classic-quantum split patterns is Shor’s algorithm which involves quantum computations for finding the prime factors of an integer with its application in computer security and cryptography [27]. The algorithm is composed of two parts. The first part of the algorithm turns the factoring problem into the problem of finding the period of a function and may be implemented classically. The second part finds the period using the quantum fourier transformation and is responsible for the quantum speedup. The quantum service can create (1) factors of numbers and provide this an input, (2) the factorization is done on a quantum computer and (3) results are returned for further processing on a local (classical) computer [26].

Quantum computing vendors such as Amazon, IBM, and Google have started to offer their QC systems and infrastructures that can be utilised by individuals and organisations by means of quantum service computing [28]. For example, Amazon Braket as a QC platform relies on Amazon Web Services (AWS) to support the execution of quantum applications [6]. In addition to industrial development and commercialisation, academic research is also striving for solutions that can offer algorithms, hardware, and mathematical problems as services [11]. To harness QC as utility computing, there is a need to tailor existing principles and methods of service-orientation or develop new architectures and frameworks, empirically grounded processes to synergise QC and SOA research in the context QCaaS. The QCaaS also provides foundations for building a quantum computing cloud (QC cloud), led by technology giants like IBM to offer computation resources to the end-users based on cloud computing model [29].

## 3 Research Method

This section details the research method to conduct this study as illustrated in Figure 3. Each of the four steps of the research method, as visualised in Figure 3, are elaborated below in Section 3.1 - Section 3.4.

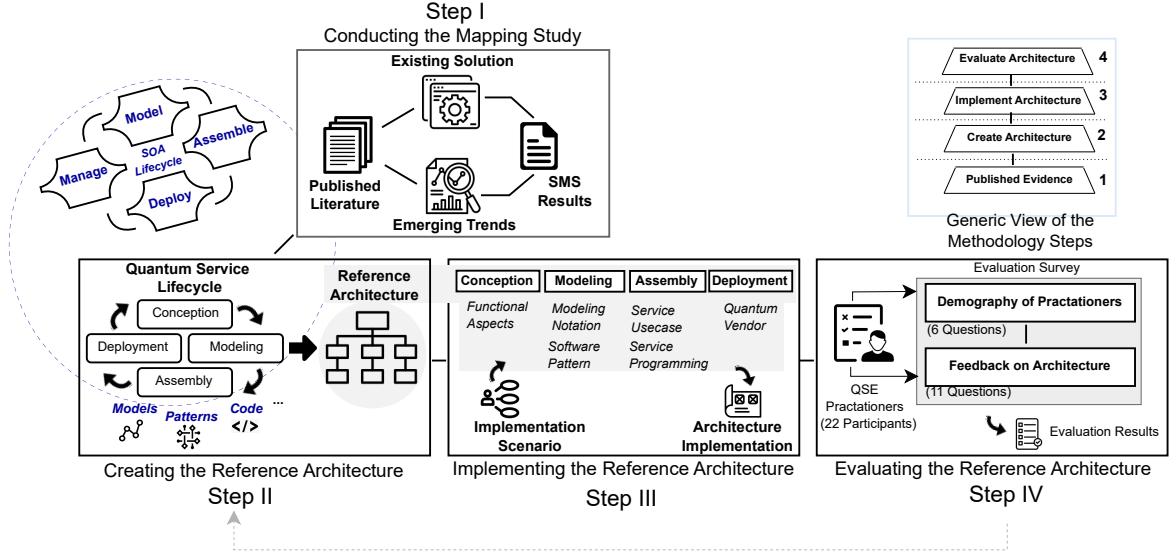


Figure 3: . Process and Steps of the Research Method

### 3.1 Step I - Conducting the Mapping Study

As per Figure 3, the initial step of the research method focused on conducting a systematic mapping study to analyse the existing literature, emerging trends, and prominent challenges of architecting and implementing quantum computing as a service. Systematic mapping study (SMS or mapping study for short) as an approach is grounded in evidence-based software engineering method for reliable and replicable identification, analysis and synthesis of data or facts on a topic under investigation. We followed the guidelines in [30] to conduct SMS of existing published research that enables or enhances architecting and development of quantum computing as a service. The results of SMS on QCaaS along with the process to conduct and document the study are detailed in [11]. We only highlight the core findings of the SMS in Table 1 that synthesises the available evidence, derived from published research and proposed solutions, to provide the basis for creating reference architecture for QCaaS (in Section 3.2). A synoptic view of the SMS results, reflected in Table 1, helped us identify four phases and six activities to support the conception, modelling, assembly, and deployment of QCaaS.

### 3.2 Step II - Creating the Reference Architecture

From software engineering and architecting perspective, a reference architecture denotes a blue-print as a collection of documents, notations, or design artifacts etc. to provide a recommended structure to instantiate solutions in terms of software-intensive systems, services, and applications. Empiricism is central to design, develop, and validate the reference architectures. One specific example of an empirically grounded reference architecture derived from five industrial projects is presented in [31] which suggests three main phases in terms of architectural analysis, architectural synthesis, and architectural evaluation to develop software systems. In the context of quantum software engineering, the research in [18] extends the generic reference architecture from [31] to propose a process and reference architecture for quantum software. Based on the reference architecture for general purpose software [31] and architecture process for quantum software applications [18], we developed a reference architecture based on four phases from SOA lifecycle [21]. Table 1 acts as a structured catalogue to organise four phases and six activities for architecting QCaaS.



Table 1: Summary of the Core Findings form SMS [11]

Research Type	Functional Aspects	Modeling Notation	Software Pattern	Service Usecase	Programming Model	Quantum Vendor	
Solution	Experimental	Unified Modeling Language	API Gateway	Optimisation	Python	Amazon Braket	
Opinion	Service Development	Business Process Modeling Notation	Layered Architecture	Process Automation	Flask	IBM Quantum	
Validation	Number Generator	Graph Models	Classic-Quantum Split	Mathematics	Q-Sharp	IBM Q	
	Data Search	Ontologies	Service Wrapping	Simulation	Cirq	Rigetti	
	Prime Factorisation		Repository Pattern		Qiskit		
	Bio-inspired Computing				Java		
Study ID & Research Type	Conception		Modeling		Assembly		Deployment
	Functional Aspects	Modelling Notation	Software Pattern	Service Use case	Service Programming	Quantum Platform/ Vendor	
 [S1]	Quantum Service Delivery	Deployment Diagram	API Gateway	Optimal Service Provider	Python, Flask	Amazon Braket	
 [S2]	Enterprise Services Development	Business Process	Layered Architecture	Process Automation	No Evidence	No Evidence	
 [S3]	Quantum Random Number Generation Quantum Search Algo	Class, Sequence Diagram	Classic- Quantum Split	Mathematics	Q# Q sharp	IBM Q IBM Quantum	
 [S4]	Integer Factorisation	Deployment Diagram	Solution	Mathematics	Python	Amazon Braket	
 [S5]	Experimental Quantum Service Computing)	No Evidence	Service Wrapping	Optimisation	Python, Flask	Amazon Braket	
 [S6]	Experimental Services Algorithm	Sequence Diagrams	API Gateway	Algorithm as a Service	Python	Rigetti	
 [S7]	Integer Factorisation	Directed Graph	API Gateway	Optimisation	No Evidence	Amazon Braket	
 [S8]	Bio-inspired Computing	Ontologies	Repository Pattern	Simulation	No Evidence	No Evidence	
 [S9]	Integer Factorisation	No Evidence	Service Wrapping	No Evidence	Python	Amazon Braket	

### 3.3 Step III - Implementing the Reference Architecture

The third step of the research method corresponds to implementing a QCaaS solution based on the reference architecture (Step II), as in Figure 3. We refer to the implementation as developing a prototype solution, representing a QCaaS proof-of-the-concept based on the architecture.

A use case of prime factorisation is included for the implementation that exemplifies how each of the four phases and six activities adopted from Table 1 are utilised in the reference architecture as in Figure 5. Details for reference architecture implementation are presented in Section 5.

### 3.4 Step IV - Evaluating the Reference Architecture

Finally, the last step of the research method involves evaluating the reference architecture based on feedback from quantum software engineering practitioners. As per the ISO/IEC 25010 model for evaluating the quality of software-intensive systems artifacts, we evaluated the functional suitability and usability of reference architecture [20]. We engaged a total of 22 QSE practitioners and collected their feedback based on 17-point criteria for architecture evaluation. Details of the architecture evaluation are presented in Section 6.

## 4 Creating the Reference Architecture for QCaaS

Reference architectures provide a point of reference to structure a system by identifying and representing architectural components and connectors, applying reuse knowledge and best practices, and facilitating communications between domain professionals (a.k.a. system stakeholders) to design concrete architectures [32]. We present a layered view of the reference architecture in terms of three distinct layers namely *service development*, *service deployment*, and *service split* layers, illustrated in Figure 4 and detailed below. Figure 4 a) shows an abstract view to conceptualise the structural composition of the architecture, whereas Figure 4 b) uses the abstract view to present a fine-grained and concrete view of the architecture in terms of i) architectural layers, ii) phases and activities (from Table 1) encapsulated inside each layer, iii) human roles, and iv) service artifacts. A fine-granular representation of the reference architecture based on individual layers and elements encapsulated inside each layer is detailed below based on the illustrations in Figure 4.

### 4.1 Architecture Layers

Architectural layering allows a separation of functional concerns which means architectural elements that support similar functionality can be structured in the same layer. For example, Figure 4 shows that conception, modelling, and assembly of quantum services can be unified into a layer named service development layer, service execution/hosting is presented as part of deployment layer, and the split between quantum and classic execution is presented as part of the service split layer. In addition to supporting the structuring of the system, a reference architecture provides a template that helps to design a (software) solution for a particular domain, quantum computing in this case. Reference architectures also provides a common architecting vocabulary such as functional requirements and their representation as architectural components and connectors. For example, the service development layer has one of the phases named *Quantum Service Conception* that encapsulates two activities namely *Functional Specifications* and *Quality Attributes*. These two activities allow a human role in the architecture such as *Service Developer* who can specify the required functionality and desired quality of the service. The phase produces a *Quantum Significant Requirements* (QSRs) as an artifact that provides the foundation for service design. In short, architectural structuring as layers contains phases that encapsulate multiple activities, enabling a human role to produce the service artifact for an incremental conception, modelling, assembly, and deployment of QCaaS.

### 4.2 Phases and Activities

The phases and activities summarised in Table 1 are adopted from the SMS [11] and organised based on IBM service-oriented architecture (SOA) lifecycle [21] to develop the reference architecture. As per the SOA lifecycle for service engineering and development, each phase tells *what needs to be done?* while an activity or collection of activities demonstrates *how it is to be done?* For example, in Table 1,



the phase referred to as modeling helps to focus on how to model or represent the quantum services for their implementation, while the activities named (i) **modeling notation** show class and component diagrams as UML-based modelling notation to enable service modelling and (ii) **patterns** indicate best practices to model the QC services.

To architect QCaaS, we divided the **Model** activity from SOA life cycle into two activities namely *Conception* and *Modeling* to distinguish between functional needs (conception) and representation (modeling) of quantum service design. The distinction allows a fine-grained representation of the reference architecture that delineates (i) the conception of functional needs and (ii) the model that architecturally represents the functional needs for their implementation. Model represents the conception as the design specification of functional needs for quantum services. For example, the initial row of Table 1 highlights that the conception, i.e., required functionality that enables the delivery of quantum software is modeled, i.e., architecturally represented using a UML deployment diagram and applying the API Gateway pattern. The reference architecture in Figure 4 do not have the **Manage** phase from SOA life cycle as we could not find any evidence in the literature that supports identity, compliance, and business metrics management of quantum services. The phases, their underlying activities, and the data in Table 1 is used to create the reference architecture shown in Figure 4.

### 4.3 Human Roles

Human roles represent human knowledge, expertise, or activities as part of the architecting that enables development or utilisation of the quantum services [33]. There are two types of human roles in the reference architecture, referred to as service developers and service users. Service users can be individual(s) or a group of users (part of a team or an organisation) that requires quantum computation. In contrast, service developers can include a multitude of roles that include but are not limited to quantum service developers, quantum algorithm designers, and quantum domain engineers. A recent study on architecting quantum software highlights the need for quantum-specific expertise such as quantum software architects who can map the operations of QuBits to architectural components, and quantum code managers who can simulate and analyse the flow of quantum information processing [18]. A specific human role such as quantum domain engineer can analyse quantum-specific attributes like mapping between QuGates and their corresponding QuBit representation [17]. Quantum domain engineer focuses on design and fabrication activities to improve QPU performance by optimising quantum algorithms and programming languages. Quantum domain engineers, guide quantum hardware and software teams to realise quantum significant requirements that need to be implemented as quantum algorithms for execution on quantum computing platforms, as in Figure 4 b).

### 4.4 Service Artifacts

Service artifacts refer to any (tangible) outcome as a document, design model, or source code script that enable the development and delivery of the quantum service. As in Figure 4, each phase in the architecture layer produces an artifact as an outcome of the specific phase. There are four artifacts referred to as Quantum Significant Requirements, Quantum Service Design, Quantum Service Implementation and Quantum Service Deployment. For example, the quantum service conception phase produces QSR having function and quality attributes as an artifact that supports modelling and pattern-based design of quantum service.

## 5 Implementating the Reference Architecture

Reference architecture-based implementation of QCaaS is highlighted in Figure 5 that is incremental and driven by four phases of the service lifecycle from Figure 4 that is based on the evidence from published research documented in the SMS and its core findings synthesised in Table 1. The implementation does not refer to a comprehensive or full-scale development of the solution, it provides a proof-of-the-concept in terms of executable specifications for quantum computing as a service. The proof-of-the-concept represents a use-case of the reference architecture that provides the basis for further implementation and validation of the architecture. The use case as a specific instance of implementation is elaborated later in this section while discussing the service assembly.

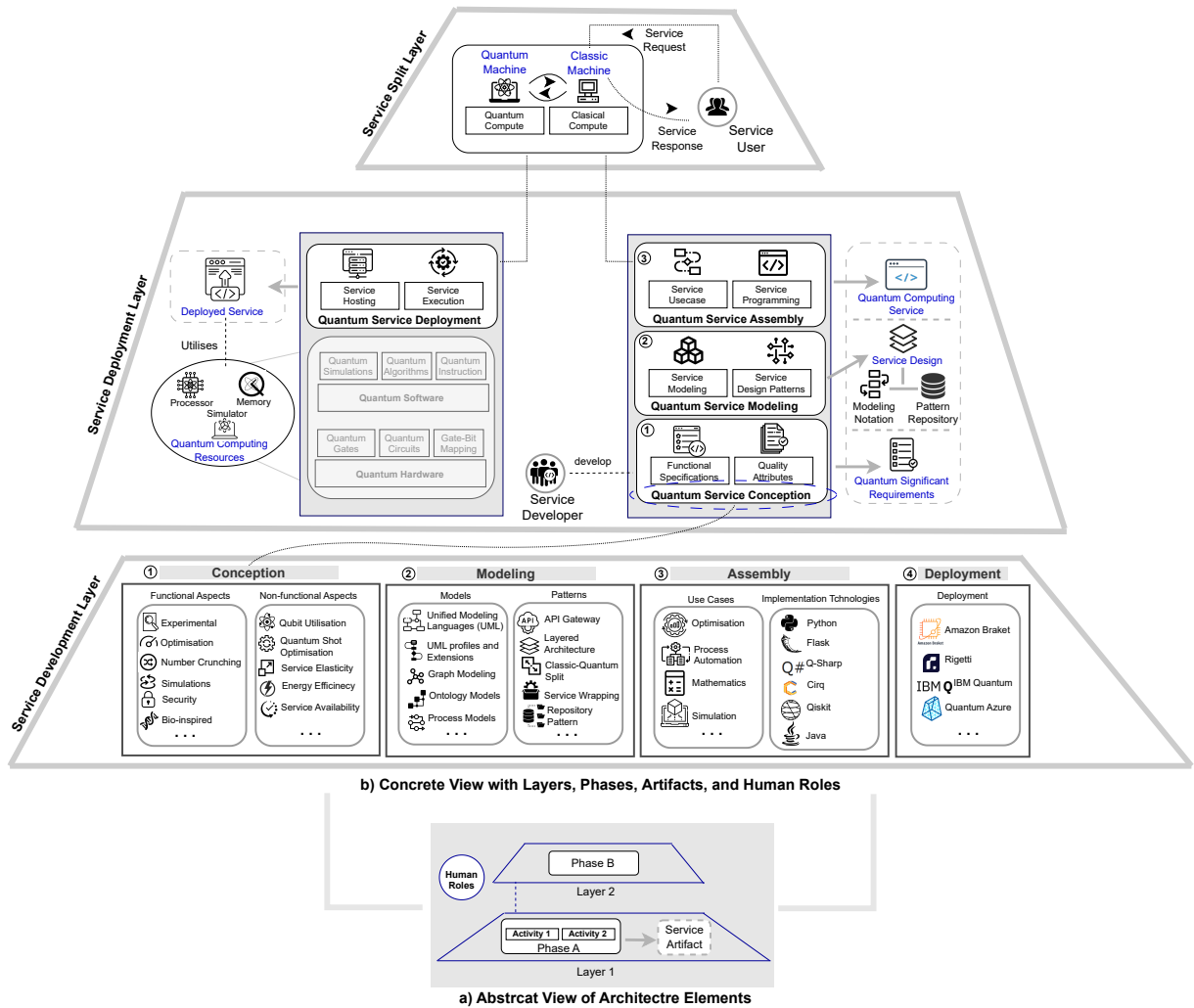


Figure 4: Overview of the Reference Architecture for QCaaS.

## 5.1 Classical vs Quantum Microservices

Microservices help software designers and developers to structure software applications, via the application of microservice architecture style, as loosely coupled and deployable modules of code that enable computation and data storage [12]. Microservices systems enable enterprises to adopt the service computing model by migrating or modernising their monoliths to a single application that consists of multiple small services. These microservices operate independently, with each service running in its own process and communicating, i.e., enabling message passing among each other, using lightweight mechanisms like HTTP resource APIs [21]. Industry-leading service providers such as Netflix and Amazon represent typical examples of how microservices architecture helps businesses to deliver their core business functionality (e.g., video streaming, online shopping) with increased scalability that efficiently serves millions of customers across the globe. Quantum microservicing is a term that refers to microservices that are deployed and executed on quantum computing platforms. For example, the solution [34] utilises Qiskit an open-source quantum software development kit to develop and deliver quantum computation as a collection of microservices that can be executed on IBM Quantum platform. From an implementation perspective, which primarily involves writing and executing source code, both classical and quantum microservice are similar because they implement an algorithm or contain a module of source code that can be executed on QCs [11]. However, from operational and deployment point of view, which focuses on managing the execution of service on quantum platforms, QSRs need to be fulfilled as implemented functionality and desired quality of the microservice. Microservices that fulfill QSRs are referred to as quantum-enabled microservices or simply quantum microservices. For example, QSRs may require quantum-classic split of a quantum algorithm, efficient utilisation of QuBits, and quantum error minimisation in quantum task execution. For example, the study [12] suggests that a quantum algorithm can be wrapped in a microservice using the service wrapping pattern that allows classical microservices to be executed on a QC platform.

## 5.2 Service Conception

Figure 5 provides conception of quantum services by outlining the functional aspects and quality attributes, which collectively refer to as the QSRs, as specified below. The listings below highlight the functional specification of the quantum services to generate the prime factors of a given integer. To provide this functionality, the computations need to be split between a classical machine and a quantum machine. There is also a need to assess the correctness of the implementation and efficiency of the solution for prime factorisation. The listings below as the functional aspects and quality attributes two quality attributes concerning an efficient utilisation of the QC resources in terms of the utilisation of the QuBits. The quality attributes complement the functional specification with desired quality of the solution.

Develop a solution that acquires an input integer  $N$  and outputs its prime factors  $F$ .

- Split classical and quantum computations.
- Validate correctness and efficiency.

**QuBit utilisation** The solution should efficiently utilize the available qubits by minimizing the number of qubits required for factoring integers of a given size.

For example, Figure 5 indicates two architecting activities as part of service conception to specify the functional and quality aspects of prime factorisation. Functional and quality aspects are specified as quantum significant requirements that represents an architectural artifact of the conception phase. The service architect/developers can rely on the QSRs to utilise available modelling notations (e.g., UML, ADL) and apply any patterns (e.g., service orchestrator) to create a service design that acts as an artifact for assembling the microservices.

## 5.3 Service Modeling

The QSRs as part of the conception act as the foundation to create a service design model that is reflective of the functional aspects. The quantum service modeling phase relies on two activities (i)

creating the model, i.e., visual representation of the service and (ii) applying the pattern, i.e., the design decision to model the service.

*Modeling the quantum service* As per Figure 5, we have used the UML as the modelling notation to create the service model based on UML component diagram and UML sequence diagram. To create quantum-enabled models for QSE, the quantum UML profile extends the traditional UML diagrams to support structural and behavioral modeling of quantum software [35]. Specifically, the UML component diagram in Figure 5 shows a structural view of the quantum services and their interconnection. Following the notations from UML profile [35], from a service modeling perspective each service is represented as an individual component. The component provides a computational service (e.g., generate a random number) and communicates with another component (s) using a connector. For example, the service named `GetGCD` interconnects with another service named `Controller` to generate the Greatest Common Divisor (GCD) of a randomly generated number as part of the algorithmic implementation. In contrast to a structural view of the quantum service model using a component diagram, the UML sequence diagram reflects the behavior of the services in terms of message passing among the services to enable service communication. For example, in the sequence diagram, the controller service `C: Controller` passes a message `Generate(N)` to the number generator service `N: NumGenerator` that generates a random number `R` and returns it to the controller service. A number of other UML diagrams can also be used such as the UML use case diagram to represent the QSRs. For demonstration purposes, we have used the UML component and sequence diagrams to exemplify service modeling in terms of structural composition and behavioral representation of the services to be assembled.

*Pattern-based Modeling:* Service design and development patterns provide reuse knowledge and best practices to engineer service-oriented solutions [21]. Patterns can be particularly useful, as concentrated wisdom of experienced software designers and developers, that can guide novice engineers (e.g., quantum algorithm designers, quantum code developers) to rely on existing knowledge and practices to develop quantum software effectively and efficiently [26]. In Figure 5, we have applied two patterns namely the *orchestrator* and *classic-quantum split* pattern. The orchestrator pattern is a classic SOA pattern that helps to orchestrate the execution of a number of services to complete a service-driven task [36]. For example, the application of the orchestrator pattern helped to orchestrate the factorisation of a randomly generated prime number via `NumGenerator` and `Factorise` services. The classic-quantum split pattern helps to split the functionality between quantum and classic computing, also referred to as hybrid quantum computing [37]. In the context of Figure 5, the `Controller` service orchestrates the functionality between classical and quantum microservices. For example, the services named `NumGenerator` and `GetGCD` can be executed on a classical machine, whereas the services `QuantumModularExponentiation`, `QuantumInverseQFT`, and `Factorise` are executed on the quantum machine. As part of the reference architecture, the modeling phase exploits modeling notation and applies a pattern to create a service design, i.e., a 'visual model' that can be assembled into executable quantum microservices.

## 5.4 Service Assembly

Service assembly refers to assembling a service, i.e., identifying a use-case and programming the service that implements the usecase as shown in Figure 5 and detailed below.

*Service Usecase:* Implementation details are elaborated based on Figure 5 that demonstrates the conception, modelling, assembly, and deployment of software services that implement the Shor's. Shor's algorithm is a quantum computing algorithm that is used for factoring integers in a polynomial time [27]. Shor's algorithm has a number of use cases such as prime factorisation, cryptography (e.g., breaking the RSA scheme), and finding the period of a function. In the scope of this work, we only focus on developing microservices, executed on a QC platform that implement Shor's algorithm for prime factorisation. We provide an illustrative case in Figure 5 that exemplifies the architecture-based development of the services (from Figure 4) to implement Shor's algorithm. Specifically, Figure 4 as a reference architecture highlights *what needs to be done?* by providing a blue-print to architect quantum service-orientation. Figure 5 implements the reference architecture to highlight *how it is to be done?* with an illustrative example of implementing quantum service-orientation.

### Conception

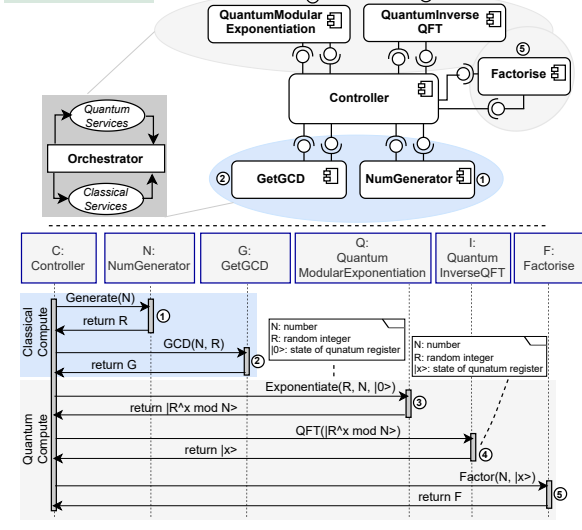
#### Functional Aspects

- Develop a solution that acquires an input integer  $N$  and outputs its prime factors  $F$ .
- Split: classical and quantum computer.
- Validate: correctness and efficiency. (Experimental Service)

#### Quality Aspects

- The solution should be computationally efficient specifically for large integers (Computation Efficiency)
- The solution should efficiently utilize the available qubits by minimizing the number of qubits required for factoring integers of a given size (Qubit Utilisation)

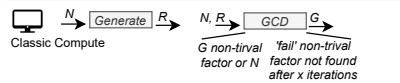
### Modeling



### Assembly

#### Algo-1: Preprocessing for Prime Factorisation (classic compute)

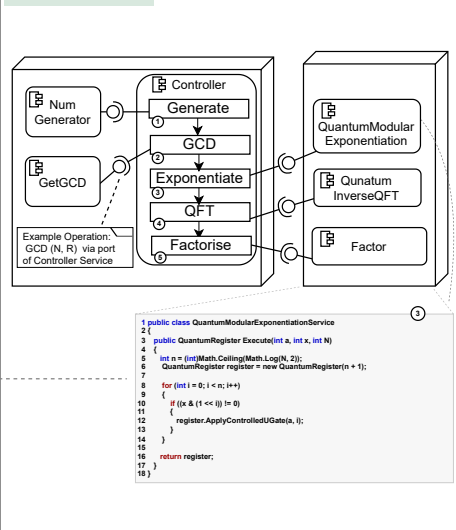
- Require:  $N$ : positive integer  
 Ensure:  $F$ : non-trivial factor of  $N$  or  $\epsilon$ : fail
- Pick Random Integer  $1 \leq r \leq x$
  - Compute  $G \leftarrow \text{GCD}(R, N)$
  - IF:  $G \neq 1$ ,  $G$  is non-trivial factor of  $N$ . ELSE: step 3
  - Compute  $r \leftarrow (R^r \equiv 1) \% N$
  - IF:  $(r \% 2 = 0) \& (R^{r/2} + 1) \neq 0 \% N$ . ELSE: step 1
  - Return  $\epsilon$  if a non-trivial factor has not been found after  $x$  iterations



```

1 using System.ServiceModel;
2 * Controller Service */
3
4 namespace PrimeFactorisationController
5 {
6     public interface IPrimeFactorisationController
7     {
8         [OperationContract] // random number generation
9         public int NumGenerator();
10
11     [OperationContract] // compute the GCD
12     public int GetGCD(int N, int R);
13
14     [OperationContract] // Quantum Modular Exponentiation
15     public QuantumRegister QuantumModularExponentiation(int N, int R, QuantumRegister Reg);
16
17     [OperationContract] // Quantum Inverse QFT
18     public QuantumRegister QuantumInverseQFT(QuantumRegister Reg);
19
20     [OperationContract] // Compute Factors
21     public Factorise (int N, QuantumRegister Reg);
22 }
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
    
```

### Deployment



```

1 public class QuantumModularExponentiationService
2 {
3     QuantumRegister Execute(int a, int x, int N)
4 {
5     int n = (int)(Math.Ceiling(Math.Log(N, 2)));
6     QuantumRegister register = new QuantumRegister(n + 1);
7     for (int i = 0; i <= x; i++)
8     {
9         if ((i & 1) <= 0)
10            register.ApplyControlledUGate(a, i);
11    }
12    return register;
13 }
14 }
15 }
16 }
17 }
18 }
19 }
20 }
21 }
22 }
23 }
24 }
25 }
26 }
27 }
28 }
29 }
30 }
31 }
32 }
33 }
34 }
35 }
36 }
37 }
38 }
39 }
40 }
41 }
42 }
43 }
44 }
45 }
46 }
47 }
48 }
49 }
50 }
51 }
52 }
53 }
54 }
55 }
56 }
57 }
58 }
59 }
60 }
61 }
62 }
63 }
64 }
65 }
66 }
67 }
68 }
69 }
70 }
71 }
72 }
73 }
74 }
75 }
76 }
77 }
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }
    
```

Figure 5: Architecture Based Implementation of the QCaaS.

*Service Programming:* In order to implement the use-case service needs to be programmed for its execution. This means that design specifications from the modeling phase need to be translated into executable specifications as part of service assembly. Figure 5, shows the algorithmic design and its implementation as part of service assembly. The algorithm shows a partial assembly of services by exemplifying the classical services that generate a random number and compute the GCD for quantum computation of the Shor’s algorithm. The UML sequence diagram is used as a model to assemble the service via an algorithm and its implementation in a given programming language. A synoptic view of the code snippet (written in C#) is shown that highlights the source code skeleton for a controller service PrimeFactorisationController that orchestrates the classical and quantum services.

## 5.5 Deployment

As the last phase in the reference architecture, deployment consists of two main activities namely service hosting and service execution. While execution and hosting primarily depend on the QC platform on which the quantum services are deployed.

*Service Hosting:* The hosting of the assembled service is presented as a UML deployment diagram as in Figure 5. UML deployment diagram show two deployment nodes (hosting machines). The classical compute node hosts three services NumGenerator, GetGCD, and Controller, whereas the quantum compute node hosts three services namely QunatumModularExponentiation, QunatumInverseQFT, and Factor.

*Service Execution* depends on the platform on which the quantum service has been deployed. We have used the Microsoft Qunatum Azure platform to deploy the qunatum microservices for prime factorisation. Figure 6 shows the circuit diagram for utilisation of the QuBits for quantum service execution. The notation  $q[0] \dots q[3]$  shows the allocation of Qubits and  $c[0] \dots c[3]$  represent classical bits. All the Qubits are initiliased such as  $U^{x_{modN}}_{q[0]}$  with a unitary operator for modular exponentiation acting on the first QuBit. The measurement results of each qubit are stored in the classical bits  $c[0] \dots c[3]$ . These classical bits are then used for classical post-processing, such as applying the continued fraction algorithm to extract the factors. This is expressed as: Circuit = QuantumCircuit(QuantumRegister, ClassicalRegister) to measure  $q[0] \dots c[0]$ .

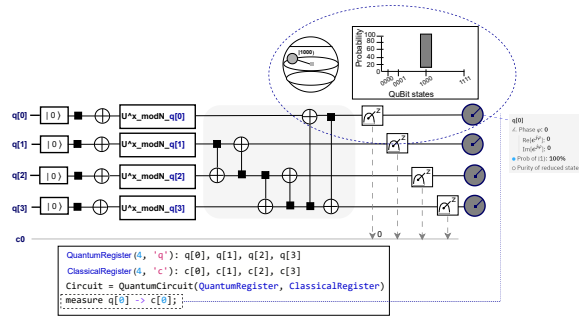


Figure 6: Circuit Diagram for QuBit Utilisation.

## 6 Practitioners’ Evaluation of the Reference Architecture

This section details evaluation of the reference architecture based on practitioners’ perspectives to assess the suitability and usability of the architecture (per ISO-IEC-25010 quality model [20] in the context of software services for QCs. First, we highlight the architecture evaluation methods and distinguish between the evaluation of a concrete and a reference architecture in Section 6.1. We then introduce the practitioners who evaluated the architecture in terms of their geo-diversity, professional roles, years of experience, domain of experience etc. regarding quantum service computing in Section 6.2. The results of architecture evaluation, driven by a structured questionnaire and practitioners’ feedback are detailed in Section 6.3.



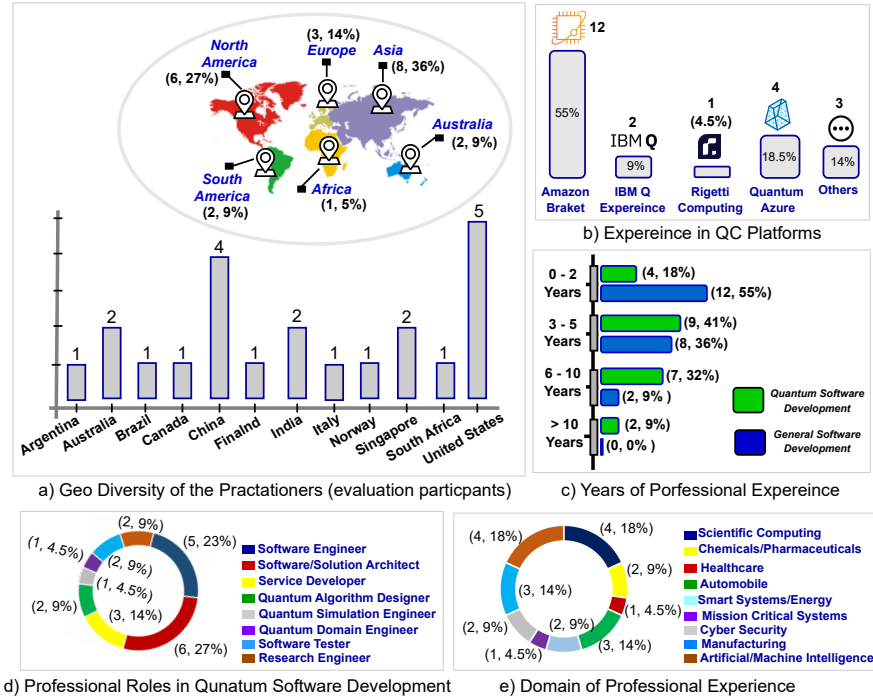


Figure 7: Overview of the Demographic Details of Practitioners.

## 6.1 Evaluation Methods and Concrete vs Reference Architectures

*Architecture evaluation methods:* Academic research and industry use-cases highlight two primary approaches to evaluate the reference architecture based on (i) scenario-based evaluation using architecture evaluation method [38] or (ii) evaluation by the domain experts using evaluation workshops or surveys [39]. Evaluation methods such as Software Architecture Analysis Method (SAAM) or Architecture Trade-off Analysis Methods (ATAM) rely on scenarios that represent use-cases for evaluating the architecture. For example, the SAAM method can represent an architecturally significant requirement as an evaluation scenario by expressing: ‘...generate a random integer that must be stored and processed to its factorisation (i.e., functionality: ‘number generation’ and ‘factorisation’). Architecture evaluation should validate this scenario by assessing if the random number has been generated and is factored. Evaluation method like SAAM helps architects to develop scenarios, prioritise them, and assess the impacts of scenarios as part of the architectural evaluation. Similarly, methods like ATAM rely on architectural scenarios to evaluate the trade-offs (e.g., usability vs efficiency) of the system under development. In contrast to the evaluation methods, evaluation surveys or workshops focus on presenting the architecture to domain experts such as software architects or developers in a particular domain (e.g., pervasive computing, mission-critical software) to review and evaluate the architecture based on pre-defined criteria such as fit for purpose or functional completeness. The evaluation criteria can be formulated as a collection of survey questions to be answered by the practitioners. For example, the studies provide a checklist [40] and empirically derived methods [32] that allow the practitioners (system stakeholders) to evaluate the reference architecture for embedded software and e-contracting systems. We followed the empirical guidelines from [38] and considered the distinction between architectural abstractions (concrete vs reference) [39] to evaluate the reference architecture.

*Evaluating concrete vs reference architectures:* From a software evaluation perspective, the study [39] differentiates between two types of architectures namely concrete architecture and reference architecture. A concrete architecture, as exemplified in Figure 5 (component and connector view) is derived from a reference architecture in Figure 4. The concrete architecture expresses the structural composition, functionality, and constraints if any of the software to be implemented as per the architectural description. In comparison, the reference architecture provides a generic template or a point of reference that provides architectural vocabulary such as modelling notations, layers, patterns etc. to express the architecture or simply instantiating a concrete architecture. Due to a generic nature

of them, reference architectures provide an abstraction of concrete architecture and existing architecture evaluation methods such as SAAM or ATAM etc. needs to be altered to evaluate reference architectures. Research in [32, 38] shows that evaluation methods need to be extended or altered by accommodating evaluation criteria such as completeness, applicability, and buildability etc. from stakeholders’ point of view. For emergent classes of software such as blockchains or quantum software there may be a lack of evaluation methods and their underlying scenarios. In case of a lack of evaluation methods, practitioners survey is one possibility to evaluate the architecture. By adhering to ISO-IEC-25010, we used functional suitability and usability as the evaluation criteria for the reference architecture. ISO-IEC-25010 standard aims to determine the quality characteristics of software artifacts, systems, and products [20], further detailed in Section 6.3.

## 6.2 Demography Details of the Practitioners

Demography details of the practitioners, who evaluated the reference architecture, are visualised in Figure 7 which highlights five aspects relating to geo-/professional diversity of the participants each detailed below. Each individual practitioner (P) was assigned a unique identifier referred to as (P1, P2, ..., P22). While collecting data based on practitioners’ feedback, the idea of saturation can provide some practical guidance for estimating sample sizes, prior to data collection, necessary for conducting qualitative evaluation. As per the guidelines in [41], conducting 12 to 15 interviews of a homogeneous group is adequate to reach saturation of data sampling. Demography details in Figure 7 complement the evaluation and provide a fine-granular interpretation of the evaluation results. For example, P12 identified as a practitioner with an experience of less than 2 years as a quantum algorithm designer, has worked with Amazon Braket (QC platform), to develop mission-critical systems. P12 indicated that 😊 ‘... quantum algorithm for simulation or optimisation can be wrapped inside microservices and it helps to translate quantum software development challenges as a problem of developing microservices ...’ and 😊 ‘... based on my prior experience with AWS (Amazon Web Services), Amazon Braket is the preferred quantum computing provider as it provides a comprehensive platform to build, test, deploy amazon web services for quantum computers ...’

- **Geo-distribution** indicates the diversity of practitioners in terms of their geo-location, as in Figure 7 a). Geo-distribution indicates feedback from practitioners across the globe, reflecting diverse participation in architecture evaluation. Geo-distribution and diversity of QC professionals can be vital in an attempt to minimise the challenge of quantum divide [10]. A total of 22 practitioners evaluated the architecture, participating from 12 different countries across 6 continents. For example, the QC practitioners from the United States (5/22, 23% approx.) and China (4/22, 18% approx.) combinedly represented a total of 41% of all the participants of the architecture evaluation.
- **Usage of QC platforms** indicates practitioners’ experience working with different QC platforms or quantum service providers, also referred to as the quantum vendors. As per Figure 7 b), practitioners indicated their experience with four different types of QC platforms with an overwhelming majority (i.e., 12/22, 55%) have experience with Amazon Braket, a finding that is consistent with the results of our mapping study on QCaaS [11]. The mapping study highlighted that Amazon Braket (a managed Amazon Web Services (AWS)) is the most preferred platform to design, test, and run quantum algorithms. One of the reasons for selecting Amazon Braket by practitioners for service deployment is that it can allow service users/developers to design their own quantum algorithms. This can be particularly handy for novice developers unfamiliar with the technicalities of quantum systems to utilise a set of pre-built algorithms, tools, and documents to develop and manage quantum services on Amazon platform. Two practitioners, i.e., 14% approx. indicated ‘Others’ as the QC platforms of their experience.
- **Years of experience** highlights practitioners’ professional experience to engineer (i) general class of software such as web or mobile systems, and (ii) quantum software and services, as in Figure 7 c). The professional experience quantified as the number of years highlights that a total of 12/22, i.e., 55% of the participants have an experience of two years or less. On the contrary, 73% of practitioners’ reflected experience between 3 to 10 years with the development of general software systems.

- **Professional roles** indicate practitioners' expertise in engineering and development of software systems and/or services for quantum software as in Figure 7 d). Practitioners' responses identified a total of 8 roles highlighted in Figure 7 d) including but not limited to software/solution architect, quantum algorithm designer, and quantum domain engineer. Quantum domain engineer is considered as a QSE-specific role that enables software engineers to map the requirements of domain with software systems such as manipulating Qubits expressed as the configuration of the software components and services [18]. Software engineers, software/solution architects, and service developers were identified as the predominant roles in quantum service development, collectively representing 14/22, i.e., 64% of all the participants.
- **Domain of experience** refers to the context or area of application that requires quantum software to perform computation or enable automation in a particular domain, highlighted in Figure 7 e). For example, automobile indicates a domain for which quantum software or services can be developed to provide simulation, optimisation, or decision support for researching and developing automobile technologies. Scientific computing, artificial intelligence/machine learning, and automobile represent the predominant domain of professional expertise indicated by a total of 11/22, i.e., 50% of the participants.

Demography details summarised above can enrich the analysis of architecture evaluation results. Specifically, demography data as extended details can help us understand if practitioners' perspectives on architecture evaluation may be influenced by certain factors such as their years of experience, professional roles, or knowledge of any specific QC platforms. For example, one of the practitioner expressed ☺ '*... my experience with microservices development, in particular working with (Amazon) Braket that is effectively an Amazon managed web service endorses layering a quantum system into classic and quantum parts. Layering or split in this case may be intuitive, however; such a split could be counter-productive if ill-designed. What I mean is that theoretically, it looks appropriate to structure your system into the classic and quantum parts, split of a quantum program (pre-processing) and then merge the compute results (post-processing) may be time-consuming and it may result into an anti-pattern.*'

### 6.3 Results of the Practitioners' Evaluation

The results of architecture evaluation based on the practitioners' feedback and its analysis are summarised in Figure 8. For the clarity of presentation, the results, as per Figure 8 are organised along two dimensions, i.e., survey questions (presented at the vertical axis) and practitioners' responses to the questions (presented at the horizontal axis), detailed below.

#### 6.3.1 Presenting the Survey Questionnaire

The survey questionnaire comprising of a total of 11 questions (Q1 - Q11) to evaluate architecture in terms of architectural elements including architecture layering, architecting phases, architecture activities, architecture artifacts, human roles, and general feedback, presented on the horizontal axis. For example, as shown in Figure 8, the question on architecture layering seeks the practitioner's view if: '*the three layers, i.e., service development, service deployment, service split are appropriate and sufficient to structure the architecture*'. The question (Q1) follows up by seeking any suggestion (Q2) by asking: '*Suggest any layers that you deem as missing or should be part of the reference architecture*'. One of the survey questions (Q11) aims to gather spontaneous feedback on the overall architecture by asking: '*Provide any general feedback regarding (potential improvements, strengths, limitations, or any missing element) that can help us refine the architecture*'.

#### 6.3.2 Presenting the Survey Questionnaire

The practitioner's feedback to the questionnaire is visually summarised along the horizontal axis that highlights 4 types of information. The information includes a five-scale Likert option for each question (range: Strongly Agree to Strongly Disagree), bar and pie graphs to quantify the Likert response, a summary of the recommendations provided by the practitioners, and a view of potential refinement that could be made to the architecture based on the recommendation. For example, in response to Q1, i.e., is the presenting layering sufficient and appropriate to architect quantum service-orientation,

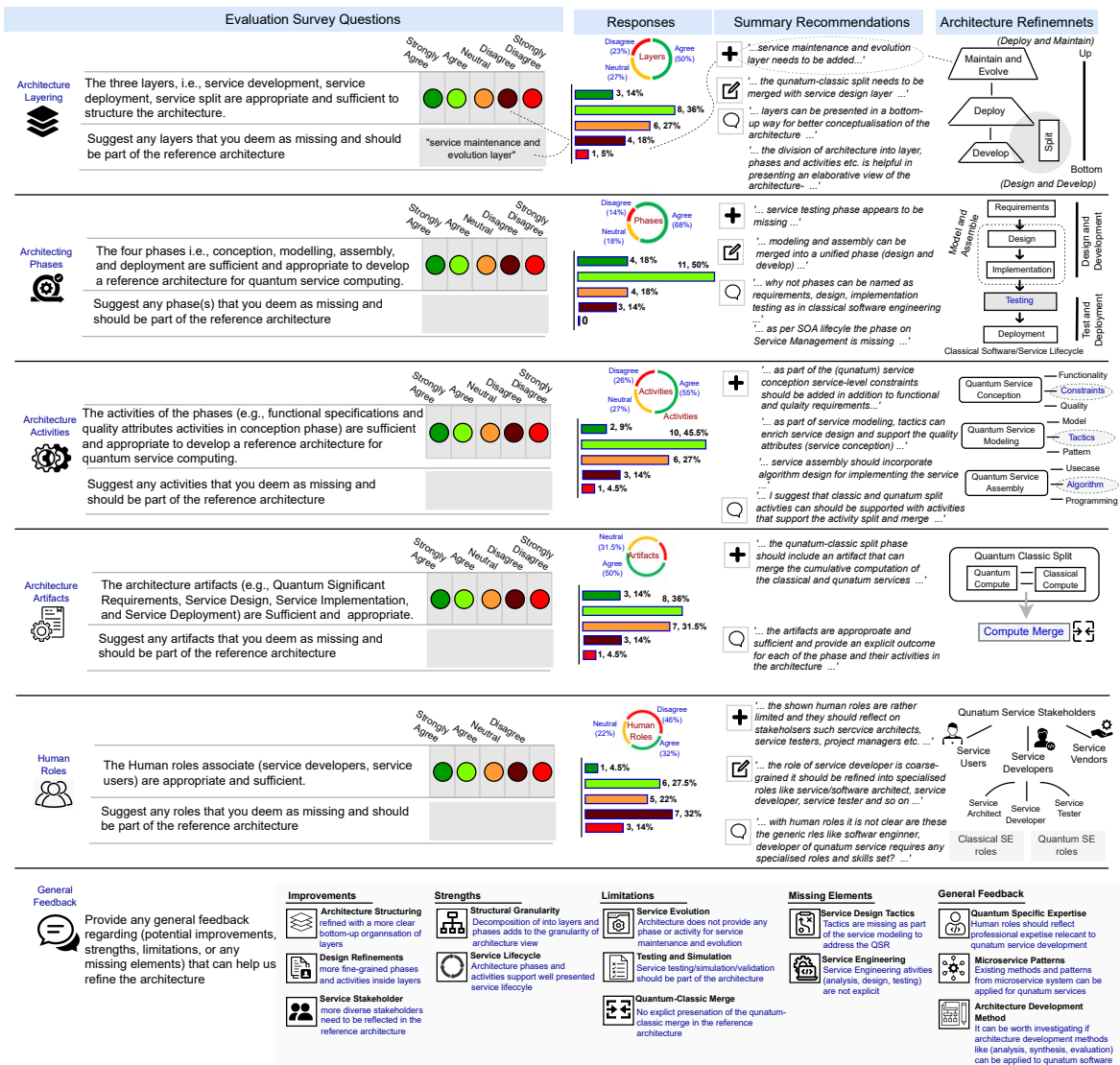


Figure 8: Summary of Results Based on Practitioners' Evaluation.

the practitioners responded as: Strongly Agree (3 respondents: 14% approx.), Agree (8, 36%), Neutral (6, 27%), Disagree (4, 18%), and Strongly Disagree (1, 5%). This reflects a cumulative agreement to sufficiency and appropriateness of the layering as 50%, neutrality as 27% and 23% disagreement. As an example, one of the suggestions corresponding to Q2 about the architecture layering was about the addition of 🗨️ ‘... *service maintenance and evolution layer*’ that appears as missing as per a response from one of the practitioners. The recommendations (Q2, Q4, Q6, Q8, Q10) complementary to each of the Likert question (Q1, Q3 etc.) generally represent any addition, removal, modification or general improvement comments. The suggestion about addition, removal, or modification of any architecture elements are viewed as modification operations. For example, the suggestion 🗨️ ‘... *service maintenance and evolution layer needs to be added...*’ is classified as an addition of a specific architecture element, an additional layer in this case. Similarly, the suggestion 🗨️ ‘...*the quantum-classic split layer needs to be merged with service design layer...*’ represents a modification of the existing architecture. The general suggestion 🗨️ ‘...*layers can be presented in a [classical] bottom-up way for a better conceptualisation of the architecture ...*’.

Q11 represents an exception as it seeks a relatively spontaneous response, as overall suggestions for improvements, identified strengths, limitations, any missing elements, or any conclusive general feedback about the reference architecture. We exemplify some of the responses. In terms of any suggested improvements, one of the practitioners suggested **Architecture Structuring** can be 🗨️ ‘*refined with a more clear bottom-up organisation of the layers*’. **Service Lifecycle** presentation reflects a strength of the architecture where 🗨️ ‘*architecture phases and activities support well-presented service lifecycle*’. One of the identified limitations was missing **Testing and Simulation** suggesting 🗨️ ‘*service testing/simulation/validation should be part of the architecture*’. One of the missing factors relates to **Service Design Tactics** 🗨️ ‘*tactics are missing as part of the service modelling to address the QSR*’. Finally, the General Feedback highlighted a multitude of suggestions such as investigating and integrating **Architecture Development Method** indicating 🗨️ ‘*it can be worth investigating if architecture development methods like (analysis, synthesis, evaluation) can be applied to quantum software*’.

## 7 Related Work

We now discuss the related work in terms of published research on architecting quantum software (Section 7.1) and quantum service computing (Section 7.2). From the QSE perspective, a synoptical view of the most relevant research helps us to contextualise the scope and contributions of the proposed research in terms of architecting quantum computing as a service.

### 7.1 Architecting Software-intensive Systems for QCs

Quantum software engineering entails a multitude of engineering activities that can range from quantum domain modelling or quantum algorithmic design to quantum simulation management to support a systematic and incremental development of quantum software and services [2, 17]. QSE acts an umbrella to support different phases of software development that include but are not limited to architecting [18], programming [23], and testing [42] of quantum software. The role of quantum software architectures in QSE becomes pivotal as it allows designers and developers to map the QSRs to a software model that is independent of technical implementations and acts as a blue-print to implement and validate quantum software. A recently conducted systematic review of quantum software architectures indicates the role that architectural processes, patterns, and tools play to empower the role of software developers to design and implement quantum software [33]. The systematic review indicates that QSE is a relatively new engineering paradigm and often software engineers and developers find themselves underprepared and often lack the expertise to tackle quantum-specific software development challenges. In-line with the findings of the systematic review in [33], the research in [18] presents an architectural process composed of a number of architecting activities and highlights the needs for professional expertise (i.e., human roles in QSA) to effectively develop quantum software. To support architecture-centric engineering and development, Quantum UML profile as an extension of the Unified Modeling Language (UML) supports modelling and architecting quantum software [35]. Quantum UML-based models (e.g., class, use-case, deployment diagrams) can be transformed into implementation or executable specification using quantum model-driven engineering [19]. A well-crafted architec-



ture can abstract the implementation details via architectural components and can assist developers to achieve model-to-code transformations by exploiting model-driven QSE [35, 18, 19]. Considering architecture as a point of reference for QSE, there is no reference architecture that can consolidate system design knowledge in terms of patterns, modelling notations, service use case, and human-roles to support architecture-centric design and development of quantum service computing.

## 7.2 Quantum Service Computing

A recently conducted mapping study on quantum computing as a service investigated existing research and identified some trends for future research that unifies quantum computing and service-oriented software [11]. Results of the study indicate that emerging and future work on quantum service computing relies on patterns and tactics, low-code development, and agile practices for quantum service design and implementation. The status-quo on quantum service-orientation reflects experimental research on implementing quantum algorithms [43] and functions [34] implemented as microservices that can be executed on QC platforms such as Amazon Braket. Although the use cases for quantum service computing (e.g., number crunching, mathematical optimisation) are rather limited, however, work is in progress to exploit QC platforms in software services context of bio-inspired computing [44]. Specifically, the study proposes a tool-chain for quantum cloud computing where computation-intensive tasks can be outsourced from classical machines to quantum servers that are configured via quantum cloud computing model [29]. Some studies have demonstrated that existing development processes (e.g., DevOps) and service architectural styles (microservices) can be extended and successfully applied to quantum service computing [45]. Moreover, a number of classical patterns such as service wrapping or API gateway have been successfully applied to design software-intensive systems and services for quantum computing [12]. These classical patterns when combined with quantum specific patterns such as quantum-classic split can help develop a catalogue of patterns that can help novice developers and engineers to rely on reuse knowledge and best practices for quantum service-orientation driven by microservices. Quantum service computing is being envisioned as a utility computing model that can breach the quantum divide by offering QC resources such as QPU, memory or simulators to end-users. However, there is a lack of processes and patterns to enable a systemised development of quantum service-oriented solutions. Recent studies on quantum service systems indicate the challenge of empirically investigating the extent to which classical service-orientation knowledge can be reused in the context of QCaaS [11, 12, 34, 43]. The reference architecture can enable engineers to abstract complexities of quantum source code into architectural components, apply reuse knowledge via quantum software patterns, and adopt best practices such as microservices architecture style to develop QCaaS. Proof-of-the-concept as a prototype can enable QC users to discover and utilise hardware, software, and networking resources offered by QC vendors via quantum software services [21].

*Conclusive Summary:* Based on the review of the most relevant existing research, we conclude that architectural models help software engineers to tackle design and development issues by abstracting complex and implementation specific (i.e., source coding details) with high-level architectural components [33]. Research and development on QSE [2] in general and QSA [18] to be more specific highlights that lack of professional expertise and unique challenges of quantum software development require developers to rely on reference models, patterns, and processes to develop quantum software services. The proposed solution complements the emerging research on QSE and QSA and more specifically focuses on an empirically grounded reference architecture that acts as a point of reference or a blueprint to implement quantum service orientation. The proposed solution draws inspiration from recommendations and guidelines from a generic model in [44] for quantum cloud computing and pattern-based service development [26] to architect and implement quantum software services. The proposed solution advocates the need to establish reference architectures that can provide foundations to apply architectural knowledge, process, and principle that can synergise classical and quantum approaches for quantum service development.

## 8 Threats to the Validity of Research

This study draws empirically-grounded evidence from SMS and uses practitioners' feedback for evaluation thus inheriting some threats to the validity [46]. These threats represent potential limitations, constraints, or flaws in the study that can impact various aspects like the generalisation, replicability,



and validity of results. Any future research that relies on the presented study in terms of research design or its results must consider these threats. Future work should focus on minimising these threats to ensure methodological rigor and generalisation for avoiding any bias in the results. We discuss three main types of threats, each detailed below.

### 8.1 Threat I - Internal Validity

It examines the extent to which any systematic error (bias) is present in the design, conduct, and analysis etc. of the study. In the context of internal validity, we refer to the research method (Figure 3) that overview different steps to design and conduct the study to present its results. To minimise this threat, we derived the reference architecture from a mapping study [11] that followed guidelines of evidence-based software engineering to objectively collect and analyse the data [30]. Based on the collected data, we aligned the evidence from the mapping study to the phases and activities of IBM SOA lifecycle to create the reference architecture for quantum service-orientation [21]. We relied on an architecture use case to develop a proof-of-the-concept implementation of the reference architecture. The steps as part of research methods aim to minimise the bias and threats to internal validity. However, more work is required to understand and assess if the study results can be validated with a different architecting process or by adopting other evaluation methods.

### 8.2 Threat II - External Validity

It examines whether the findings of a study can be generalised to other contexts. From an external validity perspective, it needs to be determined if the same process can be applied to develop other reference architectures or this reference architecture can be used to develop other systems in a quantum computing context. We only experimented with a single case study of prime factorisation and only two patterns classic quantum split and orchestrator patterns of moderate complexity that can compromise the study’s generalisation. Specifically, scenarios with the increased complexity of architecting process (quantum simulations), types of patterns (microservice patterns), and human expertise (novice/experienced engineers) can affect the external validity of this research. We did try to minimise the external validity by engaging 22 QSE practitioners and their feedback to improve the suitability and usability of the reference architecture [20]. Future work requires more rigorous evaluation, preferably in a more practical industrial context to further assess the external validity of the research.

### 8.3 Threat III - Conclusion Validity

It determines the degree to which the conclusions reached by the study are credible or believable. In order to minimise this threat, we followed a three-step process (Figure 3) to support a fine-grained process to architect (Figure 4) the software and validate the results (Figure 5). Moreover, a case study-based approach was adopted to ensure scenario-based demonstration of the study results. However, some conclusions (e.g., practitioners’ perspective, practical context) can only be validated with more experimentation involving multiple case studies, and real context scenarios of architecting quantum service computing.

## 9 Conclusions

Quantum computing has started to emerge as a disruptive technology – striving to offer quantum computational supremacy over traditional digital computers - by exploiting hardware, software, and networking technologies that are driven by the operations of quantum bits and quantum gates. QCs are in a phase of their inception, however; they have started to demonstrate their computing superiority in areas that range from bio-inspired systems, data security and cryptography solutions to tackling optimization problems. A plethora of issues such as hardware limitations, lack of software ecosystems, scarcity of human expertise to engineer QCs ecosystem, and quantum error rate impede a wide-scale adoption and commercially viable solutions of quantum computing. This research aims to synergise quantum software engineering and service computing to architect quantum service orientation for pay-per-shot usability of QC resources. Specifically, the research focuses on unifying quantum software engineering methods and service-orientation patterns to promote reuse knowledge and best practices

to tackle emerging and futuristic challenges of architecting and implementing Quantum Computing as a Service (QCaaS).

*Primary contributions and implications:* The primary contribution of this research include (a) an empirically-derived reference architecture as a blue-print to develop software services for QCaaS, (b) a proof-of-the-concept that demonstrates architecture-centric implementation of quantum software services, and (iii) practitioners' evaluation of the reference architecture that provides recommendations and guidelines to design and develop solutions for QCaaS. The research can have implications for researchers and practitioners of quantum software engineering. Specifically, the results of the study help academic researchers to understand the role of reference architectures and quantum service-orientation the challenges of QSE in the context of QCaaS. Moreover, the practitioners can explore the reference architecture as a system blueprint and patterns as reuse knowledge that can be adopted to develop solutions for QCaaS.

*Needs for future research:* Based on the study results, we envision future work in two directions with a focus on empirical research including (a) mining social coding platforms and (b) practitioners' interview to further understand the architecting and development of quantum computing as a service. Specifically, by mining social coding platforms (e.g., GitHub) we can empirically discover knowledge and understand the practices adopted by developers' communities in open-source QCaaS. The study provides foundations to design and conduct semi-structured interviews by engaging service developers and engineers to seek their feedback and synthesise the results as practitioners' perspectives to complement the evidence from design and implementation of the reference architecture.

## References

- [1] A. W. Harrow and A. Montanaro, "Quantum computational supremacy," *Nature*, vol. 549, no. 7671, pp. 203–209, 2017.
- [2] S. Ali, T. Yue, and R. Abreu, "When software engineering meets quantum computing," *Communications of the ACM*, vol. 65, no. 4, pp. 84–88, 2022.
- [3] W.-L. Chang and A. V. Vasilakos, *Fundamentals of Quantum Programming in IBM's Quantum Computers*. Springer, 2021.
- [4] M. A. Cusumano, "The business of quantum computing," *Communications of the ACM*, vol. 61, no. 10, pp. 20–22, 2018.
- [5] T. S. Humble, A. McCaskey, D. I. Lyakh, M. Gowrishankar, A. Frisch, and T. Monz, "Quantum computers for high-performance computing," *IEEE Micro*, vol. 41, no. 5, pp. 15–23, 2021.
- [6] C. Gonzalez, "Cloud based qc with amazon braket," *Digitale Welt*, vol. 5, pp. 14–17, 2021.
- [7] M. Riedel, M. Kovacs, P. Zoller, J. Mlynek, and T. Calarco, "Europe's quantum flagship initiative," *Quantum Science and Technology*, vol. 4, no. 2, p. 020501, 2019.
- [8] "State of quantum computing: Building a quantum economy." <https://www.weforum.org/reports/state-of-quantum-computing-building-a-quantum-economy/>. Accessed: 2023-01-25.
- [9] R. Biswas, Z. Jiang, K. Kechezhi, S. Knysh, S. Mandra, B. O'Gorman, A. Perdomo-Ortiz, A. Petukhov, J. Realpe-Gómez, E. Rieffel, *et al.*, "A nasa perspective on quantum computing: Opportunities and challenges," *Parallel Computing*, vol. 64, pp. 81–98, 2017.
- [10] "The world is heading for a 'quantum divide': here's why it matters." <https://www.weforum.org/agenda/2023/01/the-world-quantum-divide-why-it-matters-davos2023/>. Accessed: 2023-01-27.
- [11] A. Ahmad, M. Waseem, P. Liang, M. Fehmideh, A. A. Khan, D. G. Reichelt, and T. Mikkonen, "Engineering software systems for quantum computing as a service: A mapping study," *arXiv preprint arXiv:2303.14713*, 2023.

- [12] J. Garcia-Alonso, J. Rojo, D. Valencia, E. Moguel, J. Berrocal, and J. M. Murillo, “Quantum software as a service through a quantum api gateway,” *IEEE Internet Computing*, vol. 26, no. 1, pp. 34–41, 2021.
- [13] M. G. Raymer and C. Monroe, “The us national quantum initiative,” *Quantum Science and Technology*, vol. 4, no. 2, p. 020504, 2019.
- [14] A. Bouguettaya, M. Singh, M. Huhns, Q. Z. Sheng, H. Dong, Q. Yu, A. G. Neiat, S. Mistry, B. Benatallah, B. Medjahed, *et al.*, “A service computing manifesto: the next 10 years,” *Communications of the ACM*, vol. 60, no. 4, pp. 64–72, 2017.
- [15] A. Bouguettaya, M. Singh, M. Huhns, Q. Z. Sheng, H. Dong, Q. Yu, A. G. Neiat, S. Mistry, B. Benatallah, B. Medjahed, *et al.*, “Global cloud computing revenue by segment 2015-2022 — statista,” <https://stagingfr.statista.com/statistiques/540499/worldwide-cloud-computing-revenue-by-segment/>.
- [16] V. Andrikopoulos and P. Lago, “Software sustainability in the age of everything as a service,” *Next-Gen Digital Services. A Retrospective and Roadmap for Service Computing of the Future: Essays Dedicated to Michael Papazoglou on the Occasion of His 65th Birthday and His Retirement*, pp. 35–47, 2021.
- [17] J. Zhao, “Quantum software engineering: Landscapes and horizons,” *arXiv preprint arXiv:2007.07047*, 2020.
- [18] A. Ahmad, A. A. Khan, M. Waseem, M. Fahmideh, and T. Mikkonen, “Towards process centered architecting for quantum software systems,” in *2022 IEEE International Conference on Quantum Software (QSW)*, pp. 26–31, IEEE, 2022.
- [19] F. Gemeinhardt, A. Garmendia, and M. Wimmer, “Towards model-driven quantum software engineering,” in *2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE)*, pp. 13–15, IEEE, 2021.
- [20] P. Nistala, K. V. Nori, and R. Reddy, “Software quality models: A systematic mapping study,” in *2019 IEEE/ACM International Conference on Software and System Processes (ICSSP)*, pp. 125–134, IEEE, 2019.
- [21] M. Keen, G. Ackerman, I. Azaz, M. Haas, R. Johnson, J. Kim, and P. Robertson, “Patterns: Soa foundation-business process management scenario,” *IBM Redbooks*, 2006.
- [22] S. F. Lin, S. Sussman, C. Duckering, P. S. Mundada, J. M. Baker, R. S. Kumar, A. A. Houck, and F. T. Chong, “Let each quantum bit choose its basis gates,” in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1042–1058, IEEE, 2022.
- [23] M. De Stefano, F. Pecorelli, D. Di Nucci, F. Palomba, and A. De Lucia, “Software engineering for quantum programming: How far are we?,” *Journal of Systems and Software*, vol. 190, p. 111326, 2022.
- [24] F. T. Chong, D. Franklin, and M. Martonosi, “Programming languages and compiler design for realistic quantum hardware,” *Nature*, vol. 549, no. 7671, pp. 180–187, 2017.
- [25] P. Murali, D. C. McKay, M. Martonosi, and A. Javadi-Abhari, “Software mitigation of crosstalk on noisy intermediate-scale quantum computers,” in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 1001–1016, 2020.
- [26] F. Leymann, “Towards a pattern language for quantum algorithms,” in *Quantum Technology and Optimization Problems: First International Workshop, QTOP 2019, Munich, Germany, March 18, 2019, Proceedings 1*, pp. 218–230, Springer, 2019.
- [27] M. Weigold, J. Barzen, F. Leymann, and D. Vietz, “Patterns for hybrid quantum algorithms,” in *Service-Oriented Computing: 15th Symposium and Summer School, SummerSOC 2021, Virtual Event, September 13–17, 2021, Proceedings 15*, pp. 34–51, Springer, 2021.

- [28] J.-F. Bobier, M. Langione, E. Tao, and A. Gourevitch, “What happens when ‘if’ turns to ‘when’ in quantum computing,” *Boston Consulting Group*, 2021.
- [29] D. Castelvechi, “Ibm’s quantum cloud computer goes commercial,” *Nature*, vol. 543, no. 7644, 2017.
- [30] K. Petersen, S. Vakkalanka, and L. Kuzniarz, “Guidelines for conducting systematic mapping studies in software engineering: An update,” *Information and software technology*, vol. 64, pp. 1–18, 2015.
- [31] C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, and P. America, “A general model of software architecture design derived from five industrial approaches,” *Journal of Systems and Software*, vol. 80, no. 1, pp. 106–126, 2007.
- [32] M. Yli-Ojanperä, S. Sierla, N. Papakonstantinou, and V. Vyatkin, “Adapting an agile manufacturing concept to the reference architecture model industry 4.0: A survey and case study,” *Journal of industrial information integration*, vol. 15, pp. 147–160, 2019.
- [33] A. A. Khan, A. Ahmad, M. Waseem, P. Liang, M. Fahmideh, T. Mikkonen, and P. Abrahamsson, “Software architecture for quantum computing systems—a systematic review,” *Journal of Systems and Software*, p. 111682, 2023.
- [34] H. T. Nguyen, M. Usman, and R. Buyya, “Qfaas: A serverless function-as-a-service framework for quantum computing,” *arXiv preprint arXiv:2205.14845*, 2022.
- [35] R. Pérez-Castillo and M. Piattini, “Design of classical-quantum systems with uml,” *Computing*, vol. 104, no. 11, pp. 2375–2403, 2022.
- [36] L. M. Vaquero, F. Cuadrado, Y. Elkhatib, J. Bernal-Bernabe, S. N. Srirama, and M. F. Zhani, “Research challenges in nextgen service orchestration,” *Future Generation Computer Systems*, vol. 90, pp. 20–38, 2019.
- [37] M. Weigold, J. Barzen, F. Leymann, and D. Vietz, “Patterns for hybrid quantum algorithms,” in *Service-Oriented Computing: 15th Symposium and Summer School, SummerSOC 2021, Virtual Event, September 13–17, 2021, Proceedings 15*, pp. 34–51, Springer, 2021.
- [38] J. Lee, S. Kang, and C.-K. Kim, “Software architecture evaluation methods based on cost benefit analysis and quantitative decision making,” *Empirical Software Engineering*, vol. 14, pp. 453–475, 2009.
- [39] S. Angelov, J. J. Trienekens, and P. Grefen, “Towards a method for the evaluation of reference architectures: Experiences from a case,” in *Software Architecture: Second European Conference, ECSA 2008 Paphos, Cyprus, September 29-October 1, 2008 Proceedings 2*, pp. 225–240, Springer, 2008.
- [40] J. F. M. Santos, M. Guessi, M. Galster, D. Feitosa, and E. Y. Nakagawa, “A checklist for evaluation of reference architectures of embedded systems (s).,” in *SEKE*, vol. 13, pp. 1–4, 2013.
- [41] G. Guest, A. Bunce, and L. Johnson, “How many interviews are enough? an experiment with data saturation and variability,” *Field methods*, vol. 18, no. 1, pp. 59–82, 2006.
- [42] A. García de la Barrera, I. García-Rodríguez de Guzmán, M. Polo, and M. Piattini, “Quantum software testing: State of the art,” *Journal of Software: Evolution and Process*, p. e2419, 2021.
- [43] M. De Stefano, D. Di Nucci, F. Palomba, D. Taibi, and A. De Lucia, “Towards quantum-algorithms-as-a-service,” in *Proceedings of the 1st International Workshop on Quantum Programming for Software Engineering*, pp. 7–10, 2022.
- [44] J. Barzen, F. Leymann, M. Falkenthal, D. Vietz, B. Weder, and K. Wild, “Relevance of near-term quantum computing in the cloud: A humanities perspective,” in *Cloud Computing and Services Science: 10th International Conference, CLOSER 2020, Prague, Czech Republic, May 7–9, 2020, Revised Selected Papers*, pp. 25–58, Springer, 2021.

- [45] A. A. Khan, M. A. Akbar, A. Ahmad, M. Fahmideh, M. Shameem, V. Lahtinen, M. Waseem, and T. Mikkonen, “Agile practices for quantum software development: Practitioners perspectives,” *arXiv preprint arXiv:2210.09825*, 2022.
- [46] A. Ampatzoglou, S. Bibi, P. Avgeriou, M. Verbeek, and A. Chatzigeorgiou, “Identifying, categorizing and mitigating threats to validity in software engineering secondary studies,” *Information and Software Technology*, vol. 106, pp. 201–230, 2019.