The Journal of Systems & Software 201 (2023) 111682

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Software architecture for quantum computing systems — A systematic review☆

Arif Ali Khan [a,*], Aakash Ahmad [b], Muhammad Waseem [c], Peng Liang [c], Mahdi Fahmideh [d], Tommi Mikkonen [e], Pekka Abrahamsson [e]

[a] M3S Empirical Software Engineering Research Unit, University of Oulu, 90014 Oulu, Finland
[b] School of Computing and Communications, Lancaster University, Leipzig, Germany
[c] School of Computer Science, Wuhan University, Wuhan, China
[d] School of Business at University of Southern Queensland, Queensland, Australia
[e] Faculty of Information Technology and Communication Sciences, Tampere University, 33014 Tampere, Finland

ARTICLE INFO

Article history:
Received 16 July 2022
Received in revised form 10 March 2023
Accepted 20 March 2023
Available online 24 March 2023

Keywords:
Quantum computing
Quantum software engineering
Quantum software architecture
Systematic literature review

ABSTRACT

Quantum computing systems rely on the principles of quantum mechanics to perform a multitude of computationally challenging tasks more efficiently than their classical counterparts. The architecture of software-intensive systems can empower architects who can leverage architecture-centric processes, practices, description languages to model, develop, and evolve quantum computing software (quantum software for short) at higher abstraction levels. We conducted a Systematic Literature Review (SLR) to investigate (i) architectural process, (ii) modelling notations, (iii) architecture design patterns, (iv) tool support, and (iv) challenging factors for quantum software architecture. Results of the SLR indicate that quantum software represents a new genre of software-intensive systems; however, existing processes and notations can be tailored to derive the architecting activities and develop modelling languages for quantum software. Quantum bits (Qubits) mapped to Quantum gates (Qugates) can be represented as architectural components and connectors that implement quantum software. Tool-chains can incorporate reusable knowledge and human roles (e.g., quantum domain engineers, quantum code developers) to automate and customise the architectural process. Results of this SLR can facilitate researchers and practitioners to develop new hypotheses to be tested, derive reference architectures, and leverage architecture-centric principles and practices to engineer emerging and next generations of quantum software.

1. Introduction

Quantum computing relies on quantum mechanics, a discipline more familiar and centre of attention to physicists rather than computer scientists or software engineers (Zhao, 2020; Deutsch, 1985; Dirac, 1981). However, in recent years, with an emergence of quantum algorithms and Quantum Programming Languages (QPL), software programmers have been able to exploit the theory and principle of quantum mechanics to process information and perform specific computation tasks faster than classical computing systems (Chong et al., 2017; Ying, 2016). Compared to classical algorithms for computation, quantum algorithms have the potential to solve a set of computationally challenging problems such as nature-inspired computing, financial modelling, and advanced encryption with increased efficiency (Montanaro, 2016; Grimsley et al., 2019; Krüger and Mauerer, 2020). Quantum computing attributes (e.g., Qubits, superposition, entanglement, and interference) lie at the heart of quantum information processing (Zeilinger, 1999; Gay, 2006). Quantum programming languages that implement quantum algorithms enable quantum supremacy in computing that is lacking in traditional computing systems. Montanaro (2016), Gay (2006), Sofge (2008). One class of such problems relate to information and computation science that requires large amounts of parallel processing (Nguyen et al., 2022) for tackling challenges, such as optimisation, encryption, big data analytics, and machine learning (Biamonte et al., 2017; Rebentrost et al., 2014). Other set of problems relate to efficient and accurate simulation of quantum systems in natural sciences, such as physics (Grimsley et al., 2019), chemistry (McArdle et al., 2020), mathematics (Krüger and Mauerer, 2020), and challenges relating to their applications (Stepney et al., 2005;

☆ Editor: Prof. Neil Ernst.

* Corresponding author.
E-mail addresses: arif.khan@oulu.fi (A.A. Khan), a.ahmad13@lancaster.ac.uk (A. Ahmad), m.waseem@whu.edu.cn (M. Waseem), liangp@whu.edu.cn (P. Liang), mahdi.fahmideh@usq.edu.au (M. Fahmideh), tommi.j.mikkonen@jyu.fi (T. Mikkonen), pekka.abrahamsson@tuni.fi (P. Abrahamsson).

https://doi.org/10.1016/j.jss.2023.111682

Childs et al., 2018; Mosca, 2018). However, QPL and their underlying algorithms focus on computation and implementation details to produce executable specifications, but lack an overall global view of the software systems under design. Source code based implementation details undermine architectural view(s) as system blueprint, that can compromise the quality and functionality of end product, i.e., quantum software (Zhao, 2020; Moguel et al., 2020; Piattini et al., 2021). Technology giants are scaling up their financial and strategic investments in quantum computing platforms, more specifically quantum programming languages such as Q# from Microsoft, Qiskit from IBM, and Cirq from Google, however; quantum software engineering and development is still in its infancy (Microsoft, 2021; Behera et al., 2019; Courtland, 2017). Some recent research studies also indicate that quantum software projects that overlook design principles to primarily focus on quantum source code implementations, often lead to faulty implementations and bugs in quantum software (Zhao et al., 2021; Campos and Souto, 2021).

Software architecture as described in the ISO/IEC 42010 standard provides a global view of software-intensive systems, representing their blue-print, by abstracting complex implementation details with architectural components and connectors (Anon, 2022; Hofmeister et al., 2007; Li et al., 2013). Software developers and architects have successfully used architectural descriptions and specifications to design, develop, validate, and evolve software-intensive system at higher-level of abstractions while maintaining system functionality and quality (Malavolta et al., 2012; Di Francesco et al., 2019). Architectural models have been exploited to design, develop, and validate emerging generations of software-intensive systems including but not limited to the internet of things, blockchain applications, and artificially intelligent systems (Alreshidi and Ahmad, 2019; Fahmideh et al., 2021a; Xu et al., 2017; Fahmideh et al., 2021b; Graef and Georgievski, 2021). Quantum Software Architecture (QSA), as a new genre of Software Architectures (SA), can provide architectural descriptions (i.e., components, connectors, and configurations) to design and develop quantum software, while abstracting complex and implementation specific tasks (Hofmeister et al., 2007; Garcia et al., 2021). Specifically, architectural components can represent modules of source code while architectural connectors specify interactions between modules to represent the structure and behaviour of a system (Garcia et al., 2021). Transformation from abstract high-level models (i.e., design artifacts) to low-level executable specifications (i.e., source code artifacts) can be enabled via model-driven architecting of quantum software (Moin et al., 2021; Pérez-Castillo et al., 2021). However, QSA as an emerging discipline remains an under-explored area by the current generation of designers and architects who find themselves less prepared to tackle the challenges related to QSA in the development life-cycle of quantum software (Magnani, 2022; Shepard, 2021; Naveh, 2021). Despite a plethora of published research in recent years that focuses on engineering and architecting quantum software, there do not exist any evidence, i.e., empirical study or data-driven analysis to consolidate a collective impact of existing research on architecting quantum software (Garcia et al., 2021; Moin et al., 2021; Pérez-Castillo et al., 2021; Magnani, 2022).

Systematic Literature Reviews (SLRs) rely on Evidence-based Software Engineering (EBSE) approach to identify, classify, compare, and synthesise published research as an evidence to empirically investigate the topic under investigation (Di Francesco et al., 2019; Kitchenham and Charters, 2007). Recently, a number of SLRs and review based studies have been conducted to investigate the application of Software Engineering (SE) to quantum computing systems, however; there is no effort to review the state-of-the-art on architecting quantum software (Zhao, 2020; Moguel et al., 2020; Piattini et al., 2021; Gill et al., 2022). Therefore, the objective of this review is to complement SE based studies and specifically focus on *identification, classification, and synthesis of the published research on the role that software architecture plays in developing quantum computing systems*. We aim to investigate the core concepts, underpinning fundamentals of software architectural aspects, often overlooked in SE focused studies, by outlining a number of Research Questions (RQs). These RQs focus on (i) architectural process (unifying architecting activities), (ii) modelling notations (architectural representation), (iii) patterns and design decisions (reusable knowledge and best practices), (iv) tool support (enabling automation and customisation) and (v) emerging challenges for quantum software architectures. These RQs are motivated by academic research and industrial studies on software architecture that highlight the needs for process-centric architecting, where a process acts as an umbrella to support various architectural aspects (Hofmeister et al., 2007; Malavolta et al., 2012). Moreover, in quantum software engineering lifecycle (Dey et al., 2020), during system design, architectural aspects such as software modelling, patterns, tools, and human roles are as fundamental for architecture-centric engineering of quantum software (Piattini et al., 2021). Results and findings of this SLR complement existing surveys on Quantum Software Engineering (QSE) and can provide foundations for further secondary studies that can explore architectural principles and practices to design and develop quantum software.

The results of this SLR indicate that although quantum software represents a new generation of software applications, foundations for quantum software architectures are grounded in architectural processes and architecting activities of classical systems (e.g., object, service, or component-based) (Krüger and Mauerer, 2020; Malavolta et al., 2012; Di Francesco et al., 2019; DiAdamo et al., 2021). Quantum-specific features involving Qubits (e.g., quantum entanglement and quantum superposition) elaborated later, do require tailored architectural processes and modelling notations, such as exploiting the Unified Modelling Language (UML) to effectively address the challenges of the quantum age architectures (Pérez-Castillo et al., 2021). Specifically, existing processes and notations need customisation to enable co-design of quantum systems that can enable the mapping between Qugates and Qubits to software architectural components and connectors. Tool-chain to support quantum architecting process can facilitate system and software architects to achieve automation and incorporate human decision support while designing and implementing quantum software. The results of the SLR can be beneficial for:

(i) Researchers who are interested in understanding theory and principles of architecture-intensive development, establishing new hypotheses to be tested, and developing reference architectures and solutions for quantum software.
(ii) Practitioners who would like to understand the architecting activities, patterns as reusable knowledge, existing and required tool chain, and the extent to which the academic research can be leveraged to develop industry scale solutions for quantum software.

The rest of the paper is organised as follows: Section 2 presents the context and background of this research study. Section 3 details the research methodology to conduct the study. Sections 4–5 present the result of the study. Section 6 discusses the core finding and implications of the study results. Section 7 elaborated on threats to the validity of the research. Section 8 reviews and provides comparative analysis of the most relevant research studies. Section 9 concludes the study with a discussion of potential future research.

2. Context: Architecting software for quantum computing

This section contextualises quantum computing systems in terms of their building blocks, i.e., (a) quantum hardware, (b) quantum software, and (c) quantum software architecture as shown in Fig. 1. More specifically, Fig. 1 provides a visual reference that correlates the Qubits and Qugates to quantum source code, representing design and implementation phase of QSE lifecycle. Software architectural components and connectors provide a blue-print to implement the quantum source code. We use the illustrations in Fig. 1, elaborated below, to introduce fundamental concepts and terminologies that will be used throughout the paper.

2.1. Quantum computing systems

To gain strategic advantages of quantum information processing, technology giants, such as IBM, Google, Microsoft and governmental organisations are heavily investing in the research and development of quantum systems (Microsoft, 2021; Behera et al., 2019; Courtland, 2017; Goled, 2021). From the system's engineering perspective, as shown in Fig. 1, fundamental to quantum computing hardware is the concept of Qubit (quantum bit) that represents the most fundamental unit of quantum information processing (Zeilinger, 1999; Gay, 2006). Contrary to the classical bit (binary digit) that is expressed as [1, 0] in digital computing systems, a Qubit represents a two-state quantum computer and these two states are specified as $|0\rangle$ and $|1\rangle$. The combinations of bits represent flow of digital information that alters the state of binary logic gates (on: 0 off: 1) to make digital systems work. Analogous to the binary gates, quantum gate (a.k.a. the quantum logic) represents the building blocks of a quantum circuit and transits its state via Qubit (Gay, 2006) as in Eq. (1). A Qubit can be in a state $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ or (unlike a classical bit) in a linear combination of both states.

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

In Fig. 1(a), we illustrate and elaborate on the distinction between a Bit and Qubit. A Bit is like a gate in an electronic circuit that can be either on or off, whereas a Qubit uses the unique properties of quantum mechanics to provide a unit that can be one or zero- or anything in between. The bit can take a value of '0' or '1' as either 'Off' or 'On' with 100% probability (left). A qubit can be in a state of $|0\rangle$ or $|1$ or in a superposition state with 50% $|0\rangle$ and 50% $|1\rangle$, superposition state (left). Two Qubits are in an entangled state (right) - entangled qubits are linked such that by looking (i.e., measuring) one of these two, will reveal the state of other Qubit. Further details about Qubit and Qugate in the context of operationalising the QC systems can be found in Zhao (2020), Zeilinger (1999). Like the classical computing systems, controlling the Qubits that manipulate Qugates, there is a need for quantum software systems and applications that can exploit benefits of quantum information processing by operationalising quantum computers. For example, QuNetSim (DiAdamo et al., 2021) is a Python software framework that is capable of managing quantum circuits to simulate processing and transmission of quantum information via quantum networks. Fig. 1 shows that in order to enable quantum software applications to utilise quantum hardware, there is a need for quantum code compilers that can translate high-level computational instructions into machine translated code to control quantum hardware (Chong et al., 2017; Sunita et al., 2021). As a typical example of such compilation are the solutions by proposed by Ying (2016) and, Krüger and Mauerer (2020), which receive the compiled code that can be executed or simulated on quantum platforms to enable quantum processing for optimising solutions regarding unstructured data searching, parallel processing, and nature inspired computing. In recent years, a plethora of research and development has emerged that focused on quantum algorithms and programming languages to address the above-mentioned computational challenges effectively and efficiently (Sunita et al., 2021). Quantum algorithms have the potential to provide computation efficiency to software engineering problems in areas including but not limited to data mining, machine learning, and cryptography that do not scale optimally on non-quantum computing platforms (Miranskyy et al., 2022). Despite the significance of quantum programming languages to produce executable specifications for quantum hardware; there is a need for overall engineering lifecycle(s) that goes beyond level of source code to specify, execute, validate, and evolve software-intensive system based on required functionality and desired quality (Moguel et al., 2020; Piattini et al., 2021).

2.2. Software Engineering (SE) for quantum computing

Software engineering, as defined in the ISO/IEC/IEEE 90003: 2018 standard aims to apply engineering principles and practices to design, develop, validate, deploy, and evolve software-intensive systems effectively (ISO/IEC/IE.E.E. 90003:2018, 2021). In recent years, SE focused research and development started to tackle, such as quantum software models, their algorithmic specifications, and simulated evaluations to leverage benefits of quantum hardware for quantum information processing (Montanaro, 2016; Grimsley et al., 2019; Rebentrost et al., 2014; Childs et al., 2018; Svore et al., 2006). More specifically, software engineers can leverage SE practices and patterns by following software process(es) that comprises of a multitude of engineering activities including but not limited to requirements engineering, design, implementation, evaluation, and deployment, as shown in Fig. 1. SE activities adopted from quantum and classical software engineering concepts are used to represent a simplified view of quantum SE process (see Fig. 1 (b)) (Zhao, 2020; Dorfman and Thayer, 1997). Such generalised process can be tailored (adding, removing, and/or customising any activities) as per the context of system development.

Quantum computing systems are in a phase of continuous evolution and consequently quantum SE represents a new generation of software-intensive engineering activities to develop applications that can control the underlying hardware (Piattini et al., 2021). In recent years, research communities on software engineering and software architecture have focused on establishing dedicated forums, i.e., conferences, workshops and alike forums in an attempt to set the agenda(s), streamline emergent challenges, and propose community wide initiatives to engineer and architect quantum software (Abreu et al., 2021a; Barzen et al., 2021). These QSE focused research communities intend to gather researchers and practitioners and provide a forum to collaborate and explore the possibilities to exploit existing software engineering methods that can be applied to quantum era computing and software systems (Abreu et al., 2021b; Ali et al., 2022). The Quantum Flagship represents a prime example to support sustainable research and development for consolidating and expanding scientific leadership, achieving excellence and innovation in quantum computing technologies (Anon, 2022a). QSE process may involve an additional challenge of managing hybrid applications and algorithms. A hybrid application and its underlying implementation involve splitting the overall application into classical modules (pre/post-processing) and quantum modules (quantum computation) referred to as the quantum-classic split (Weder et al., 2022), as shown in Fig. 1(b). Research

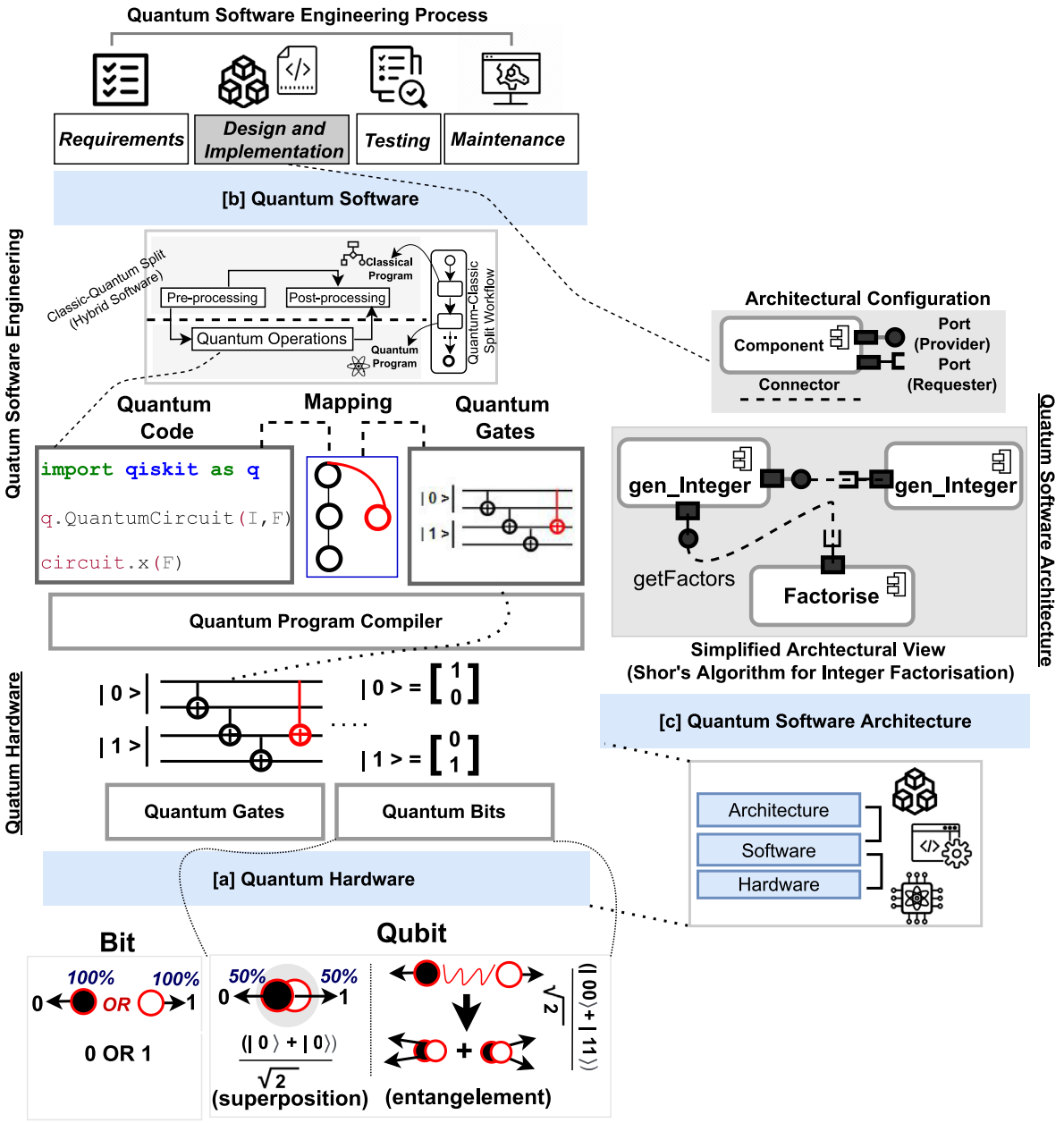

Fig. 1. A simplified view of quantum computing systems ((a) Quantum hardware, (b) Quantum software, (c) Quantum software architecture).

on the quantum-classic split is gaining attention with an aim to develop QSE process(es) that enable quantum software designers and developers to engineer hybrid applications by applying the quantum-classic split pattern (Pérez-Castillo and Piattini, 2022).

In addition to the needs for innovative technologies and processes, principle, and practices that specifically tackle challenges for quantum software modelling and architecting, coding, and simulation, existing classical SE processes can be customised to engineer and develop quantum software (Svore et al., 2006; Baczewski et al., 2017). For example, the concept of architectural modelling as a generic architecting activity, can be customised with initiatives like quantum UML profile, exploiting the UML activity diagrams that could help model parallel computing for quantum search algorithms (Pérez-Castillo et al., 2021). UML profiles for quantum systems enable software designers to create multiple views as different perspectives of system under design. For example, the designer can utilise the activity diagram to design quantum circuits (Pérez-Castillo et al., 2021) or utilise use case, sequence, or deployment diagrams to design the interaction, control flow, and configuration views of classical-quantum software (Pérez-Castillo and Piattini, 2022). Similarly, existing requirements engineering process can be tailored to support requirements for quantum (i.e., quantum entanglement) that is missing in the existing models. In SE process(es), architecting represents a pivotal activity that accumulates system requirements as a model thus leading to software implementation, validation, and evolution while maintaining a global view of the system and managing architectural trade-offs (Malavolta et al., 2012; Di Francesco et al., 2019).

2.3. Architecture for quantum software

Architecture of software intensive systems, as described in the ISO/IEC/IEEE 42010:2022 standard, aims to abstract complex and implementation specific details to represent system blueprint in an implementation and technology neutral way (Anon, 2022).

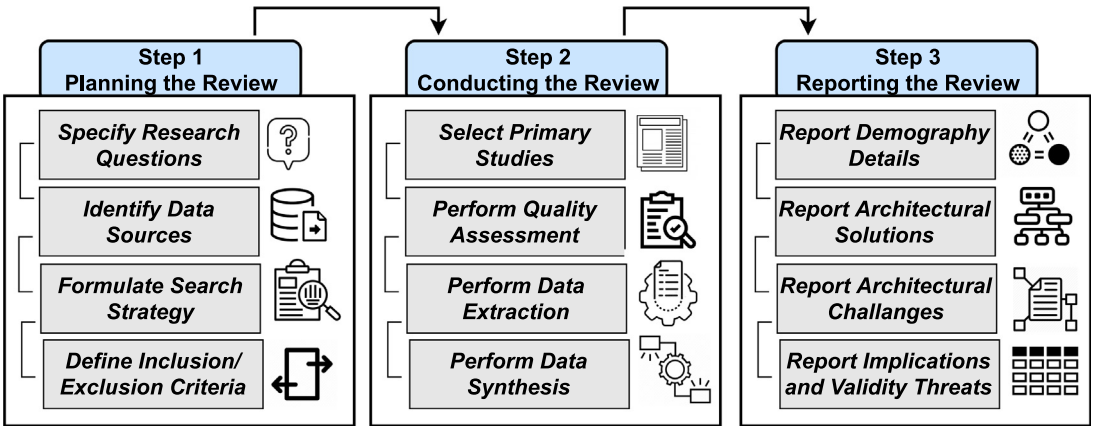

Fig. 2. An overview of the research methodology for SLR.

Empirically-grounded academic research and industrial studies on architecting software-intensive systems have highlighted that there is no unified view to represent software architectures (Hofmeister et al., 2007; Malavolta et al., 2012; Medvidovic and Taylor, 2000). Different architectural views (also referred to as architectural models or representations) can also be attributed to a multitude of modelling approaches supported via UML, ADL, and graph models that allow software practitioners to create customised architectural view(s) that fits their context in a specific architecting activity (Pérez-Castillo and Piattini, 2022; Medvidovic et al., 2002). For example, considering the 4 + 1 architectural view (Hofmeister et al., 2007), requirements engineers may be more interested in the interaction model(s) expressed as graphs or UML use case diagrams that capture architecturally significant requirements (functionality and quality of system). In comparison, software developers and quality engineers/testers are more likely to utilise the component and connector models that represent modules of source code and their interactions, and runtime view that models system execution as UML sequence diagrams. As per the 4 + 1 architectural view, in this study, we have mainly relied on the component and connector architecture model (Fig. 1) that represents software in terms of computations and data stores. However, during architectural review and synthesis, the component and connector architectural models alone are not sufficient and the effort to consolidate a singular or unified view that supports various architectural activities may be impractical. Once expressed, some architectural models, i.e., model driven architecture can help to generate the necessary skeleton or libraries of source code in a (semi-) automated way using model-driven engineering (Moin et al., 2021). In recent years, architectural models and notations have proven to be successful to design and develop software intensive systems by enabling reusability (patterns and styles), evolvability (architectural reconfigurations), and elasticity (auto-scaling) (Hofmeister et al., 2007; Li et al., 2013). Fig. 1(c) illustrates a partial architectural view of a quantum algorithm to factorise integers that is modelled as UML component diagram (Pérez-Castillo et al., 2021). The architectural view abstracts the source code level details to present design decisions in terms of components (Shor_Factor, Shor_Order) that coordinate via a connector (getFactors) for integer factorisation. Architecture in itself represents non-executable specifications of the quantum search system, however; the application of model-driven engineering can help architects and designers to derive source code directly from architecture models.

In the overall view of Fig. 1, we can conclude that in quantum computing systems, software architecture represents a blue-print to develop software systems and applications that manipulate quantum hardware. Quantum software projects primarily focused on producing quantum source code while overlooking quantum software design are often prone to bugs and unfulfilled requirements (Campos and Souto, 2021). The role of software architecture in quantum SE is pivotal to develop the requirements, which lead to software designing, coding, validation, and deployment, all facilitated using architectural notations. Software architecture for quantum computing systems (quantum software architecture) can empower the role of software engineers and developers to create models that act as basis for system implementation. Based on the architectural models, model driven engineering and development can be exploited for the automated generation of quantum source code (code modules and their interactions) from the corresponding quantum software architecture (based on architectural components and their connectors) (Ying, 2016; Moin et al., 2021; Shepard, 2021).

3. Research methodology

We followed EBSE approach to conduct this research (Pizard et al., 2021). As part of our research methodology, we adopted the Systematic Literature Review (SLR) approach to identify, analyse, and investigate the available literature based on the outlined research questions. Specifically, SLR follows the principle of evidence-based software engineering approach to adopt a rigorous process for conducting the review based on well-defined protocol to extract, analyse, and report the results (Kitchenham et al., 2004). SLR provides *"a means of evaluating and interpreting all available research relevant to a particular research question, topic area, or phenomenon of interest",keele2007guidelines*. We followed the guidelines provided by Kitchenham and Charters to conduct this SLR (Kitchenham and Charters, 2007), which consists of three core steps, i.e., *planning, conducting,* and *reporting* the review as illustrated in Fig. 2.

Each step of SLR, as illustrated in Fig. 2, is elaborated below. While performing the literary reviews and secondary studies in the context of software engineering research, there is an ongoing debate about conducting the Multivocal Literature Reviews (MLRs) – including grey literature – instead of SLRs for fast evolving areas like quantum computing and quantum software engineering (Garousi et al., 2019). We preferred the SLR, based on the guidelines in Kitchenham and Charters (2007), to only review peer-reviewed published research as secondary studies on quantum software architecting. Non-peer reviewed studies and grey literature are also discussed to discuss the results of primary studies, however; such studies and literature are complementary and are not included in the list of primary studies for SLR.

Table 1
Research questions of this SLR.

A: Demographic details of published research		
#	Research question	Rationale
RQ1.1	What are the types and the frequency of publications on quantum software architecture?	This RQ aims to pinpoint the types of publications (e.g., journal articles, conference proceedings) and highlight the frequency of publications (number of publications per year). The RQ provides an understanding of the research progress (i.e., type and frequency published research over the years) with respect to the topic under investigation.
RQ1.2	What are the research types and reported contributions in published studies on quantum software architecture?	Types of research (i.e., solution type, evaluation type) and research contributions help us to understand the diversity of published research, solutions to address the problems, empirical foundations, and theoretical principles as the available evidence in the SLR.
RQ1.3	What are the application domains to which the proposed architectural solutions can be applied?	Application domain refers to the areas (e.g., network security, system engineering) to which architectural solutions can be applied to address specific challenges. A classification of application domains help us understand the extent to which architectural solutions address software design challenges pertaining to different areas.
Architectural solutions for quantum software and emerging challenges		
#	Research question	Rationale
RQ2.1	Are there any architectural processes for quantum software?	Architectural process include a number of architecting activities to provide a step-wise and incremental approach to develop architectural solutions. By investigating the architectural process and its underlying architecting activities, we can understand architectural analysis, synthesis, and evaluation of proposed solutions.
RQ2.2	What modelling notations have been used to represent quantum software architectural solutions?	Modelling notations visually depict the detail sequence of architecting activities and show the relations between the numerous units of the software system. Answer to this RQ will give an understanding of existing graphical notation used to specify quantum software architecture.
RQ2.3	What patterns exist for quantum software architectures?	Patterns represent reusable knowledge and best practices to design and implement software solutions. The answer to this RQ will help to investigate the patterns which reveal reusable (architectural) knowledge and best practices to architect quantum software systems.
RQ2.4	Are there any tools and/or frameworks to support automation and customisation of architectural solutions for quantum software?	To study the available tools and framework support that can enable automation and customisation (i.e., user decision support) of the architectural process and its activities. We aim to further analyse tools that complement the architectural solutions with their automation and customisation.
RQ2.5	What challenges have been reported for quantum software architecture?	Various challenges could impact the process of developing quantum software architecture. Analysing the challenges will pinpoint the issues and factors that impact architectural solutions for quantum software.

3.1. Planning the review

As the initial step, the planning phase starts with developing the research questions that encapsulate the key research objectives of the SLR.

3.1.1. Step 1: Specify research questions

We outline the Research Questions (RQs) to investigate multi-faceted information including *demography, architectural activities, architectural modelling notations, architectural design patterns, tools and frameworks, and challenges*. The RQs to investigate the mentioned multi-faceted information are outlined and the details along rationale of each RQ is provided in Table 1. Answer to the reported RQs helps us document the SLR results described in subsequent sections of this paper.

3.1.2. Step 2: Identify data sources

In systematic reviews and mapping studies, Electronic Data Sources (EDS) allow an automated search, based on predefined and often customised search string(s), to identify the relevant literature on a topic under investigation (Chen et al., 2010). A number of empirical studies have investigated methods for conducting systematic searches along with putting forward a list of EDS that can help select literature efficiently while minimising the potential bias and risk of missing relevant data (Mourão et al., 2017). Based on the recommendations for adopting a systematic search process and selecting the most relevant data source, we selected five EDS for an automated search (Zhang et al., 2011). These EDS include ACM Digital Library, IEEE Xplore, Science Direct, SpringerLink, and Wiley Online Library that represent prominent sources to search literature on computing in general and software engineering and software architecture research in particular. The list of EDS that we selected is not an exhaustive, nor does it guarantee to cover all possible existing literature, however, prior empirically-based studies on SLRs have highlighted these five electronic sources as necessarily sufficient and appropriate to identify the relevant literature (Chen et al., 2010; Zhang et al., 2011).

3.1.3. Step 3: Formulate search strategy

The first three authors analysed the RQs to identify the key terms or keywords. Moreover, all the authors were invited to participate in the group meeting to finalise the key terms. The aim of reporting the key research terms is to develop the search string and explore the selected digital libraries using that string. Finally, the authors agreed to consider the following search string for the data search: *(Software) AND (Architecture OR Design OR Framework OR Pattern) AND (Quantum)*

The key terms are concatenated using the *"OR"* and *"AND"* boolean operators to develop the above-given search string. The decision to finalise the search string was based on a pilot search of relevant literature on IEEE eXplore and Google Scholar. In the pilot search, we aimed at identifying the titles of existing studies and various synonyms used to refer to software architecture in the quantum computing context. For example, we observed that use of key term *'model'* as a synonym for architecture yielded a

Table 2
Inclusion and exclusion criteria.

Code	Inclusion criteria	Code	Exclusion criteria
Inc1	Studies that specifically focus on software architecture studies in quantum computing domain	Excl1	Exclude grey literature and duplicate studies.
Inc2	Peer-reviewed published research (e.g., conference proceedings, journal articles, workshop/symposium papers	Excl2	If multiple studies are published in the same project, then consider the one with maximum contribution.
Inc3	Peer-reviewed studies available in full-text.	Excl3	Exclude studies that do not model or describe structure and/or behaviour of quantum software.
Inc4	Reported in English language.	Excl4	Exclude the studies that do not discuss any of the software architectural aspects as outlined in the RQs (e.g., process, patterns, notations, tools)

significantly large but irrelevant number of studies that discuss software process models (focused on QSE rather than QSA). Based on the consensus of the researchers, we omitted the key term *'model'* to avoid an exhaustive search space. Moreover, based on the pilot searching phase, we included key term *'framework'* that did identify some relevant studies. The main goal of the final search string was to identify the most relevant literature as much as possible while avoiding potentially irrelevant studies that can exhaust manual scanning of titles, keywords, and abstract for study selection. The replication package based on the given search string is provided in Khan et al. (2022a).

3.1.4. Step 4: Define inclusion and exclusion criteria

Based on the guidelines by Kitchenham and Charters (2007) for including or excluding the identified studies, we outlined the inclusion and exclusion criteria in Table 2. By following the criteria any irrelevant, redundant, or non-English studies were excluded. Study inclusion and exclusion was followed by a quality assessment step to assess the quality of each included study and eliminate any study that did not satisfy the qualitative assessment criteria (see Section 3.2.2). The inclusion and exclusion criteria filters the search findings returned by the search string. The key points of the criteria were developed by the first three authors based on Kitchenham and Charters (2007). Table 2 provides the criteria for the inclusion and exclusion of the literature for review along with the codes (Incl 1–4: as the inclusion criteria and Excl 1–4: as the exclusion criteria). We discuss the details in Table 2 later to elaborate the selection of primary studies to be included in the SLR.

3.2. Conducting the review

The second phase of the SLR process is conducting the review, which is based on the protocol defined in the first phase, i.e., *planning the review* (See Fig. 2). Following are the key steps involved in this phase:

3.2.1. Step-1: Select primary studies

Primary studies search process started with exploring the selected digital repositories using the search string discussed in Section 3.1.3. The search process was initiated on 30th September 2021 and ended on 9th October 2021. Initially, the search string returned a total of 8,406 studies, which are further filtered by the first three authors based on the studies titles, keywords, and abstracts against the inclusion and exclusion criteria (see Fig. 3). The second phase screening returned a total of 589 studies. The third phase of inclusion and exclusion screening was performed based on the full-text review of the studies, where 32 primary studies were finally selected (see Fig. 3). Additionally, the fourth and fifth authors were invited to confirm the search findings and list of selected studies.

For example, we used the advanced search option for IEEE Xplore ('Search Term') to execute the search string to identify published studies (in 'Full Text & MetaData). The search yielded a total of 32115 studies, majority of which focused on quantum systems in general and quantum hardware in particular. While trying to eliminate an exhaustive list of irrelevant studies, we interchanged the search parameter (from 'in Full Text & MetaData' to 'in Abstract') and found 397 studies that missed some relevant studies that were discovered before the search parameter interchange. Therefore, we decided to manually scan through the 32115 studies after we applied further digital library-specific filtering to eliminate search results classified under 'Standards', 'Books' and other alike categories to get a total of 1751 candidate studies from IEEE eXplore. Based on a similar approach, often digital library-specific filtering, we extracted and identified the candidate studies to proceed with their screening, inclusion/exclusion, and qualitative assessment, as in Fig. 3.

Moreover, the backward snowballing approach was used to manually search the references list of the selected 32 primary studies to identify additional studies that might have been missed during the search string-based review process (Wohlin, 2014). The backward snowballing eventually returned two more studies that explicitly fulfilled the inclusion and exclusion criteria. The snowballing process was mainly performed by the first and second authors. Additionally, the third and fourth authors were invited to mutually verify the findings reported by the first and second authors. To include studies that discuss software architecture, we specifically looked for architectural models (graphical notations, e.g., UML diagrams) or architectural specifications (descriptive notations, e.g., ADLs) that represent the structure or behaviour of software system (Medvidovic and Taylor, 2000; Medvidovic et al., 2002). Finally, (32+2) studies are shortlisted (see Fig. 3) to review, analyse and address the research questions based on their findings. The selected primary studies list is provided in Appendix (Table 11). Furthermore, We included several non-peer-reviewed studies available on the arXiv open-access repository (Zhao, 2020; Nguyen et al., 2022; Fahmideh et al., 2021a,b; Graef and Georgievski, 2021; Moin et al., 2021; Pérez-Castillo et al., 2021; Dey et al., 2020; Miranskyy et al., 2022; Wang et al., 2022; Paulo and de Camargo, 2021; Khan et al., 2022b) to complement the study's overall findings. However, we did not include them in the primary studies list (Appendix – Table 11) as per the guidelines of SLR (Kitchenham et al., 2004) approach. In addition to following the guidelines of the SLRs for literature inclusion (Kitchenham et al., 2004), our decision was also motivated by the fact that preprints are often subject to changes overtime, with several versions having the same title but differing content. Given the fast-paced research fields of quantum software engineering/architecture, preprints may contain errors or changes that could compromise the reliability of our SLR's results and threaten its internal validity. Therefore, we excluded preprints from our list of selected primary studies to minimise the

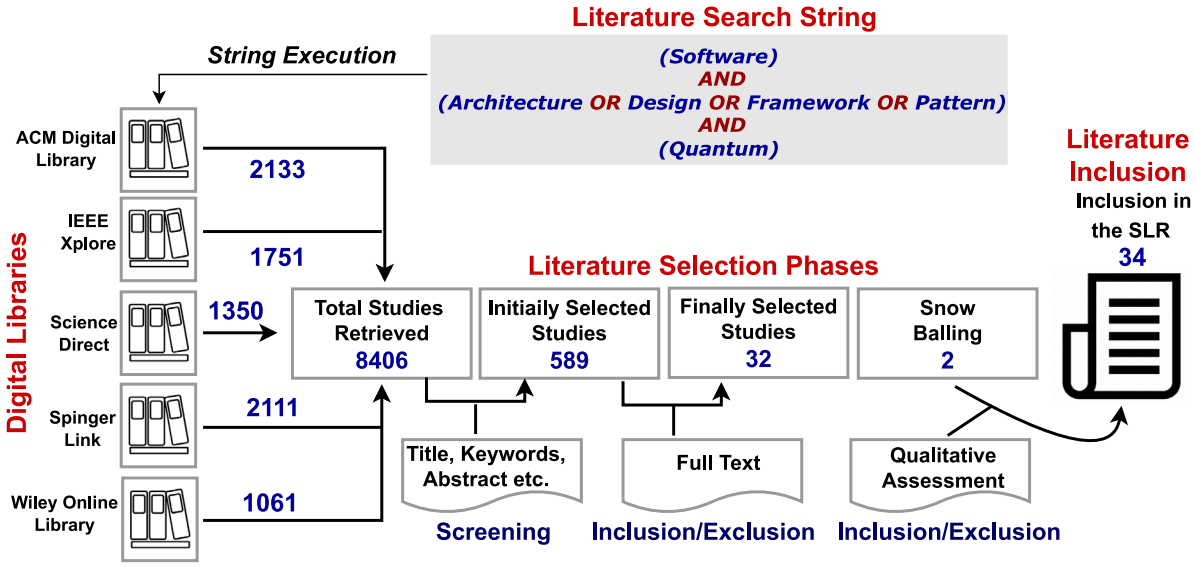

Fig. 3. Studies selection process.

mentioned risk. For instance, one preprint changed (Abbott et al., 2018) its content four times within two years, highlighting the need for caution when incorporating preprints into a systematic review.

In addition to our adopted approach for automated search in electronic databases and backward snowballing to identify the relevant studies, several other approaches could be used. Some of these approaches include but are not limited to searching individual publication venues (e.g., conference proceedings, journal volumes), research group publications, and forward snowballing (Felizardo et al., 2016). Specifically, forward snowballing – searching for studies that cite the studies contained in the seed set – is found to be more useful in updating or extending an already conducted secondary study but is still prone to missing relevant literature. Jalali and Wohlin (2012) investigates the application of snowballing approaches in SLRs and suggests that similarity in identified literature is expected to increase if both the backward and forward snowballing are performed since the overlap in the included papers would be greater. This influenced our decision to avoid forward snowballing, however; future extensions of this SLR can benefit from forward snowballing with an updated seed list of studies (Felizardo et al., 2016).

3.2.2. Step-2: Perform quality assessment (QAs)

The quality of the selected studies is evaluated based on the quality assessment criteria that aim to remove the research bias and evaluate the degree of significance and completeness of the selected studies (Kitchenham and Charters, 2007). The quality assessment guidelines provided by Kitchenham and Charters are followed to develop the assessment criteria (see Table 3) (Kitchenham and Charters, 2007). The criteria consist of five assessment questions, and each selected primary study assessed against these questions *(QAs1-QAs5)*. Assigned score (1) if the primary study explicitly addressed the QAs questions and (0.5) points if the questions are partially addressed. Similarly, studies with no evidence of considering the assessment questions are given 0 point. The final quality assessment score for each primary study is the sum of the score assigned against each QAs question. The first author applied the assessment criteria and the results were further independently verified by second and third authors. We include those studies in the final list which had accumulative QAs score greater than or equal to 1.5 (Waseem et al., 2020). The accumulative final score of each primary study against the QAs questions is given in Appendix (Table 11).

3.2.3. Step-3: Perform data extraction

We defined a set of data extraction items (see Table 4) to address the RQs formulated in Section 3.1.1. Data items are the particular types of data extracted from each selected primary study that directly map to the study RQs. The first author performed the pilot data extraction process for ten studies to evaluate the reliability of the extracted data items. The second and third author assessed the pilot study findings, and based on their suggestions, the first author revised the data extraction items. The formal data extraction process was performed by the first three authors by equally distributing the total number of selected primary studies, and the studies distribution was done based on the authors' research expertise and interest. The general (demographic) details of each selected primary study were extracted against the data items (DI1-DI4), and the rest (DI5-DI13) are specific to the study RQs.

We finally conducted the Cohen's Kappa test to check inter-personal bias in the primary studies selection (Section 3.2.1), quality assessment (Section 3.2.2), and data extraction (Section 3.2.3) phases. Mainly, the first three authors were involved in the studies selection, quality assessment, and data extraction process. To remove the inter-personal bias for the mentioned phases of the SLR process, we invited the remaining authors and merged them across two different groups (authors 4–5, authors 6–7). They were asked to randomly select a set of ten primary studies and sequentially perform the studies selection, quality assessment, and data extraction process as performed by authors 1–3. Eventually, the Cohen's Kappa test was performed to measure the agreement level and identify the significant differences across the mentioned phases between all the three groups of authors (authors 1–3, authors 4–5, authors 6–7). The Cohen's Kappa test is widely adopted in EBSE research (Pérez et al., 2020). Cohen's kappa coefficient (k) is *the proportion of chance-expected disagreements which do not occur, or alternatively, it is the proportion of agreement after chance the agreement is removed from consideration,cohen1960coefficient*. The (k) coefficient measures the level of agreement between a group of raters that evaluate N-objects into (c) mutually exclusive categories (Cohen, 1960). The agreement level between the raters equals chance agreement when Cohen's kappa coefficient value (k)=0. The level of agreement is positive when (k) is greater than the chance agreement and negative if it is less than it. The perfect agreement occurs between a group of raters when the value of k ranges from

Table 3
Studies quality assessment criteria.

Code	Quality assessment questions	Score
QAs1	Do the research objectives of the study are explicitly defined?	(1/0.5/0)
QAs2	Does the adopted research methodology is clearly discussed?	(1/0.5/0)
QAs3	Do the experimental settings are explicitly reported?	(1/0.5/0)
QAs4	Do the results and findings are thoroughly discussed?	(1/0.5/0)
QAs5	Do the real-world implications of the study are reported?	(1/0.5/0)

Table 4
Relevant data items extracted from the selected primary studies.

Code	Data item	Description	Related RQ
QI1	Index	The study ID	Demographic
QI2	Study title	Full title of primary study	Demographic
QI3	List of authors	Authors full names	Demographic
QI4	Publication's venue	Name of the Journal, Conference, Workshop, Book, symposium, Magazine	Demographic
QI5	Publication's year	Temporal information of each study.	RQ 1.1
QI6	Publication type	Journal, Conference, Workshop, Book chapter, Magazine	RQ1.1
QI7	Research type	Studies mapping across research facets	RQ1.2
QI8	Research domain	Develop themes and sub-themes of studies research focus across different domains	RQ1.3
QI9	Architectural activities	Key activities to define quantum software architecture process	RQ2.1
QI10	QSA modelling notations	The existing modelling notations to structure quantum software architecture	RQ2.2
QI11	QSA patterns	Identify the patterns for quantum software architectural design problems	RQ2.3
QI12	Architectural tools and frameworks	The tools discussed in the primary studies to support architecting activities	RQ2.4
QI13	QSA challenges	The challenges reported to develop quantum software and system architecture	RQ2.5

(0.81 to +1.00). The interpretation of the k-value to measure the strength of agreement is adopted from the observer agreement study conducted by Landis and Koch (1977)

We used R-3.6.3 to conduct the (k) test for interpreting the agreement level between the groups of raters (authors). The R-code (see Appendix (Table 12)) was executed and obtained the Cohen's Kappa coefficient value (k= 0.62), which shows a positive and substantial agreement level (Landis and Koch, 1977) between all the authors for the primary studies selection, quality assessment, and data extraction phases. Based on the test results, we concluded that no personal bias exists between the authors that could significantly impact the core SLR phases.

Similarly, the Cohen's Kappa test was performed to evaluate the interpersonal bias between the authors for the snowballing process. Mainly authors 1–4 performed the snowballing process (Section 3.2.1), however, authors 5–6 were invited to participate in the Cohen's Kappa test to assess inter-personal bias. Both groups (authors 1–4, authors 5–6) selected the first five primary studies (S1 to S5) and performed the snowballing search. The Cohen's kappa coefficient ($k = 0.50$) is calculated based on the search findings of both groups using the R-code given in Appendix (Table 12). The given value of (k) reveals that both groups of authors have an unbiased, positive, and moderate level of agreement for the snowballing process.

3.2.4. Step-4: Perform data synthesis

Data items (DI1-DI4) were analysed using the descriptive statistical approach. Similarly, we generated initial codes for the data items (DI7, DI8, DI12 and DI13) to define the research themes and address RQ1.2, RQ1.3, RQ2.4 and RQ2.5. Thematic analysis guidelines for qualitative data provided by Braun and Clarke are considered to systematically analyse, organise, and develop themes across the extracted data (Braun and Clarke, 2006). In line with the outlined RQs 1, the following thematic data analysis steps are followed to develop the key themes of extracted data items:

1. **Data familiarisation**: The first three authors thoroughly read the selected primary studies and noted the data items given in Table 4.
2. **Generating the initial codes**: The initial codes from the extracted data are generated to define the research themes for RQ1.2, RQ1.3, RQ2.4, and RQ2.5.
3. **Searching for themes**: The codes define in the previous step are analysed and encapsulated across broader themes.
4. **Reviewing themes**: The first three authors examined the themes to separate, drop and merge based on the mutual discussion and understanding.
5. **Defining and naming themes**: The defined themes are characterised with precise names.
6. **Producing the report**: This step involves to refine the developed themes and their respective characteristics.

The thematic analysis process of this SLR is given in Fig. 4, and all the authors finally participated in the brainstorming session to remove bias in the thematic approach by defining and naming the key themes. To complement the methodological steps of this SLR, a replication package is provided that details the selected primary studies based on the customised search string, scoring of quality assessment of identified studies, and the extracted data for each individual RQs (Khan et al., 2022a).

3.3. Reporting the review

We reported the results of SLR, presented in dedicated sections, based on the categories of outlined RQs (see Table 1). Specifically, (i) *demography details* of published research (i.e., RQ1.1 to RQ1.3) are discussed in Section 4, and (ii) *architectural solutions and challenges* (i.e., RQ2.1 to RQ2.5) are detailed in Section 5. The analysis of the SLR and summary of key results are detailed in Section 6.

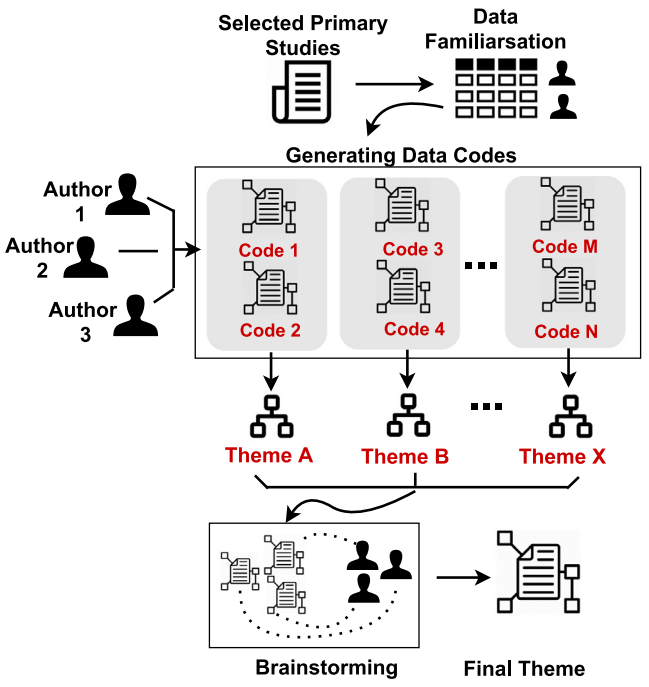

Fig. 4. Thematic analysis process.

4. Demography details of published research

In this section, we answer RQ1, having three sub-questions, i.e., RQ1.1 - RQ1.3 that rely on mapping analysis to present demography details of published research (Petersen et al., 2015). Specifically, within the SLR, we performed systematic mapping to investigate demography details of published research focus on (i) types and frequency of publications (RQ1.1: Section 4.1), (ii) types and contributions of research studies (RQ1.2: Section 4.2), and (iii) application domains of architectural solutions (RQ1.3: Section 4.3), all detailed below. The demography details complement the presentation of overall results and discussion of proposed architectural solutions. For example, the types of research studies (answering RQ1.2) discussed here indicate a multitude of research contributions, such as solution proposals, validation research, and/or philosophical studies, and their roles in deriving the architecting activities for quantum software.

4.1. Types and frequency of publications (RQ1.1)

It is significant to classify the selected primary studies based on their frequency and type of publications. This analysis highlights the research trend of a particular research area and the research community's interest. The frequency indicates how frequent is the occurrence of publications over the years, whereas the types refer to a specific type of publications (e.g., a journal article) as illustrated in Fig. 5. The total number of published studies are presented across (Y-axis) and their year of publication across (X-axis). Moreover, Fig. 5 is a bar graph that relatively highlights different publication types, i.e., conference proceedings, journal articles, symposium papers, and workshop articles. The initial study was published in 2004 and final in October 2021. The bar graph reveals that a total 21 (62%) of the selected studies were published in the last four years (from 2018 to October 2021), which is an interesting finding that interprets the significance of quantum software architecture in present-day quantum computing research. It reveals that the research community are significantly working on designing architectural solutions for quantum software systems. Moreover, 15 (44%) primary studies are published in journals, 13 (38%) in conference proceedings, 4 (12%) workshop papers and 2 (6%) symposium article. A report by Scopus in 2021 highlights recent research trends on quantum computing, reflected via Scopus-indexed documents, in terms of demography details of published research. The report provides a multi-faceted overview of published research in terms of frequency, types, top institutes, and top contributors in regard to their research on quantum computing (Anon, 2022b).

Key Findings of RQ1.1

Finding 1: Maximum number of primary studies ($n = 21$, 62%) are published from 2018 to 2021. It exhibits that quantum software architecture is emerging research area and got significant attention of research community.
Finding 2: Regarding publications type, the given results underline that journals ($n = 15$, 44%) and conferences ($n = 13$, 38%) are the popular venues to publish the relevant studies.

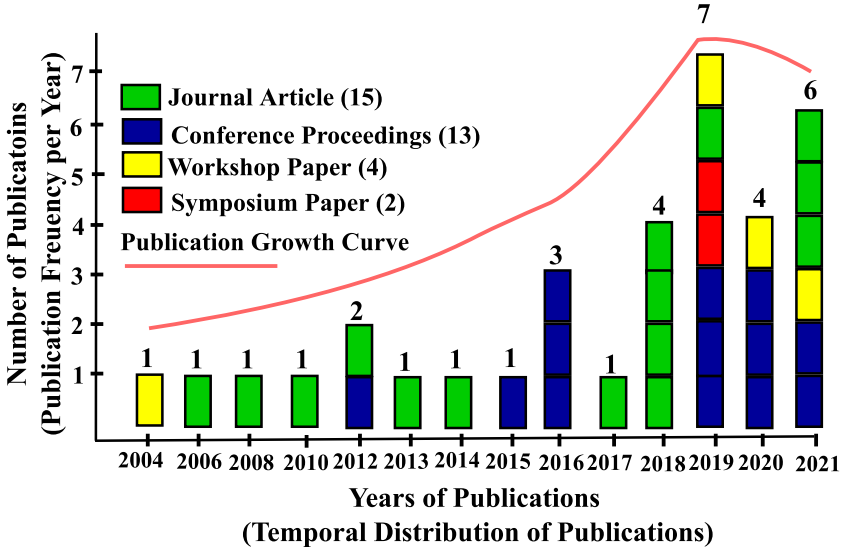

Fig. 5. Overview of frequency and types of publications.

4.2. Types of research and contributions (RQ1.2)

The selected publications are categorised based on the following six well-established research types proposed by Wieringa et al. (2006): *evaluation research*, *proposal of solution*, *validation research*, *philosophical papers*, *opinion papers*, and *personal experience papers*. *Evaluation research* is conducted to evaluate a specific problem or solution in practice using different empirical research techniques. *Proposal of solution* articles develop a method or solution for a relevant problem without fully validating its significance. *Validation research* is conducted to evaluate the quality attributes of the proposed solution, which has not yet been deployed in a real-world environment. *Philosophical papers* focus on architecting theoretical or conceptual frameworks. *Opinion papers* discussed authors' negative or positive opinions regarding a specific framework, model, a solution. In *Personal experience papers*, the authors report their personal experiences regarding a particular project or group of it. Additionally, we reported the research contribution of each paper classified across the mentioned research types.

Thematic analysis process discussed in Section 3.2.4 is followed to address RQ1.2 and classify the selected 34 primary studies across the given research types (see Fig. 6(a)). The set of selected studies consist of (n = 8, 24%) proposal of solution, (n = 3, 9%) personal experience papers, (n = 2, 6%) philosophical papers and (n = 1, 3%) opinion papers. Moreover, we identified (n = 20, 59%) studies that cover both proposal of solution and validation research categories. These studies are classified in a separate category (i.e., *proposal of solution and validation research*) (see Fig. 6(a)). We did not identify any paper that fits in the evaluation research category; therefore, it is excluded from the mapping process.

The results given in Fig. 6(a) reveal that majority of the studies, i.e., (n = 20, 59%) are mapped in the heterogeneous (i.e., *proposal of solution and validation research*) category. It means that the selected studies proposed their own solutions and conducted sample implementation to validate the significance of those solutions. It is aligned with the fact that quantum software architecture is a new paradigm, and there is a demanding need of novel architecture solutions. The second most common category is *proposal of solution* (n = 8, 23%), where various architectural solutions are proposed. However, the proposed solutions are not evaluated or validated both empirically and in real-world practice. For example, software architecture is proposed in [S1][1] to set up an ecosystem for quantum key distribution (QKD) in quantum networks. The architecture is build using a set of modules i.e., QKD module, relay modules, and QKD node. However, the proposed solution is not validated or evaluated to assess its real-world implications and contributions. Three (n = 3, 9%) primary studies are mapped into the *personal experience papers* category [S21, S31, S34]. For example, Leymann et al. [S31] reported an understanding of the architectural model to support business processes for developing and sharing the quantum software systems. The *philosophical papers* category covers two (n = 2, 6%) primary studies [S26, S30]. For instance, Nallamothula proposed a theoretical decision-making framework for quantum software architecture selection [S26]. The proposed framework has not been evaluated experimentally or in real-world practice. One single study (n = 1, 3%) is categorised as *opinion paper*, where the authors shared the opinion of quantum and classical co-design architecture [S20]. Regarding overall contribution, we noticed that 9 studies focused on quantum–classical intersection (i.e., *co-design of quantum systems*), where both classical and quantum techniques used to develop the quantum software architecture (see Fig. 6(a)) [S3, S20, S21, S16, S17, S18, S23, S24, S28]. It is a known fact that quantum software development is not a well establish field. Presently, its not possible to entirely develop a quantum software architecture based on quantum computing concepts. We still need to consider the classical software development concepts and techniques, at least at the interface level, to structure a quantum software system.

Key Findings of RQ1.2

Finding 3: Analysing the types of published research highlight that the combination of solution proposals and validation research represent most frequent publications. A total of 20 studies (i.e., n = 20, 59% approx) represent this category to propose architectural solutions and validate quantum software solutions via simulation or case studies.

Finding 4: Thematic classification of the research contributions highlights that (n = 9, 26% approx) studies focused on proposing architectural solutions for *co-design of quantum systems*. Quantum system co-design refers to mapping between classical and quantum concepts during the development of quantum computing systems. It means that most of the studies focused on developing quantum software systems using both classical and quantum computing concepts.

4.3. Classification of application domains (RQ1.3)

Thematic process defined in Section 3.2.4 is followed to categorise the selected primary studies based on the common application domains. Systematic identification, categorisation, and naming process of identified themes and sub-themes are given in Fig. 6(b).

We collected at least two or more studies of common application domains and encapsulate them under a single umbrella called theme. In this study, the following five core themes are identified and the selected studies are classified across them: (i) *systems and hardware engineering (n = 20, 59%)*, (ii) *software engineering (n = 5, 15%)*, (iii) *smart systems (n = 1, 3%)*, (iv) *source code compilation (n = 4, 12%)*, and (v) *network security (n = 4, 12%)*. In sub-thematic classification, we further categorised the main themes into more specific topics. Sub-themes are secondary to core themes, where the overall application domain (core theme) is classified more narrow (sub-themes). For example, the core theme (*systems and hardware engineering*) is classified across three distinct sub-themes including *co-design of quantum systems (n = 9, 26%)*, *optimisation of quantum processor (n = 6, 18%)*, and *software control for quantum hardware (n = 5, 15%)*. Similarly, *software engineering* is sub-classified into *quantum software development (n = 2, 6%)* and *quantum software modelling (n = 3, 9%)*. *Source code compilation* has two sub-themes: *compiling quantum source code (n = 3, 9%)* and *quantum code optimisation (n = 1, 3%)*. Moreover, the classification given in Fig. 6(b) illustrates that *network security* is further categorised across sub-themes *quantum key transmission (n = 2, 6%)*, *quantum information networking (n = 1, 3%)*, and *simulating quantum information flow (n = 1, 3%)*. Finally, the core theme *smart systems* has only one sub-theme i.e., *quantum game theory for smart classroom (n = 1, 3%)*.

Fig. 6(b) provides the high level categorisation of the existing quantum software architectural solutions with respect to different application domains. For instance, *systems and hardware*

[1] Please note, the notation [S_n], where n represents a numerical value (range: 1 to 34) to indicate a reference to the selected primary studies for SLR, listed in Appendix (Table 11). This notation also help to distinguish the selected primary studies from references in the bibliography section of this paper.

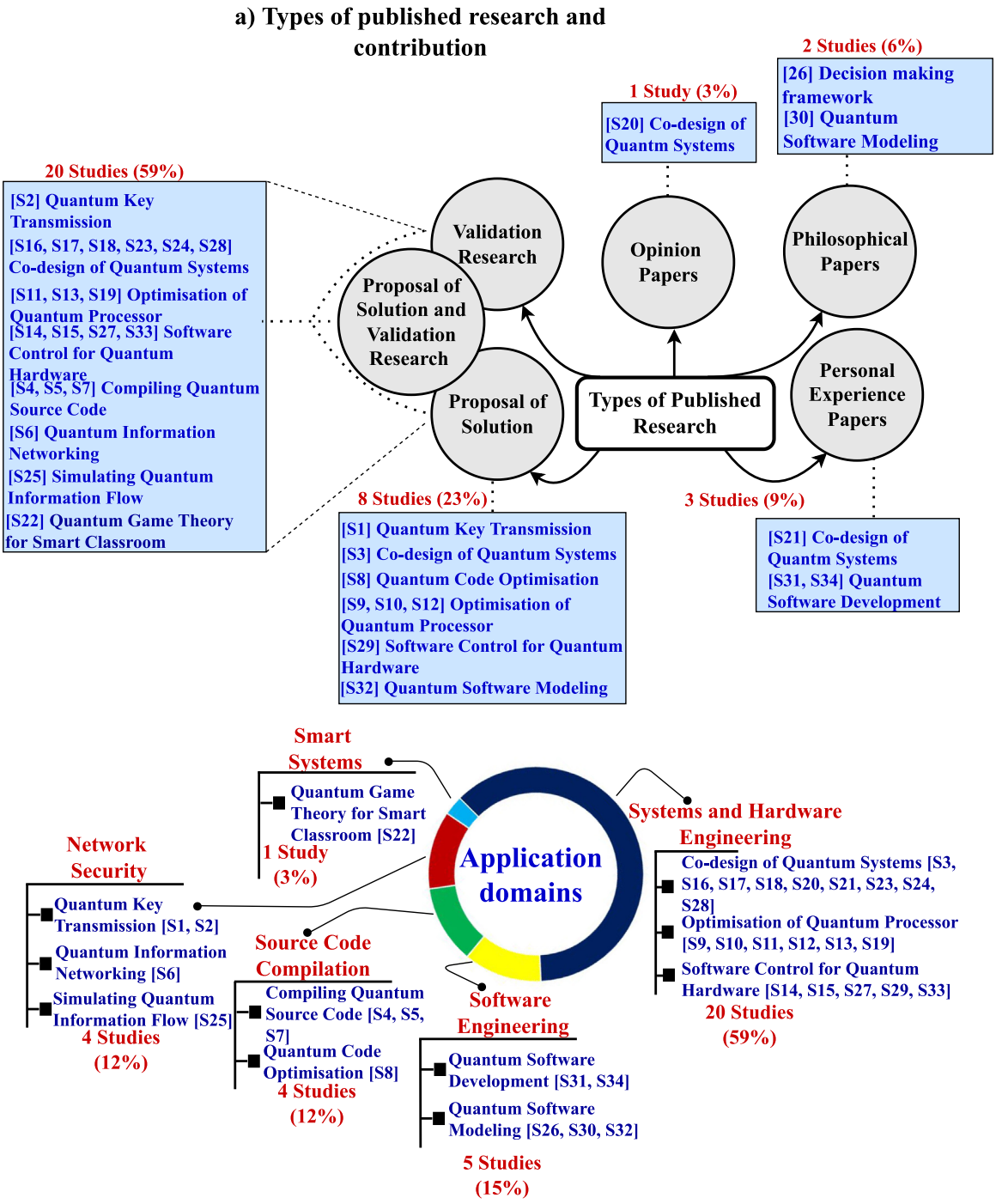

Fig. 6. Overview of types of research and application domains.

engineering is the most common and explicitly explored application domain with twenty research studies. It is aligned with the fact that the research focus on quantum software development is heating up (Zhao, 2020). Technology giants e.g., Google, Alibaba, and IBM are marching forward to propose advance architectural solutions to take the lead in quantum software technologies (Lapedus, 2021). Similarly, *co-design of quantum systems* is a sub-theme of *systems and hardware engineering*, which has a total of nine studies. It highlights the significance of quantum–classical hybridisation (Everitt et al., 2016). Quantum–classical collaborative relationship will have significant impact on quantum software architecture in the near-term (Baczewski et al., 2017). It will improve the architectural efficiency and meet the require performance.

In summary, Fig. 6(b) provides a holistic overview of studies mapping with respect to the application domains. It enables different interpretations of published studies based on core research themes and sub-themes. The given mapping provides a taxonomical understanding of state of the art application domains.

Key Findings of RQ1.3

Finding 5: The core application domains are: *systems and hardware engineering , software engineering , smart systems, source code compilation,* and *network security*. The selected primary studies are categorised across the mentioned domains.

Finding 6: *Systems and hardware engineering* (n = 20, 59%) is identified as the most common application domain. It reveals the fact that research community significantly focuses on presenting architectural solutions for quantum system and hardware problems. The reason might be that the existing classical system engineering approaches are not able to explicitly encompass the attributes of quantum physics (Everitt et al., 2016). There is a need of novel system and hardware engineering frameworks that tackle the quantum interface problems.

5. Architecture-centric solutions for quantum software

We now discuss architecture-centric solutions and emerging challenges, answering RQ2.1 to RQ2.5, that highlight some of the core aspects of designing and implementing quantum software. Specifically, (i) we present architectural process and its underlying activities (RQ2.1: Section 5.1), (ii) architectural modelling notations (RQ2.2: Section 5.2), (iii) architectural patterns and design decisions (RQ2.3: Section 5.3), (iv) tools and frameworks (RQ2.4: Section 5.4), and challenges of quantum software architecture (RQ2.5: Section 5.5).

5.1. Architectural process and activities (RQ2.1)

We now answer RQ2.1 that aims to investigate the existing process(es) that can support a process-centred – incremental and structured – approach to architect quantum software systems (Hofmeister et al., 2007). Specifically, an architectural process comprises of a collection of activities (a.k.a. architecting activities to support analysis, synthesis, and evaluation of the architecture for quantum software systems and applications (Li et al., 2013; Svore et al., 2006). During system design and implementation phase, architectural process streamlines *what* needs to be done and provides an umbrella to accumulate a collection of architecting activities that demonstrate *how* it is to be done. For example, in an architectural process, the activity called architectural requirements aims to analyse and outline the design challenges/issues that a particular architecture must resolve. The outcome of architectural analysis activity is a set of architecturally significant requirements (ASRs) to highlight the needed functionality and desired quality of the system under design (Li et al., 2013). For example, as in Fig. 7, as part of architectural requirements one of the ASR is: *how to effectively and securely transmit quantum information over quantum network?* The ASR outlines a design challenge that must be addressed by designing the appropriate architecture that supports transmission of quantum information (i.e., required functionality) over quantum network in an efficient and secure manner (i.e., desired quality attribute). The relevant studies, as an evidence, that support architectural process are indicated in Fig. 7. For example, Fig. 7 shows that two studies specify the requirements of an reference architecture, as a software blueprint, to generate quantum source code [S14, S27].

From quantum software engineering perspective, existing architectural processes represent a concentrated knowledge and wisdom (derived from architects' experiences, industrial practices, and academic solutions that can be attuned to architectural challenges for quantum genre of software systems) (Zhao, 2020; Hofmeister et al., 2007; Malavolta et al., 2012; Di Francesco et al., 2019). However, architecting quantum systems entail some specific challenges that cannot be effectively addressed by existing processes that have been designed for classical computing systems. Some of the quantum specific challenges include but are not limited to co-design, i.e., mapping quantum algorithms to Qubits of a Qugates, compiling hybrid source code into a unified quantum instruction set, and configuring simulators to simulate and execute quantum code (Svore et al., 2006; Leymann, 2019). This means that existing architectural processes need customised activities to address design challenges of quantum software.

To present the results, we followed available guidelines and empirically-based studies, grounded in industrial practices and academic research to document software architectures in terms of architectural processes and their underlying architecting activities (Hofmeister et al., 2007; Li et al., 2013; Malavolta et al., 2012; Di Francesco et al., 2019). We followed a generic process pattern derived from five industrial approaches to document architectural processes in terms of architectural design activities namely architectural analysis, architectural synthesis, and architectural evaluation (Hofmeister et al., 2007). The architectural process model proposed by Hofmeister et al. (2007) is incorporated by Tang et al. (2010) with two additional activities namely architectural implementation and architectural maintenance. Some industrial surveys, incorporating practitioners' perspective also highlight the needs for fine-grained representation, specifically in the context of architectural synthesis activity to effectively represent architectural solutions (Malavolta et al., 2012). To support a fine-granular representation of the architectural synthesis activity, we divided it into two distinct activities namely architectural modelling (representing ASRs as an architectural model) and architectural implementation (transform architectural model into specifications that can be executed or simulated). In the following, we detail the architectural process for quantum software, defined in terms of architecting activities, illustrated in Fig. 7 that also acts as a running example for demonstrative purposes. Fig. 7 provides a visual catalogue of the process and activities that are exemplified based on the available evidence from the reviewed literature. During the review, each study that corresponds to an architecting activity was identified, whereas Fig. 7 was constructed by synthesising the overall contributions from a collection of studies for their generic representation as a unified architectural process.

For example, as per Fig. 7, the selected studies help us to identify the **architectural requirements** to support efficient and secure transmission of quantum information over a quantum network [S1, S2]. The proposed **architectural model** as in Fig. 7 relies on a pipe and filter architectural pattern that supports generation, transmission, and reconciliation of a quantum key to secure quantum information that travels over quantum network (Garcia et al., 2021). To support the modelling, **architectural implementation** is enabled via components, representing computational units for source (transmitter) and target (receiver) nodes in the network that coordinate quantum information via component ports. A case study based approach is adopted for architectural validation in terms of efficiency and security of generating, transmitting, and reconciling the quantum key [S1, S2]. Peer to peer configuration of network nodes is adopted for **architectural deployment**.

(i) Architectural Requirements as the initial activity of the process aims at analysing, filtering, and/or reformulating architectural concerns to derive a set of architecturally

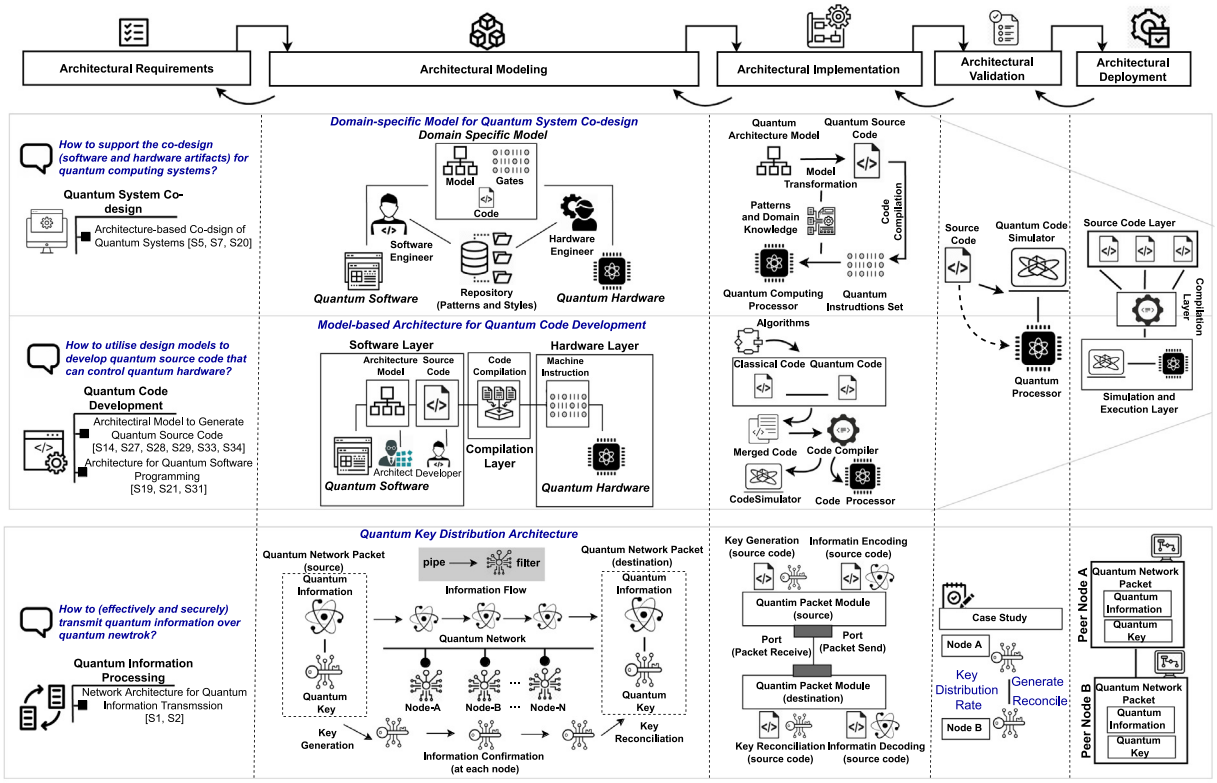

Fig. 7. Overview of quantum architecting process and its activities.

significant requirements (a.k.a., architectural requirements). This activity aims to define the problems that an architecture needs to address.

(ii) Architectural Modelling aims to satisfy the identified architectural requirements by creating an overall architecture of the system that acts as a blue-print for the implementation. This activity represents the first steps towards providing an architectural solution for ASRs, while bridging the gap between requirements (i.e., desired functionality and quality) and implementation (i.e., executable or simulatable specifications).

(iii) Architectural Implementation exploits the architectural model to implement the software system in terms of algorithmic specifications and executable source code. The implemented software relies on programming languages, compilers, and tools to write, compile, and execute the software.

(iv) Architectural Validation focuses on validating the functionality and quality of the implemented software in the context of architectural requirements. Architectural validation assesses the extent to which the required functionality (i.e., functional requirements) and desired quality (i.e., non-functional requirements) are being satisfied by the implemented software.

(v) Architectural Deployment as the last activity of life cycle is concerned with deploying the validated software for its operationalisation. The deployment involves configuring the executable specification (architectural implementation) on a deployment node (typically an application server) that facilitates the execution of the deployed software.

Based on the available evidence, as illustrated in Fig. 7, the architectural process and its underlying architecting activities for classical computing systems can be tailored to support the architecting process for quantum software systems. However, architectural modelling and implementation activities must explicitly cover architectural requirements specific to quantum software (Hofmeister et al., 2007; Li et al., 2013; Tang et al., 2010). For example, the **architectural requirement** in Fig. 7, i.e., quantum system co-design requires analysing and selecting the hardware (e.g. quantum processor) as well as software (e.g., quantum search algorithm) components to effectively design a quantum computing system (Fig. 1) [S5]. To satisfy this requirement, software as well as hardware engineer need a collaborative design of **architectural model**, referred to as a domain specific model that incorporates software architectural components mapped to instruction set for quantum computing processor [S7]. The co-designed model for a quantum computing system requires architectural **implementation** via model transformation. Model transformation exploits the concepts of model-driven architectures to transform architectural model into the high-level source code that is compiled into quantum instruction set by means of model traceability (mapping between architectural model and executable instruction set) and mode transformation (transition from architectural model to executable instruction set) [S20]. As in Fig. 7, the architecting activities can be iterative, for example, in case of any mismatch between architectural model (i.e., design) and instruction set (i.e., execution) at architectural implementation phase requires maintenance or refactoring of the domain specific model at architectural modelling phase to ensure consistency between design and implementation.

Table 5
Summary view of modelling notations, modelling artifacts, and lifecycle support. (AR = Architectural Requirements, AD = Architectural Design, AI = Architectural Implementation, AE = Architectural Evaluation, AT= Architectural Deployment).

Study ID	Modelling notation	Modelling artifact	Process support				
			AR	AD	AI	AE	AD
S1	Box and arrows	Component diagram		✓			
S2	Graph-based model	State graph		✓	✓		
S4	UML	Class diagram		✓	✓		✓
S5	Graph-based model	Process flow model	✓	✓			
S6	Box and arrows	State transition diagram		✓			✓
S7	Graph-based model	State graph		✓			
S9	UML	State transition diagram	✓	✓			
S14	Box and arrows	Component diagram			✓	✓	
S21	Box and arrows Graph-based model	State graph	✓	✓			
S22	Box and arrows	Component diagram		✓		✓	
S25	Graph-based model	State graph		✓			✓
S27	Graph-based model	Process flow model		✓	✓	✓	
S28	Box and arrows Graph-based model	State graph		✓	✓		
S31	Graph-based model	Process flow model		✓			
S32	UML (Q-UML)	Class diagram Sequence diagram	✓	✓			
S33	Box and arrows	Component diagram		✓	✓		✓

Key Findings of RQ2.1

Finding 7: An architecture design endeavour for the quantum software requires an architecting process to incorporate a number of architecting activities. Existing architectural process can be leveraged to support five architecting activities for quantum software namely *(i) architectural requirements, (ii) architectural modelling, (iii) architectural implementation, (iv) architectural validation,* and *(v) architectural deployment*.
Finding 8: Quantum specific requirements such as modelling Qubits to Qugates and co-design of quantum hardware and software requires domain specific modelling and transformation to be supported by architectural process activities.

5.2. Architectural modelling notations (RQ2.2)

We now answer RQ2.2 that investigates the modelling notations, representing a multitude of graphical models or descriptive notations to specify, document, or represent the architectural models. From architectural process perspective (RQ2.1), the terms modelling notation, modelling language, and architectural language are virtually synonymous and often used interchangeably all referring to same concept of architectural representation either graphically or textually (Malavolta et al., 2012; Pérez-Castillo et al., 2021). For example, to support quantum modelling languages for specifying QSAs, Carlos et al. [S32] have developed Q-UML - an extension to classical UML (Unified Modelling Language) – to support structural and behavioural representation of quantum search algorithms (Medvidovic et al., 2002). Specifically, considering the (co-) design and implementation challenges of QSAs, the role of architectural modelling becomes pivotal to provide a software blue-print model that acts as a bridge between architectural requirements and their implementations, as in Fig. 7. Architectural models essentially becomes the driving artifact in the context of model-driven architecting, where architectural models and model transformation can be exploited for model-based implementation and validation of the system (Moin et al., 2021). To systematically classify, analyse and compare architectural modelling or description languages, some frameworks have been developed that provide a criteria-driven analysis of architectural modelling (Malavolta et al., 2012; Medvidovic and Taylor, 2000). These evaluation criteria can be generally classified into three main types, each type exploring the role of modelling notations to support (i) **architectural specifications** (e.g., architectural representation, architectural structure, syntax, and semantics and, analysing static and dynamic nature of the architectures), (ii) **quality attributes** (e.g., extension, customisation, interoperability of the notations), (iii) **architectural process** (architectural requirements, implementation, validation). The focus of this RQ is architectural representation, not quality attributes of modelling notations, therefore, we mainly focus on aspects of architectural representation and support for architectural process (Fig. 7) with the help of Table 5. Table 5 acts as a structured catalogue to summarise the following information to answer this question.

Available evidence reflects the published research, that provides details of the modelling notation for QSAs [S32].

Modelling notation represents a specific method or technique that is being used to represent the model for QSA. For example, the Q-UML solution provided by Carlos et al. [S32] is an extension of the UML for structural and behavioural representation of the QSA. In addition to the extensions of already existing modelling notations (i.e., QSA specific tailoring), conventional notations such as graph-based models or box and arrow structures have been exploited to specify the structure and semantics of QSAs [S16][S27]. For example, Killoran et al. [S27] exploits graph-based models to represent modules of code to implement the quantum software. Specifically, in graph-based modelling the modules of source code are represented as graph nodes (computational elements and data stores), whereas graph edges represent the interconnection the code modules. This means that **TransactionCommit** module (**node_1**) transfers control to **Update TransactionRecord** module (**node_2**) via **commit** connector (**edge_A**) in architectural graph for quantum software.

Modelling artifact represents a specific artifact (i.e., visual diagram, model) to represent an instance of the architectural model. For example, Carlos et al. [S32] used UML class diagram is being used to represent the structure, whereas UML sequence diagrams are used to represent the behaviour of the quantum search algorithm.

Architectural process support needs modelling notation (and its underlying artifacts) to support specific activities in the architectural process from RQ2.1. For example, Q-UML presents class and sequence diagrams to (i) model requirements and (ii)

specify structural representation and execution flow of the quantum search design. The proposed solution Q-UML does not provide support for other architecting activities such as architectural implementation or evaluation.

Table 5 summarises the core findings of RQ2.2 to streamline most adopted modelling notations, the artifacts being used to model the QSAs, and their impacts on architectural process. We can conclude that most prominent modelling notations can be broadly classified into three main types as UML profiles and extensions such as [S4, S9, S32] (3 studies), graph-based models including [S2, S5, S7, S21, S25, S27, S28, S31] (8 studies), and box and arrow notations including [S1, S6, S14, S16, S19, S21, S22, S28, S33] (9 studies). Some of the most used state transition diagrams, state graph, and process flow models diagram. In the context of architectural process support, existing modelling notations are primarily focused on supporting architectural requirements [S5, S9, S16, S21, S32] (05 studies), design [S1, S2, S4, S5, S6, S7, S9, S16, S19, S21, S22, S25, S27, S28, S31, S32, S33] (17 studies) and implementation phases [S2, S4, S14, S19, S27, S28, S33] (7 studies), whereas there is much less support for life-cycle activities like architectural evaluation [S14, S22, S27] (3 studies) and deployment [S4, S6, S25, S33] (4 studies). Modelling notations are fundamental to the creation of architectural design models that provide foundations for architectural implementation (Medvidovic et al., 2002). In the context of this research, models can facilitate other architectural aspects including but not limited to design decisions (patterns and styles that promote reuse) and tools that support customisation, human decision support, and automation, detailed in subsequent sections of this paper.

Key Findings of RQ2.2

Finding 9: Modelling notations to specify quantum software architectures primarily rely on *box and arrow notations* (having component diagrams) and *graph-based models* (having state graph) to represent the structures and behaviour of quantum software under design. Unlike conventional software architectures that mostly exploit UML notations (often considered as a defacto approach for software design), there is much less evidence on UML-based modelling quantum software architectures

Finding 10: It appears that there is a need for *architectural description languages* and *UML profiles* that can be helpful to leverage existing tools, frameworks, and architectural knowledge to empower the role of designers and architects to model, develop, and evolve quantum software based on re-usability and (semi-) automation.

5.3. Architecture design patterns (RQ2.3)

To answer RQ2.3, we identified a total of six quantum software architecture patterns discussed in ($n = 17$, 50%) studies. In design or architectural context, patterns represent reusable design knowledge, referred to as best practices and concentrated wisdom of designers to address recurring challenges of software development. For example, to address the challenges of system structuring and deployment the layered architecture pattern helps architects to organise software-intensive systems and applications into various layers, each dedicated to different concerns such as data management, user interfacing and computations [S1, S18]. A collection of patterns formally or informally organised into a sequence, results in architectural pattern languages (Leymann, 2019). The focus of this study is individual patterns rather than pattern languages. The set of identified quantum software architecture patterns is presented in Table 6. The most recurring

Table 6
Quantum software architecture design patterns.

Pattern name	Study IDs
Layered pattern	S3, S5, S9, S14, S18, S26, S28, S29
Pipe and filter pattern	S2, S20, S21, S27, S31
Composite design pattern	S4
Prototype design pattern	S24
Recursive containment	S9
Two-qubit gate pattern	S20

design patterns discussed in the 18 primary studies are *layered (n = 8, 24%)* and *pipe and filter architecture (n = 5, 15%)* patterns. The other patterns having low frequency of occurrence are *(composite design, prototype design, recursive containment and two-qubit gate)*. In the following text, we briefly describe the example of a *layered pattern* for the general-purpose microarchitecture of quantum software [S3]. Generally, the *layered pattern* architecture of quantum software mainly consists of several properties that we also need to estimate. These properties include appropriate instruction length, pipeline depth (for parallel quantum gates), and multiple control channels per single instruction. These properties help to construct the basic blocks of quantum software, such as the timing control unit and the microcode instruction set of the overall system. According to our results, the second most frequently reported pattern used for designing quantum software is *pipe and filter*. Killoran et al. [S27] proposed an open-source quantum programming architecture (i.e., Strawberry Fields) based on pipe and filter patterns. The elements of the proposed architecture are organised as the front-end and the back-end. The front-end layer consists of interactive server, application, field API, and quantum programming language components, and the back-end components include a quantum processor and simulator. Both layers communicate through the compiler engine. Our results indicate that the patterns for quantum software are similar to other types of software (e.g., monolithic based architecture, services-oriented based architecture, microservices-based architecture). However, these patterns deal with a series of instructions that need to be executed on quantum processors.

Key Findings of RQ2.3

Finding 11: *Layered* and *pipe and filter* patterns are identified as the most recurring quantum software architecture patterns. However, these are generic or classical patterns that can be used to design any software system. To this end, further research efforts are required to explore and propose new patterns to particularly focus on quantum computing attributes (e.g. superposition and quantum entanglement) and facilitate the architecture of quantum software systems.

5.4. Architecture tools and frameworks (RQ2.4)

RQ2.4 is developed to identify tools and frameworks used to support the architecting activities discussed in Section 5.1. We explored the selected primary studies and noticed that only ($n = 11$, 32%) studies discussed architectural tools and frameworks (see Table 7). The tools and frameworks provide semi- or fully automated solutions to perform architecting activities. Tools broadly refer to software solutions that automate, enhance, or customise process activities. On the other hand, a framework is a set of tools used to perform a bunch of activities, e.g., designing, implementation, and documentation. Each identified tool and framework is interpreted based on the following five criteria (Sajjad et al., 2018), as listed in Table 7). **Source type** refers to the

Table 7
List of identified tools.

Tool/Framework	Source type	Input instructions	Output	Automation level	Evaluation	Study
XACC (eXtreme-scale Accelerator)	CS	HL	QSC	FA	EX	S4
Link layer	CS	QI	SF	FA	EX	S6
Auto E/E framework	OS	MV	SF	SA	IM	S9
eQASM	CS	QI	QA	SA	EX	S14
JKQ (tool set)	OS	HL	SF	FA	EX	S16
Kwant	OS	MV	SF	FA	EX	S17
JKQ DDSIM	OS	HL	SF	FA	EX	S19
QuNetSim	OS	HL	SF	SA	IM	S25
Strawberry fields	OS	HL	SF	FA	EX	S27
qcor	OS	HL	SF	FA	EX	S28
GH-QPL	CS	HL	QSC	SA	IM	S33

type as open source (OS) or close source (CS). In open source, the copyright holders grant the user permissions to study, use or update the tool, framework or system. **Input instructions** are the instructions provided to execute the logic. The instruction types are categorised as high-level (HL), quantum instruction (QI), and mathematical variables (MV). **Output** are the type of post execution findings and categorised as quantum source code (QSC), quantum algorithm (QA), and simulation findings (SF). **Automation level** refers to the automation level of the tool or framework. Automation could be fully-automated (FA), semi-automated (SA), or non-automated (NA). **Evaluation** refers to the performance assessment of a particular tool and framework. Evaluation could be explicit (EX) or implicit (IM). Implicit means that tool or framework is partially evaluated or few of the components are empirically assessed.

The results given in Table 7 reveal that ($n = 7$, 64%) tools and frameworks are open source (OS). Similarly, ($n = 7$, 64%) tools and frameworks accept input code in high-level (HL) programming format (i.e instructions that are more or less independent of a specific type of computer). Moreover, ($n = 8$, 73%) tools and frameworks simulate the high-level input instructions and give the output based on the simulation findings (SF). We further noticed that ($n = 7$, 64%) tools and frameworks are fully-automated (FA) and ($n = 8$, 73%) are explicitly (EX) evaluated based on their performance. The visualisation and summary of the results on tool support are provided in Fig. 8 and Table 8.

Finally, the identified tools and frameworks are classified with respect to their contribution across the architectural process activities reported in Section 5.1. Thematic analysis approach discussed in Section 3.2.4 is followed to categorise the identified tools and frameworks and present the toolchain. It should be noted that a specific tool or framework might contribute to more than one architecting activities and we consider them across multiple activities (see Table 8).

The core architecting activities with respect to the tools and frameworks support are subsequently discussed:

Architectural requirements: We explored the selected primary studies and identified a single framework that focuses on *architectural requirements* (see Table 8) [S9]. Lan et al. [S9], proposed a quantum computing based architectural framework to minimise the gap between the functional domains and meet the requirements of the open electrical and electronic automotive embedded systems. Architectural requirements is a less focus activity with respect to tools and frameworks and the reason might be that quantum software architecture field is in the evolution phase and still the architectural requirements activities do not have tool based automation and customisation support.

Architectural implementation: We identified that a total of six tools and frameworks contributed to the *architectural implementation* activity (see Table 8). More narrow, these tools and frameworks explicitly focus on the *code compilation* and *design to code transformation* sub-activities (see Fig. 8). The power of quantum computer could only be realised by implementing quantum algorithms to control the hardware devices, improve the performance and verify the quantum attributes (Magnani, 2022). Therefore, researchers and practitioners are rushing to develop strategies, tools, frameworks and guidelines to implement algorithms in a simple and efficient way. For example, XACC (eXtreme-scale ACCelerator) provides interfaces to enhance hybrid compilation of programmes developed both in quantum and classical programming languages [S4]. XACC programming framework is designed in a manner that it is entirely independent of selected language, computational model and hardware. The implementation tools and frameworks instantly assist in realising the real-world computation benefits of quantum computers and increase its application across various industrial domains.

Architectural modelling: We noticed that only two *architectural modelling* tools and frameworks are developed, which explicitly address *design model* and *architecture model* sub-activities [S27, S28] (see Fig. 8). Modelling activities performed to develop the overall architecture, which acts as a blueprint for the implementation. The quantum software engineering field is still undeveloped, and it is important to create high-level modelling abstractions for classical software engineers to understand and model the quantum programmes. For example, Strawberry Fields is an open source architectural framework developed to design and optimise the software systems for photonic quantum computers [S27]. Strawberry Fields has built-in engine to convert the code developed in domain specific programming language (blackbird) and run using the photonic quantum computers.

Architectural deployment: Finally, we noticed that only one framework focuses on deployment activities i.e., link layer [S6]. It is developed for quantum communication that improves the entanglement attributes between quantum computers into robust and well defined services. Additionally, strategies for network scheduling are developed to evaluate the protocol performance with respect to different use cases. Architectural deployment is a slightly less focused activity and in near term the tools to automate the deployment activities will be demanding need.

Key Findings of RQ2.4

Finding 12: The identified tools and frameworks are categorise based on the five core attributes namely *(i) source type, (ii) inputs, (iii) outputs, (iv) automation,* and *(v) evaluation level.*

Finding 13: The identified tools and frameworks are mapped across the architecting activities and presented as a toolchain (see Fig. 8). *Architectural implementation* is identified as the most common activity with respect to tools and frameworks. We noticed that six tools and frameworks ($n = 6$, 55%) are developed to automate and customise the architectural implementation activities.

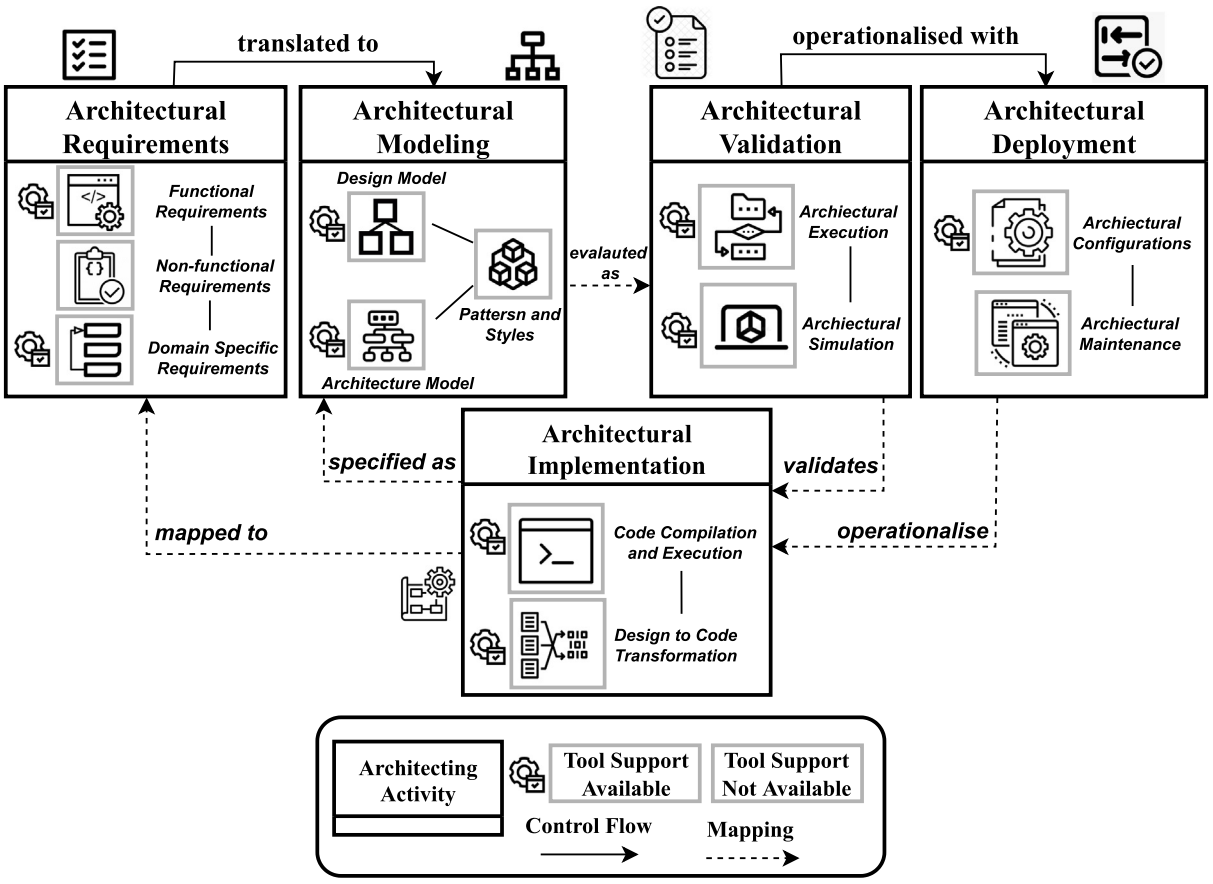

Fig. 8. Tool support for architecting activities.

Table 8
Summary view of tools and frameworks across architecting activities (AR = Architectural Requirements, AM = Architectural Modelling, AI = Architectural Implementation, AV = Architectural Validation, AD= Architectural Deployment).

Study ID	Tool name	Tool focus	Process support				
			AR	AM	AI	AV	AD
S4	xACC	Code compilation			✓		
S9	Auto.E/E Framework	Requirements	✓				
S27	Strawberry fields	Domain modelling		✓			
S28	qCOR	Design		✓	✓		
S14	eQASIM	Programme flow and execution			✓		
S16	JKQ	Code compilation			✓	✓	
S19	JKQ DDSIM	Simulation, compilation			✓		
S33	GH-QPL	Translation and compilation			✓		
S6	LinkLayer	Quantum communication					✓
S17	Kwant	Simulation				✓	
S19	JKQ DDSIM	Simulation				✓	
S25	QuNetSim	Simulation				✓	

5.5. Architecture challenges

The selected primary studies are explored to identify the key challenges of quantum software architecture (RQ2.5). We found that only ($n = 16$, 47%) primary studies reported the architecture challenging factors. The identified challenges are further classified across four core themes: *quantum data transmission and security*, *process-centric architecting*, *architectural tools and technological support*, and *architecting knowledge and expertise*. The thematic analysis approach discussed in Section 3.2.4 is followed to systematically identify the most common themes of the challenging factors (see Fig. 9). For fine-grained analysis, the main themes (core categories) and sub-themes (challenging factors) are presented in Fig. 9 and explicitly discussed below:

5.5.1. Quantum data transmission and security

This theme covers the challenging factors related to the security of network architecture developed for quantum data transmission. We identified a total of four sub-themes (challenging factors) related to the security of quantum network architecture (see Fig. 9). The identified challenging factors are thoroughly discussed as follow:

Quantum key distribution (QKD): The quantum key distribution (QKD) approach is used to develop the ultra-secure network for quantum data transmission [S1]. QKD involves sending the encrypted data and decryption keys over quantum network in qubit state. However, the existing QKD systems are designed to work on the single link quantum network and becomes challenging to operate across multiple networks where the system design and

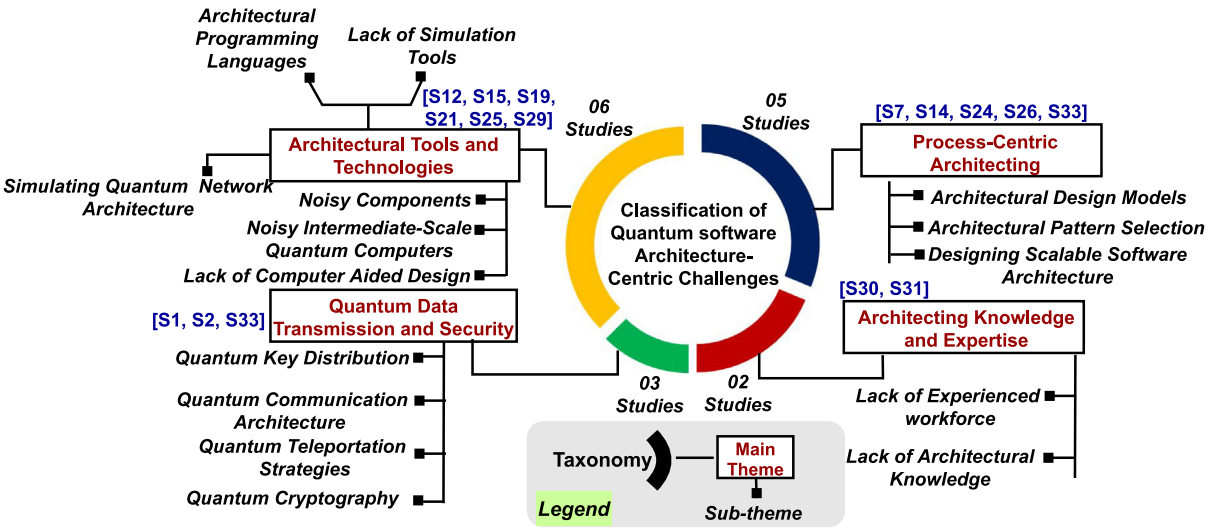

Fig. 9. Thematic classification of identified challenges.

protocols get more complex [S1,S2]. It is evident that there is a strong need of QKD architecture that could deploy across multiple networks for transmitting secure quantum data.

Quantum communication architecture: Architecting a quantum network is challenging with respect to communication perspectives. Quantum network architecture is distinct to classical because of quantum attributes including superposition, entanglement, and quantum measurement [S33]. These attributes brings significant constraints to design the quantum communication architecture. In classical communication, the data bits used to convey the message. In contrast, the qubits are used to transmit the data over quantum communication channel, however; developing a quantum communication architecture needs a major paradigm shift to consider the characteristics of quantum mechanics [S33]. The open-source community should join the efforts to design and fabricate the quantum communication architecture models and interfaces.

Quantum teleportation strategies: Techniques used to transfer quantum information between sender and receiver is called quantum teleportation. Teleportation in science fiction refers to transfer a physical object from location A to B; however, in quantum computing it is used to transfer the Qubits. It has pivotal role in the continuing progress of quantum communication, and quantum networks. However, teleportation is a major challenge in present day quantum computing science because of lack of teleportation protocols, strategies and techniques. Qubits transmission across multiple nodes and computation in the cloud domain is only possible by using the quantum teleportation strategies [S33]. There is a strong need for teleportation protocols and strategies that could reshape the quantum teleportation process.

Quantum cryptography: Practically, quantum cryptography is in its infancy because of data transmission rates and processing limitations. These issues are complicated and challenging to tackle as the high-quality single photons for long-distance required low transmission loss rates. It increases the technological cost of quantum cryptography as compared to the classical. Similarly, developing a sharing infrastructure for secure data encryption and decryption is a significant challenge for quantum cryptography [S33]. The effective encryption and decryption solution is possible by introducing the intermediate node between the sender and receiver. Presently, tackling quantum cryptography challenges is complex, and world-leading technology giants are racing to propose effective solutions.

5.5.2. Process-centric architecting

This theme is developed to categorise the key challenging factors (sub-themes) that could impact the design process of quantum software architecture. Following is the detail description of each selected challenge that covers the process-centric theme.

Architectural design models: There is a lack of models for designing quantum software architectures. The existing models are simplified extended versions of classical modelling approaches and do not explicitly cover the quantum properties including superposition, interference, and entanglement [S7, S24]. The unavailability of particular quantum software design models make it hard to design the system architecture. The expectations to consider quantum computing as alternative to classical increased exponentially [S7, S24]. Consequently, it becomes important to propose rigorous design models in advance for architecting quantum software systems.

Architectural pattern selection: Architectural pattern is a common and reusable solution for generally occurring architectural problems. Selecting an appropriate architecture pattern for a specific quantum problem is a challenging feat. The multi-criteria decision making (MCDM) model could be a best solution to choose a right pattern for right problem [S26]. MCDM model provides a platform to tackle the commonly occurred quantum architectural problems.

Designing scalable quantum software architecture: Scalable systems refer to the information processing concept where a complex system could be developed using the basic building blocks. In quantum architectural scalability, the qubits properties improve or remains consistent when they are extended across multi-qubits systems [S14, S33]. However, architectural scalability also needs to consider the Qubits operations with specific timing, in time instructions fetching and processing to ensure that desired operations are accurately performed [S14]. It is hard to live up the real-world promises and supremacy of quantum computers without architectural scalability [S33].

5.5.3. Architectural tools and technologies

The tools and technologies theme is developed to classify the challenges related to the technical support for architecting activities. In-depth discussion of these challenges is provided as follows:

Noisy Components: Constructing a scalable quantum computer is challenging due to environmental interaction noise that could

destroy its highly fragile components [S12]. Environmental interaction noise generated because of control devices and heat, which can seriously disturb the qubits superposition state and cause computational errors. The robust statistical and mathematical models to estimate the noise impact can significantly improve the computation process and protect the superposition state [S12].

Noisy Intermediate-Scale Quantum (NISQ) computers: It will take decades of research to realise the fault-tolerant quantum computer for solving the wide range real-world problems [S15]. However, the concept of noisy intermediate-scale quantum (NISQ) computer already exists, which contains fifty to a few hundred Qubits but is not smart enough to continuously perform fault-free computations [S15, S21]. The term noisy is used because the present day quantum processors are not sophisticated enough to cope with the environmental impacts, which cause to lose the quantum coherence. Experimental interest is expected and demanded in designing quantum software and hardware architectures to process and execute a large number of error-free Qubits. Transition to quantum computing or more specifically adopting the quantum hardware and quantum computing platforms requires financial investments as well as human skills to manage quantum resources. The PISQ (Perfect Intermediate Scale Quantum) enables the development of new software applications by developing algorithms and evaluating them on quantum simulators that can be executed on existing computing platforms (Bertels et al., 2021). Solutions like PISQ may not be long term solutions to support quantum software, but such solutions allow research and development of quantum logic via simulations that can be deployed and executed on non-quantum computing platforms.

Lack of computer-aided design (CAD) tools: Computer-aided design tools enable the development, change, and optimisation of the architecture design process. These tools are significantly important for developing nanoscale quantum software architectures [S19]. Research to automate and optimise the design approaches for quantum software systems is boosting; however, there is a considerable coordination gap between the CAD and quantum computing community [S19]. Consequently, various proposed CAD tools are failed to achieve the core architectural objectives.

Simulating quantum networks architecture: The quantum internet is defined to transmit quantum data, which is a network architecture of multiple devices and software tools. The concept of a quantum internet is still not in practice, and development efforts are being made to shape it practically. To analyse network protocols, it is important to assess their significance using different simulation tools [S25]. However, limited studies discussed such tools for evaluating quantum network protocols and there is a strong need for advanced simulation tools.

Architectural programming languages: Quantum architectural programming language should provide all the required abstractions both to quantum physicists and algorithm designers. The existing languages are not rich enough to consider for future high-number Qubits algorithms [S29]. They are still unpredictable for complex quantum problems. In the future, the architectural languages should support high-level abstractions for developing and deploying advance algorithms based on quantum superposition and entanglement. Quantum programming languages and frameworks provided by technology giants (e.g., Qiskit by IBM, and Q# by Google) enable software developers to implement QSA as quantum source code that can be executed or implemented on quantum computing platforms. However, the results of an exploratory study show that (i) mined quantum source code repositories available on GitHub and (ii) interviewed quantum code developers suggests that beyond the industry led projects, adoption and applicability of quantum programming in developers' community is still limited (De Stefano et al., 2022). The study also highlights that the current generation of software developers, while implementing quantum code, face a multitude of challenges that range from quantum programme comprehension to source code analysis, manipulation, and testing (Wang et al., 2022).

Lack of simulation tools: The lack of simulation tools is considered a major barrier for quantum software architecture research. The need of simulation tools escalates for large-scale practical and reliable measurements [S29]. Generally, the architects are interested in knowing how fast the architecture works for a specific application, which types of operations it can perform, and what would be the reliability level of its results? These questions could possibly be answered by proposing particular simulators for quantum software architecture [S29]. OpenQL provides a quantum programming language and its associated quantum compiler to develop and execute quantum source code. OpenQL also produces quantum assembly code that is technology independent and can be simulated using QX Quantum Computer Simulator (Khammassi et al., 2021).

5.5.4. Architecting knowledge and expertise

Designing a real-world quantum software system require adequate knowledge and expertise, which play major roles to realise the quantum software design and development activities. This theme is developed to organise the core challenges related to quantum software knowledge and expertise. Following is the detail discussion of the identified challenges (sub-themes).

Lack of experienced workforce: Building a workforce for designing a software system is substantially a major challenge in quantum computing domain. The skills needed to develop a classical computing system are different to quantum [S30]. There is a need for specific professional expertise (i.e., human roles in architecture-centric development process) such as quantum software architects, quantum code developers, and quantum domain engineers (Khan et al., 2022b). The technical team should understand physics to characterise the quantum properties of software systems. Such expertise during the quantum software design and architecting phase can enrich the architecting activities to better meet quantum-specific requirements of the software. Designing and architecting quantum software is radically a different concept, and it demands skillful quantum technical and managerial workforce [S30].

Lack of architectural knowledge: The research field to understand quantum mechanics and integrate it in computing domain by designing quantum software architecture is far from being mature. Various architectural solutions are proposed to develop a quantum software system; however, it require deep knowledge of theory, technology, and understanding to select and implement a suitable solution based on the architectural problem [S31]. It is important to educate the quantum software engineering community to reshape the architecture processes, activities and practices [S31].

Key Findings of RQ2.5

Finding 14: Following four core themes of the identified challenges are developed: *quantum data transmission and security, process-centric architecting, architectural tools and technologies*, and *architecting knowledge*.

Finding 15: We observed that most of the ($n = 6$, 40%) challenges are related to the *architectural tools and technologies* theme. The existing tools and technologies are not at advance level to tackle the architectural problems and it cause various challenges. This is inline with the finding to develop a software engineering community that focuses on devising advance level tools and technologies for managing quantum software architecture challenges (Zhao, 2020).

Table 9
A Summary of the key findings of SLR.

Demography of Published Research (RQ-1.1- RQ1.3)
RQ-1.1 – *Frequency and types of publications* **Frequency**: Years of publications = 2004 to 2021 with the most number of publications from 2018 – 2021 (21, 62%) **Types**: Journal articles (15, 44%), Conference proceedings (13, 38%), Workshop paper (4, 12%), Symposium paper (2, 6%)
RQ-1.2 – *Types of published research* Personal experience papers, Philosophical papers, Opinion papers, Validation research, Proposal of solution, Proposal of solution and validation research
RQ-1.3 – *Application domains of research* Systems and hardware engineering (20, 59%), Software engineering (5, 15%), Source code compilation (4, 12%), Network security (4, 12%), Smart systems (1, 3%)
Architectural solutions for quantum software (RQ-2.1- RQ-2.5)
RQ-2.1 – *Architectural process for quantum software* –Architectural requirements –Architectural modelling –Architectural implementation –Architectural validation –Architectural deployment
RQ-2.2 – *Architectural modelling notations* –Graph-based models –Box and arrow –UML –UML (Q-UML)
Q-2.3 – *Architectural patterns* Layered pattern, Pipe and filter pattern, Composite design pattern, Prototype design pattern, Recursive containment, Two-qubit gate patterns
Q-2.4 – *Architectural tools and frameworks* XACC, Link layer, Auto E/E framework, Strawberry fields, qCOR, eQASM, JKQ, Kwant, JKQ DDSIM,QuNetSim, GH-QPL
RQ-2.5 – *Emerging challenges for quantum software architectures* –Process-centred architecting –Architecting knowledge and expertise –Quantum data transmission and security –Architectural tools and technologies

6. Key findings and implications of the SLR

We now summarise the core findings of the SLR – discussing key results as answers to all RQs – that highlight the state-of-research on architecting quantum software in Section 6.1. We also discuss the implications of the SLR on future academic research in Section 6.2 and its significance along with the potential relevance of this SLR to industrial solutions that address challenges of quantum software architecting in Section 6.3.

6.1. Summary of key findings

A conclusive summary of each RQ is presented in Table 9 that structures the general demographic details of published research, answering RQ-1.1 to RQ-1.3 and architectural solutions for quantum software answering RQ-2.1 to RQ-2.5. Table 9 can be looked up to identify the core finding corresponding to a specific RQ quickly. For example, a summary of the answer to RQ-1.3 highlights that architectural solutions for quantum software can be applied to several domains such as systems and hardware engineering, software engineering, source code compilation, network security, and smart systems. Since the year 2018, a comparative growth in research on QSE and more specifically quantum software design, architecture, and implementation can be attributed to a number of factors. Our study identifies three such factors including (i) a number of pioneering surveys on quantum software engineering and development (Zhao, 2020; Piattini et al., 2021; Gill et al., 2022; Ali et al., 2022) (ii) community-wide initiatives with dedicated workshops and conferences for quantum software (Moguel et al., 2020; Abreu et al., 2021a; Barzen et al., 2021, 2022), along with the emergence of quantum programming models and languages (Zhao et al., 2021; Sunita et al., 2021; De Stefano et al., 2022; Khammassi et al., 2021). Moreover, beyond academic research, the recently growing interest to exploit quantum computing and technologies in IT industry is based on rapid advances in quantum hardware and quantum programming languages that support QSE initiatives in terms of developing quantum software systems and applications (Microsoft, 2021; Behera et al., 2019; Courtland, 2017). It is vital to mention that the launch of the Quantum Flagship project in 2018 (funded by the European Commission) reflects regional and global ambitions to foster research and development on quantum computing technologies (Anon, 2022a). Similarly, the studies [S31, S34] present solutions that enable software designers and architects to design and implement quantum software using architectural components and connectors that can be mapped to source code modules and interaction between the models. Similarly, Table 9 highlights the key findings for RQ-2.3 that to model and represent quantum software architectures the most prominent architectural notations are graph-based modules, box and arrow structures, and Unified Modelling Language. For example, the study [S32] presents a quantum-specific UML named Q-UML that exploits class and sequence diagrams to represent the behaviour and structure of quantum software systems. The details in Table 9 are self-explanatory and focus on summarising the core findings that have already been discussed in Sections 4–5.

6.2. Research implications

(i) Research types based analysis is performed to understand the types of research conducted by the selected primary

studies (see Section 4.2). However, we found that none of the studies conducted evaluation research to assess a particular problem or solution. Quantum software architecture is an emerging research area and no evaluation research studies conducted to assess the contributions promised by the available architectural solutions. It is a significant research gap, and we encourage the researchers to focus on evaluation research to appraise the real-world significance of quantum software systems as well as the existing relevant architectural problems.

(ii) Most of the research studies were conducted across five application areas (see Fig. 6(b)); however, we were not able to find enough evidence related to other important areas, like *model-driven quantum software architecture (MDQSA), quantum AI software architecture,* and quantum software architecture applications for the industrial problems (Ali et al., 2022; Bertels et al., 2021). The possible reason for lack of research in the mentioned areas might be that quantum software architecture is a novel research area and most of the studies focused on proposing architectural solutions for quantum hardware systems (see Fig. 6(b)). Therefore, we encourage the research community to put more focus on the following areas: (1) Model driven quantum software architecture (MDQSA) to manage complexity, achieve high level reuse and reduce the development efforts (Abreu et al., 2021b). (2) Quantum AI software architecture to improve state-of-the-art and propose solutions to operate beyond the classical competencies (Graef and Georgievski, 2021). (3) Boost industrial awareness related to quantum software architecture and develop architectural solutions to deal with complex industrial problems (Anon, 2022a).

(iii) Concerning the domain problem, we thoroughly investigated the challenging factors of quantum software architecture (see Section 5.5) and mapped these factors across different major themes. Thematic mapping provides a conceptual framework to understand the broad picture of the identified challenges and barriers of quantum software architecture (Medvidovic et al., 2002).

In conclusion, this study provides quick access to the body of knowledge based on quantum software architecture literature.

6.3. Industrial implications

(i) We systematically investigated, analysed, and mapped the existing tools and frameworks across the architecting activities (see Section 5.4). A mapping between architecting activities and corresponding tool support can guide practitioners in exploiting the available tool support (enabling automation) to perform a specific architecting activity. For example, as shown in Table 8 if a practitioner wants to conduct architectural validation, he/she can utilise the QuNetSim on implemented architecture to simulate quantum information processing on a quantum network [S25]. In general, the results of this SLR can facilitate the practitioners to get an overview and analyse the extent to which architecting activities, patterns and existing tool support that enable semi-automation can be leveraged to develop industrial-scale solutions for quantum software.

(ii) We proposed an architecting process, which consists of a sequential list of activities, actions, and events to develop a scalable quantum software architecture (see Section 5.1). The proposed process acts as a blueprint for practitioners to understand the inputs, workflow, and outputs of the quantum software architecting process (see Section 5.1).

(iii) Thematic classification of identified challenges (see Fig. 9) provides an overview of potential barriers that need to consider by practitioners before initiating the architecting activities (De Stefano et al., 2022).

(iv) Several studies ($n = 11$, 32%) discussed architectural tools and frameworks (see Section 5.4). We developed a toolchain of the identified tools and frameworks based on their contribution across the architecting activities (see Fig. 8). It will assist the practitioners to select a suitable tool or framework with respect to a specific architecting activity. However, there is still a need for industrial efforts to develop more advanced tools to manage the unexplored architecting activities (Khammassi et al., 2021; Miranskyy et al., 2022).

The quantum software architecture is a new and unexplored research area. Academic researchers and industrial practitioners working in quantum software architecture domain are invited to contribute by sharing their experiences. It will alleviate the gap between academic research and industrial practices.

7. Threats to validity

Various threats could impact the validity of this study. However, we adopted the SLR guidelines proposed by Kitchenham and Charters to alleviate these threats (Kitchenham and Charters, 2007). The potential threats are analysed based on the core four types of validity threats: internal validity, external validity, construct validity, and conclusion validity (Wohlin et al., 2012; Zhou et al., 2016).

7.1. Internal validity

The extent to which certain factors affect the results and analysis of the extracted data is called internal validity. Threats to the internal validity of this study could happen in the following SLR phases:

Search strategy:. It might be possible that relevant primary studies are missed during the search process because of the search strings and the overlap across the selected studies due to the snowballing approach, as highlighted by Jalali and Wohlin Jalali and Wohlin (2012). However, we explicitly defined the search strategy in Section 3.1.3. The first three authors extracted the search terms based on their understanding of RQs, which were further refined by all the authors in consent meetings. Moreover, the search terms were used to develop the search string, which was iteratively developed by all the authors. It should be noted that the authors have extensive research experience in conducting SLR based studies in the software engineering domain.

Studies selection and quality assessment:. The inclusion and exclusion criteria are defined in Section 3.1.4 and used to filter the search results and select the most relevant studies. The first three authors jointly participated in the studies selection process. Furthermore, the first author evaluated the quality of each selected study against the assessment criteria defined in Section 3.2.2. The second and third authors independently verified the assessment results to avoid personal bias.

Data extraction:. Personal bias is a fundamental data extraction threat in SLR studies. We mitigate this threat by defining the data extraction form (see Table 4) to consistently extract the relevant data. The first three authors initially extracted the data; however, the other co-authors participated in the discussion meetings to remove any doubt and verify the data as suggested by Wohlin et al. (2012).

Data synthesis:. Inaccurate data classification and mapping might cause subjective interpretation bias. However, this threat has been alleviated by following thematic classification guidelines provided by Braun and Clarke (Braun and Clarke, 2006). Moreover, quantitative and qualitative methods are used to analyse the collected data. The bias in the data synthesis process could impact the data interpretation process. This threat has been lessen by using the well-established descriptive statistical approaches to analyse the quantitative data and thematic mapping for the qualitative data.

7.2. External validity

External validity refers to the degree to which the study findings could be generalised. We do not claim the generalisability of this study, however; we tried to maximise it by providing an explicit overview of quantum software architecture and logically setting the collected data, results, analysis, and conclusions in the study domain. We followed the rigorous protocol-based SLR approach to attain external validity. Moreover, we followed the guidelines provided by Chen et al. (2010) to search and select the most appropriate digital repositories and target the relevant peer-reviewed studies. Methodological details (Section 3, and Fig. 2), SLR protocol, and data extraction mechanism can support the identification and synthesis of new studies and more RQs to extend this research and minimise the threat to external validity.

7.3. Construct validity

A relevant construct validity could be "data items" since we as the researches observed, decided, and pick up the text fragments or content from the identified studies. Perhaps, this data extraction might not have been correctly performed due to different reasons. For instance, inappropriate search strategies could cause threats like returning a set of irrelevant studies or missing the relevant articles. We tried to mitigate these threats by following operation measures, e.g., conducting group meetings to finalise the search string, developing studies inclusion and exclusion criteria, performing studies quality assessment, and using data extraction form to remove interpersonal bias. Additionally, the search string is customised according to the peculiarities of the selected databases to identify the most relevant studies.

7.4. Conclusion validity

Conclusion validity refers to the degree to which the study conclusions are credible or reasonable. In this SLR, the selection criteria was strict so only quality studies (a clear objective and evaluation) were selected for the analysis in this paper (Kitchenham and Charters, 2007). Additionally, brainstorming sessions are conducted by the authors to discuss the study findings and draw the correct conclusions. It was acknowledged that beyond the scope of current SLR, future efforts may be needed to evolve the results and conclusions, if newly published research is to be investigated, extending the findings of this SLR.

8. Related work

To the best of our knowledge, this work is the first comprehensive systematic literature review on the research of quantum software architecture, including architecting activities, modelling notations, design patterns, tools and frameworks, and architectural challenges. This section discussed the related work that covers different aspects of quantum software engineering (Zhao, 2020; Gill et al., 2022; Sunita et al., 2021; Paulo and de Camargo, 2021; García et al., 2022).

Zhao (2020) conducted a classical survey to cover core quantum software engineering life-cycle activities. Zhao summarised that the quantum software development concept emerges from quantum programming languages, and it is considered synonymous to quantum programming (Zhao, 2020). However, there is a significant need of complete software engineering discipline for quantum software development. This survey extensively discussed the technological support for quantum software development life-cycle phases, including requirements engineering, design, implementation, testing, and maintenance. The study findings reveal that these areas (phases) are rapidly growing; however, they are still far from being mature.

Gill et al. (2022) conducted a comprehensive literature survey to provide in-depth observations of quantum computing concepts and discuss the open challenges experienced by the quantum computing community. A list of taxonomies are proposed to provide conceptual understanding of selecting the available quantum computing techniques and determining the optimal strategies to utilise the classical supercomputing infrastructure. It is because, the existing quantum computers are still not strong enough to replace the supercomputers. Quantum computers are coping with the scaling-up challenge of quantum qubits. It is still not certain when exactly quantum computers will replace the classical; however, it is expected that many exciting improvements will happen in the next decade.

Sunita et al. (2021) conducted a systematic review that surveys available quantum programming language (QPLs) to overview the state-of-the-art in the context of computer programming for developing quantum-intensive software systems. The study formulates a number of RQs to investigate various aspects of QPL such as types of programming languages, recent trends in the development of QPLs, along with academic and industrial progress on the development and adoption of QPLs. The survey also highlights that well-curated design/architecture for quantum software impacts the selection of QPLs for quantum programming. This SLR is also completed with a recently conducted mixed method research (De Stefano et al., 2022) (mining GitHub repositories and developer survey) to investigate the state-of-practice on QPLs in the context of QSE.

Paulo and de Camargo (2021) recently performed a systematic mapping study, reviewing 24 studies, to analyse the existing research on quantum software development in regard to QSE. The focus of the systematic mapping is to understand the prominent programming infrastructures, differences between the development of classical of quantum-intensive software, and the application domain for quantum software systems. The authors highlight that in the last decade the availability of tools, technologies, and programming infrastructures have given impetus to academic on quantum software engineering.

García et al. (2022) conducted an SLR to explore different types of algorithms developed for quantum machine learning and its applications. The study findings reveal that the various conventional/classical algorithms are used for machine learning solutions in the quantum domain e.g., support vector machine and supervised machines learning k-nearest neighbours (KNN) model. The classical algorithms are mainly used for image classification problems. In broad, the implications of quantum machine learning are promising, however, achieving the full-scale benefits of quantum machine learning is still far from being mature. The large-scale implications of quantum machine learning algorithms are still exceedingly challenging because of quality, speed and scalability issues. It requires massive improvements in the existing QC infrastructure to tackle complex industrial problems.

Table 10
A comparison of results between this systematic review and the existing secondary studies.

This review results	Existing secondary studies				
	Zhao (2020)	Gill et al. (2022)	Sunita et al. (2021)	Paulo and de Camargo (2021)	García et al. (2022)
Protocol based SLR review	X	X	✓(+)	✓(+)	✓(+)
Demographic detail	X	X	✓(+)	X	X
Quantum computing basics	✓(*)	✓(*)	✓(+)	✓(+)	✓(+)
Quantum software engineering	✓(*)	✓(+)	X	✓(+)	✓(+)
Quantum software architecture	✓(+)	X (+)	✓(+)	X	X
Architecture modelling notations	X	X	X	✓(+)	X
Quantum software design patterns	✓(+)	X	X	X	X
Architecture tools and frameworks	X	X	✓(+)	X	X
Challenges	X	X	X	X	X

Note: (✓: included, X: Not included, *: Extensive discussion, +: Simple overview).

8.1. Comparative analysis

The comparative analysis of our work with the existing related studies is shown in Table 10. The results reveal that our findings are significantly distinct to the existing related work studies. For instance, sufficient number of primary studies are published, however; only one secondary study (García et al., 2022) partially followed the formal protocol-based SLR approach to conduct the review study (Kitchenham and Charters, 2007). The SLR guidelines developed by Kitchenham and Charters are widely adopted to conduct systematic literature reviews in software engineering (Kitchenham and Charters, 2007). Similarly, we reported the demographic details of each selected primary study, including publication type, frequency, research types, contribution, and application domains (RQ1), which are not considered in the related work secondary studies.

Moreover, we provided a comprehensive overview of quantum software architecture, however Zhao (2020), and Sunita et al. (2021) provided introductory level details of quantum software architecture. The subsequent comparison is made based on architecting activities and modelling notations, which are ignored in the related studies (see Table 10). We developed RQ2.1 and RQ2.2 to respectively define and discuss the key activities of quantum software architecture and modelling notations. Similarly, Zhao (2020) and Paulo and de Camargo (2021) provided a simple overview of quantum software design. However, we explicitly cover and discuss the existing design patterns (RQ2.3) used to tackle the commonly occurred quantum software architectural problems.

Additionally, no discussion of quantum software tools and frameworks is provided in the related secondary studies. We explicitly explored the selected primary studies to identify the tools and frameworks that support various architecting activities (RQ2.4). Finally, we reported quantum software architecture challenges and provided their thematic classification map (RQ2.5). However, the existing related studies do not provide any details or abstract level discussion of quantum software architecture challenging factors (see Table 10).

9. Conclusions

Quantum software architecture -*design and implementation blueprint for quantum software* - represents a new genre of software architectures to address computation-specific challenges rooted in quantum computing. With a growing momentum for the adoption of quantum age systems, industrial initiatives of technology giants (e.g., Google, Microsoft, IBM) and academic research have focused on exploiting architectural solutions to develop quantum software that manages and manipulates quantum hardware. This SLR focused on investigating peer-reviewed published research that streamlines the role of software architectures in designing, implementing, validating, and deploying quantum software. We reviewed a total of 34 qualitatively selected studies to conduct this SLR by answering a total of 08 RQs to a fine-grained presentation of the results.

Results presents that most of our reviewed studies (n = 21, i.e., 62% approx.) have been published in the last four years *(2018–2021)*. Majority of the published research types (i.e., proposal of solution and validation research *(n = 20, 59%)* indicate that quantum software architecture is in its infancy, rapidly evolving by borrowing concepts from classical software architectures to address quantum specific challenges. Quantum-specific challenges include but are not limited to quantum systems co-design and mapping Qubits/Qugates to architectural components and connectors that can be effectively addressed by deriving a process for architecting quantum software. To support the architectural process, modelling notations need to build on established foundations of UML profiles and architectural description languages for (semi-) formal specification of quantum software architectures. The SLR identified a total of five architecting activities, six architectural patterns that promote reuse, 11 tools and frameworks that can automate and customise the process of quantum software architecting. While investigating the architectural challenges, we identified a total of 15 emerging challenging factors, classified across 04 different categories, to resolve emerging issues pertaining to architectural solutions for quantum software. The implications of this SLR are for:

(i) The researchers interested in focusing on quantum software architecture and willing to fill the open research gaps discussed in the study findings.
(ii) Facilitating the knowledge transfer to practitioners regarding quantum software architecture application domains, architecting activities, modelling notations, design patterns, tools and frameworks, and challenges.

We invite practitioners to step forward to focus more on missing application domains, design architecture description languages, develop tools and frameworks to automate the less focused architecting activities, and propose solutions to tackle the challenging factors. We plan to conduct an empirical study to mine the code hosting and questions and answer public platforms to know practitioners' perceptions regarding the quantum software architecture. We finally plan to compare the results of the empirical study and this SLR to identify the gap between the research and practice regarding quantum software architecture.

CRediT authorship contribution statement

Arif Ali Khan: Conceptualization, Investigation, Writing – original draft, Visualization, Reviewing and editing, Project administration. **Aakash Ahmad:** Conceptualization, Investigation, Writing – original draft, Visualization. **Muhammad Waseem:** Conceptualization, Investigation, Writing – original draft. **Peng Liang:** Conceptualization, Investigation, Writing – reviewing & editing, Supervision. **Mahdi Fahmideh:** Conceptualization, Investigation, Reviewing and editing. **Tommi Mikkonen:** Conceptualization, Investigation, Reviewing and editing. **Pekka Abrahamsson:** Conceptualization, Resources, Investigation, Reviewing and editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article

Acknowledgment

This work has been supported by the Academy of Finland (project DEQSE 349945) and Business Finland (project TORQS 8582/31/2022).

Appendix

See Tables 11 and 12.

Table 11
Selected studies for this SMS.

ID	Authors, Publication Title, and Venue	Publication year	Publication type	Quality score
S1	Vicente Martin, Diego R. López, Alejandro Aguado, Juan Pedro Brito, Julio Setién Villarán, Pedro Jesús Salas Peralta, Carmen Escribano, Víctor Lopez, Antonio Pastor Perales, and Momtchil Peev. A Components Based Framework for Quantum Key Distribution Networks. *In 22nd IEEE International Conference on Transparent Optical Networks (ICTON)*, Bari, Italy, pp. 1-4. I, 2020.	2020	Conference	4
S2	Qiong Li, Dan Le, and Ming Rao. A design and implementation of multi-thread quantum key distribution post-processing software. *In Second IEEE International Conference on Instrumentation, Measurement, Computer, Communication and Control (IMCCC)*, Harbin, China, pp. 272-275, 2012.	2012	Conference	3.5
S3	Xiang Fu, Leon Riesebos, Lingling Lao, Carmen Garcia Almudever, Fabio Sebastiano, Richard Versluis, Edoardo Charbon, and Koen Bertels. A heterogeneous quantum computer architecture. In Proceedings of the 13th ACM International Conference on Computing Frontiers, Como, Italy, pp. 323-330. 2016.	2016	Conference	5
S4	Alexander J.McCaskey, Eugene F.Dumitrescu, Dmitry Liakh, Mengsu Chen, Wu-chun Feng, and Travis S. Humble. A language and hardware independent approach to quantum–classical computing. SoftwareX, 7: pp. 245-254, 2018.	2018	Journal	5
S5	Krysta M. Svore, Alfred V. Aho, Andrew W. Cross, Isaac Chuang, and Igor L. Markov. A layered software architecture for quantum computing design tools. Computer, 39(1): pp. 74-83, 2006.	2006	Journal	4
S6	Axel Dahlberg, Matthew Skrzypczyk, Tim Coopmans, Leon Wubben, Filip Rozpędek, Matteo Pompili, Arian Stolk, Przemysław Pawełczak, Robert Knegjens, Julio A De Oliveira Filho, Ronald Hanson, Stephanie Wehner. A link layer protocol for quantum networks. In Proceedings of the 33rd ACM Special Interest Group on Data Communication (SIGCOMM), Beijing, China, pp. 159-173, 2019.	2019	Conference	5
S7	Thomas Häner, Damian S. Steiger, Krysta Svore, and Matthias Troyer. A software methodology for compiling quantum programmes. Quantum Science and Technology , 3(2): pp. 1-19, 2018.	2018	Journal	3.5
S8	Michael Booth, Edward Dahl, Mark Furtney, and Steven P. Reinhardt. Abstractions considered helpful: a tools architecture for quantum annealers. In 5th IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA USA, pp. 1-2, 2016.	2016	Conference	2.5
S9	Hongbon Lan, Chengrui Zhang, and Hongbin Li. An open design methodology for automotive electrical/electronic system based on quantum platform. Advances in Engineering Software, 39 (6): pp. 526-534, 2008.	2008	Journal	3
S10	Victor Potapov, Sergei Gushansky, Vyacheslav Guzik, and Maxim Polenov. Architecture and software implementation of a quantum computer model. In 2nd Computer Science On-line Conference (CSOC), pp. 59-68. Springer, Cham, Zlin, Czech Republic, pp. 59-68, 2016.	2016	Conference	4
S11	Loyd R. Hook, and Samuel C. Lee. Design and simulation of 2-D 2-dot quantum-dot cellular automata logic. IEEE Transactions on Nanotechnology, 10(5), pp. 996-1003, 2010.	2010	Journal	5
S12	Ilia Polian, and Austin Fowler. Design automation challenges for scalable quantum architectures. In 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), Austin, TX, USA, pp. 1-6, 2015.	2015	Conference	3.5

(*continued on next page*)

Table 11 (*continued*).

ID	Authors, Publication Title, and Venue	Publication year	Publication type	Quality score
S13	Heranmoy Maity, Arijit Kumar Barik, Arindam Biswas, Anup Kumar Bhattacharjee, and Anita Pal. Design of quantum cost, garbage output and delay optimised BCD to excess-3 and 2's complement code converter. Journal of Circuits, Systems and Computers, 27(12): pp. 1-5, 2018.	2018	Journal	4
S14	Xiang Fu, Leon Riesebos, Adriaan Rol, Jeroen Van Straten, Hans van Someren, Nader Khammassi, Imran Ashraf, Raymond Vermeulen, V. Newsum, Kelvin Kwong Lam Loh, Jacob de Sterke, Wouter Vlothuizen, Raymond Schouten, Carmen G. Almudéver, Leonardo DiCarlo, and Koen Bertels. eQASM: An executable quantum instruction set architecture. In 25th IEEE International Symposium on High Performance Computer Architecture (HPCA), Washington, DC, USA, pp. 224-237, 2019.	2019	Symposium	4.5
S15	Prakash Murali, Norbert Matthias Linke, Margaret Martonosi, Ali Javadi Abhari, Nhung Hong Nguyen, and Cinthia Huerta Alderete. Full-stack, real-system quantum computer studies: Architectural comparisons and design insights. In 46th ACM/IEEE Annual International Symposium on Computer Architecture (ISCA), Phoenix, AZ, USA, pp. 527-540. 2019.	2019	Conference	5
S16	Robert Wille, Stefan Hillmich, and Lukas Burgholzer. JKQ: JKU tools for quantum computing. In 33rd IEEE/ACM International Conference On Computer Aided Design (ICCAD), San Diego, CA, USA, pp. 1-5, 2020.	2020	Conference	4
S17	Christoph W Groth, Michael Wimmer, Anton R. Akhmerov, and Xavier Waintal. Kwant: a software package for quantum transport. New Journal of Physics, 16 (6): pp. 1-40, 2014.	2014	Journal	4.5
S18	Nathan Cody Jones, Rodney Van Meter, Austin Fowler, Peter McMahon, Jungsang Kim, Thaddeus Ladd, and Yoshihisa Yamamoto. Layered architecture for quantum computing. Physical Review X, 2(3): pp. 1-27, 2012.	2012	Journal	5
S19	Alwin Zulehner, and Robert Wille. Advanced simulation of quantum computations. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 38 (5): pp. 848-859, 2018.	2018	Journal	5
S20	Gushu Li, Anbang Wu, Yunong Shi, Ali Javadi-Abhari, Yufei Ding, and Yuan Xie. On the Co-Design of Quantum Software and Hardware. In Proceedings of the 8th Annual ACM International Conference on Nanoscale Computing and Communication (NANOCOM), Italy, pp. 1-7, 2021.	2021	Conference	3.5
S21	Teague Tomesh, and Margaret Martonosi. Quantum Codesign. IEEE Micro, 41(5), pp-33-40, 2021.	2021	Journal	3.5
S22	Munish Bhatia, and Avneet Kaur. Quantum computing inspired framework of student performance assessment in smart classroom. Transactions on Emerging Telecommunications Technologies, 32(9): pp. 1-22, 2021.	2021	Journal	3.5
S23	Nan Wu, Haixing Hu, Fangmin Song, Huimin Zheng, and Xiangdong Li. Quantum software framework: a tentative study. Frontiers of Computer Science, 7(3): pp. 341-349, 2013.	2013	Journal	5
S24	Iaakov Exman, and Alon Tsalik Shmilovich. Quantum Software Models: The Density Matrix for Classical and Quantum Software Systems Design. In Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering Workshops (ICSEW), Madrid, Spain, pp. 1-6, 2021.	2021	Workshop	5
S25	Stephen Diadamo, Janis Nötzel, Benjamin Zanger, and Mehmet Mert Beşe. Qunetsim: A software framework for quantum networks. IEEE Transactions on Quantum Engineering, 2: pp. 1-12, 2021.	2021	Journal	3.5
S26	Lalitha Nallamothula. Selection of quantum computing architecture using a decision tree approach. In 3rd International Conference on Intelligent Sustainable Systems (ICISS), Thoothukudi, India, pp. 644-649, 2020.	2020	Conference	5
S27	Killoran, Nathan, Josh Izaac, Nicolás Quesada, Ville Bergholm, Matthew Amy, and Christian Weedbrook. . Strawberry fields: A software platform for photonic quantum computing. Quantum, 3: pp-1-27, 2019.	2019	Journal	3.5
S28	Alexander Mccaskey, Thien Nguyen, Anthony Santana, Daniel Claudino, Tyler Kharazi, and Hal Finkel. Extending c++ for heterogeneous quantum–classical computing. ACM Transactions on Quantum Computing, 2(2):pp. 1-36, 2021.	2021	Journal	5
S29	Krista Svore, Andrew Cross, Alfred Aho, Isaac Chuang, and Igor Markov. Towards a software architecture for quantum computing design tools. In Proceedings of the 2nd International Workshop on Quantum Programming Languages (QPL), pp. 145-162. 2004.	2004	Workshop	3.5
S30	Frank Leymann. Towards a pattern language for quantum algorithms. In International Workshop on Quantum Technology and Optimisation Problems (QTOP), Springer, Cham, Munich, Germany, pp. 218-230, 2019.	2019	Workshop	3.5
S31	Frank Leymann, Johanna Barzen, and Michael Falkenthal. Towards a platform for sharing quantum software. Proceedings of the 13th Advanced Summer School on Service Oriented Computing (SummerSOC), Crete, Greece, pp. 70-74, 2021.	2021	Conference	3

(*continued on next page*)

Table 11 (*continued*).

S32	Carlos A. Pérez-Delgado, and Hector G. Perez-Gonzalez. Towards a quantum software modelling language. In Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW), Seoul, South Korea, pp. 442-444, 2020.	2020	Workshop	2.5
S33	El-Mahdy M.Ameen, , Hesham A. Ali, Mofreh M. Salem, and Mahmoud Badawy. Towards implementation of a generalised architecture for high-level quantum programming language. International Journal of Theoretical Physics, 56(8): pp. 2376-2412, 2017.	2017	Journal	3.5
S34	Rob F.M. van den Brink, Frank Phillipson , and Niels M.P. Neumann. Vision on next level quantum software tooling. In Proceedings of the 10th International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking (COMPUTATION TOOLS), Venice, Italy, pp. 16-23, 2019.	2019	Conference	5

Table 12
Cohen's Kappa test R-Code.

Cohen's Kappa test between authors for the SLR process
libraryL(DescTools) QuantumSoftwareArch <- data.frame(Authors1to3=c(7,6,4,3,10,2,9,5,1,3), Authors4to5=c(7,6,4,3,9,2,8,5,1,3), Authors6to7=c(7,5,3,2,10,2,9,5,1,3)) KappaM(QuantumSoftwareArch) KappaM(QuantumSoftwareArch, method="Conger") KappaM(QuantumSoftwareArch, conf.level=0.95) KappaM(QuantumSoftwareArch, method="Light")
Cohen's Kappa test between authors for the snowballing process
libraryL(DescTools) Snowballing <- data.frame(Authors1to4=c(1,4,2,3,5), Authors5to6=c(1,5,2,3,4)) KappaM(Snowballing) KappaM(Snowballing, method="Conger") KappaM(Snowballing, conf.level=0.95) KappaM(Snowballing, method="Light")

References

Abbott, Benjamin P, Abbott, Richard, Abbott, Thomas D, Abernathy, MR, Acernese, F, Ackley, K, Adams, C, Adams, T, Addesso, P, Adhikari, RX, et al., 2018. Effects of data quality vetoes on a search for compact binary coalescences in advanced LIGO's first observing run. Classical Quantum Gravity 35 (6), 065010.

Abreu, Rui, Ali, Shaukat, Yue, Tao, 2021a. First international workshop on quantum software engineering (Q-SE 2020). ACM SIGSOFT Softw. Eng. Notes 46 (2), 30–32.

Abreu, Rui, Ali, Shaukat, Yue, Tao, Felderer, Michael, Exman, Iaakov, 2021b. Quantum software: Model-driven or search-driven? A Q-SE 2021 workshop report. ACM SIGSOFT Softw. Eng. Notes 46 (4), 23–25.

Ali, Shaukat, Yue, Tao, Abreu, Rui, 2022. When software engineering meets quantum computing. Commun. ACM 65 (4), 84–88.

Alreshidi, Abdulrahman, Ahmad, Aakash, 2019. Architecting software for the internet of thing based systems. Future Internet 11 (7), 153.

Anon, 2022. ISO/IEC/IEEE 42010:2011 systems and software engineering - architecture description. URL https://www.iso.org/standard/50508.html, (Accessed 25 January 2022).

Anon, 2022a. Introduction to the Quantum Flagship, post=, (accessed 12 april 2022)..

Anon, 2022b. Quantum computing research trends report Get ready for the second quantum revolution, post=, (accessed 09 november 2022)..

Baczewski, Andrew David, Moussa, Jonathan Edward, Sarovar, Mohan, 2017. Co-design strategies for quantum simulators. Technical Report SAND2017-0164C, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

Barzen, Johanna, Leymann, Frank, Wild, Karoline, Feld, Sebastian, 2021. 1 st workshop on quantum software architecture (QSA). In: 2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C). IEEE, p. 94.

Barzen, Johanna, Martin-Fernandez, Francisco, Wimmer, Manuel, 2022. IEEE International Conference on Quantum Software (QSW 2022).

Behera, Bikash K, Reza, Tasnum, Gupta, Angad, Panigrahi, Prasanta K, 2019. Designing quantum router in IBM quantum computer. Quantum Inf. Process. 18 (11), 1–13.

Bertels, Koen, Sarkar, Aritra, Ashraf, Imran, 2021. Quantum computing—From NISQ to PISQ. IEEE Micro 41 (5), 24–32.

Biamonte, Jacob, Wittek, Peter, Pancotti, Nicola, Rebentrost, Patrick, Wiebe, Nathan, Lloyd, Seth, 2017. Quantum machine learning. Nature 549 (7671), 195–202.

Braun, Virginia, Clarke, Victoria, 2006. Using thematic analysis in psychology. Qual. Res. Psychol. 3 (2), 77–101.

Campos, José, Souto, André, 2021. Qbugs: A collection of reproducible bugs in quantum algorithms and a supporting infrastructure to enable controlled quantum software testing and debugging experiments. In: 2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE). IEEE, pp. 28–32.

Chen, Lianipng, Babar, Muhammad Ali, Zhang, He, 2010. Towards an evidence-based understanding of electronic data sources. In: Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering. EASE, pp. 1–4.

Childs, Andrew M, Maslov, Dmitri, Nam, Yunseong, Ross, Neil J, Su, Yuan, 2018. Toward the first quantum simulation with quantum speedup. Proc. Natl. Acad. Sci. 115 (38), 9456–9461.

Chong, Frederic T., Franklin, Diana, Martonosi, Margaret, 2017. Programming languages and compiler design for realistic quantum hardware. Nature 549 (7671), 180–187.

Cohen, Jacob, 1960. A coefficient of agreement for nominal scales. Edu. Psychol. Meas. 20 (1), 37–46.

Courtland, Rachel, 2017. Google aims for quantum computing supremacy [news]. IEEE Spectr. 54 (6), 9–10.

De Stefano, Manuel, Pecorelli, Fabiano, Di Nucci, Dario, Palomba, Fabio, De Lucia, Andrea, 2022. Software engineering for quantum programming: How far are we? J. Syst. Softw. 190, 111326.

Deutsch, David, 1985. Quantum theory, the church–turing principle and the universal quantum computer. Proc. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci. 400 (1818), 97–117.

Dey, Nivedita, Ghosh, Mrityunjay, Chakrabarti, Amlan, et al., 2020. QDLC–the quantum development life cycle. arXiv preprint arXiv:2010.08053.

Di Francesco, Paolo, Lago, Patricia, Malavolta, Ivano, 2019. Architecting with microservices: A systematic mapping study. J. Syst. Softw. 150, 77–97.

DiAdamo, Stephen, Nötzel, Janis, Zanger, Benjamin, Beşe, Mehmet Mert, 2021. Qunetsim: A software framework for quantum networks. IEEE Trans. Quantum Eng. 2, 1–12.

Dirac, Paul Adrien Maurice, 1981. The Principles of Quantum Mechanics. (27), Oxford University Press.

Dorfman, Merlin, Thayer, Richard H., 1997. Software Engineering. IEEE Computer.

Everitt, M.J., Michael, J. De C., Dwyer, Vincent M., 2016. Quantum systems engineering: A structured approach to accelerating the development of a quantum technology industry. In: IEEE 18th International Conference on Transparent Optical Networks. ICTON, pp. 1–4.

Fahmideh, Mahdi, Ahmed, Aakash, Behnaz, Ali, Grundy, John, Susilo, Willy, 2021a. Software engineering for internet of things: The practitioner's perspective. arXiv preprint arXiv:2102.10708.

Fahmideh, Mahdi, Grundy, John, Ahmed, Aakash, Shen, Jun, Yan, Jun, Mougouei, Davoud, Wang, Peng, Ghose, Aditya, Gunawardana, Anuradha, Aickelin, Uwe, et al., 2021b. Software engineering for blockchain based software systems: Foundations, survey, and future directions. arXiv preprint arXiv:2105.01881.

Felizardo, Katia Romero, Mendes, Emilia, Kalinowski, Marcos, Souza, Érica Ferreira, Vijaykumar, Nandamudi L, 2016. Using forward snowballing to update systematic reviews in software engineering. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. pp. 1–6.

García, David Peral, Cruz-Benito, Juan, García-Peñalvo, Francisco José, 2022. Systematic literature review: Quantum machine learning and its applications. arXiv preprint arXiv:2201.04093.

Garcia, Joshua, Mirakhorli, Mehdi, Xiao, Lu, Zhao, Yutong, Mujhid, Ibrahim, Pham, Khoi, Okutan, Ahmet, Malek, Sam, Kazman, Rick, Cai, Yuanfang, Medvidović, Nenad, 2021. Constructing a shared infrastructure for software architecture analysis and maintenance. In: IEEE 18th International Conference on Software Architecture. ICSA, pp. 150–161.

Garousi, Vahid, Felderer, Michael, Mäntylä, Mika V., 2019. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. Inf. Softw. Technol. 106, 101–121.
Gay, Simon J., 2006. Quantum programming languages: Survey and bibliography. Math. Struct. Comput. Sci. 16 (4), 581–600.
Gill, Sukhpal Singh, Kumar, Adarsh, Singh, Harvinder, Singh, Manmeet, Kaur, Kamalpreet, Usman, Muhammad, Buyya, Rajkumar, 2022. Quantum computing: A taxonomy, systematic review and future directions. Softw. - Pract. Exp. 52 (1), 66–114.
Goled, Shraddha, 2021. Top countries pumping money into quantum computing technology. https://tinyurl.com/499hu6ca, (Accessed 05 December 2021).
Graef, Sebastian, Georgievski, Ilche, 2021. Software architecture for next-generation AI planning systems. arXiv preprint arXiv:2102.10985.
Grimsley, Harper R, Economou, Sophia E, Barnes, Edwin, Mayhall, Nicholas J, 2019. An adaptive variational algorithm for exact molecular simulations on a quantum computer. Nature Commun. 10 (1), 1–9.
Hofmeister, Christine, Kruchten, Philippe, Nord, Robert L, Obbink, Henk, Ran, Alexander, America, Pierre, 2007. A general model of software architecture design derived from five industrial approaches. J. Syst. Softw. 80 (1), 106–126.
ISO/IEC/IE.E.E. 90003:2018, 2021. Software engineering — Guidelines for the application of iso 9001:2015 to computer software. https://www.iso.org/standard/74348.html. (Accessed 09 December 2021).
Jalali, Samireh, Wohlin, Claes, 2012. Systematic literature studies: database searches vs. backward snowballing. In: Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. pp. 29–38.
Khammassi, Nader, Ashraf, Imran, Someren, JV, Nane, Razvan, Krol, AM, Rol, M Adriaan, Lao, Lingling, Bertels, Koen, Almudever, Carmen G, 2021. OpenQL: A portable quantum programming framework for quantum accelerators. ACM J. Emerg. Technol. Comput. Syst. (JETC) 18 (1), 1–24.
Khan, Arif Ali, Ahmad, Aakash, Waseem, Muhammad, Liang, Peng, Fahmideh, Mahdi, Mikkonen, Tommi, Abrahamsson, Pekka, 2022a. Replication package for the paper: Software architecture for quantum computing systems - A systematic review. http://tinyurl.com/2uh2n25h. (Accessed 03 July 2022).
Khan, Arif Ali, Akbar, Muhammad Azeem, Ahmad, Aakash, Fahmideh, Mahdi, Shameem, Mohammad, Lahtinen, Valtteri, Waseem, Muhammad, Mikkonen, Tommi, 2022b. Agile practices for quantum software development: Practitioners perspectives. arXiv preprint arXiv:2210.09825.
Kitchenham, B., Charters, S., 2007. Guidelines for Performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE Technical Report EBSE-2007-01, Keele University and Durham University.
Kitchenham, Barbara A., Dyba, Tore, Jorgensen, Magne, 2004. Evidence-based software engineering. In: Proceedings of 26th IEEE International Conference on Software Engineering. pp. 273–281.
Krüger, Tom, Mauerer, Wolfgang, 2020. Quantum annealing-based software components: An experimental case study with sat solving. In: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSE-W). pp. 445–450.
Landis, J. Richard, Koch, Gary G., 1977. The measurement of observer agreement for categorical data. Biometrics 159–174.
Lapedus, Mark, 2021. The great quantum computing race. https://semiengineering.com/the-great-quantum-computing-race/, (Accessed 16 December 2021).
Leymann, Frank, 2019. Towards a pattern language for quantum algorithms. In: Quantum Technology and Optimization Problems. In: Lecture Notes in Computer Science (LNCS), Springer International Publishing, pp. 218–230. http://dx.doi.org/10.1007/978-3-030-14082-3_19.
Li, Zengyang, Liang, Peng, Avgeriou, Paris, 2013. Application of knowledge-based approaches in software architecture: A systematic mapping study. Inf. Softw. Technol. 55 (5), 777–794.
Magnani, Roberto, 2022. Quantum computing - skill creation is a key factor. Report of informal conversations with students and professors. https://www.linkedin.com/pulse/quantum-computing-skill-creation-key-factor-report-informal/ (Accessed 25 January 2022).
Malavolta, Ivano, Lago, Patricia, Muccini, Henry, Pelliccione, Patrizio, Tang, Antony, 2012. What industry needs from architectural languages: A survey. IEEE Trans. Softw. Eng. 39 (6), 869–891.
McArdle, Sam, Endo, Suguru, Aspuru-Guzik, Alán, Benjamin, Simon C, Yuan, Xiao, 2020. Quantum computational chemistry. Rev. Modern Phys. 92 (1), 015003.
Medvidovic, Nenad, Rosenblum, David S, Redmiles, David F, Robbins, Jason E, 2002. Modeling software architectures in the unified modeling language. ACM Trans. Softw. Eng. Methodol. (TOSEM) 11 (1), 2–57.
Medvidovic, Nenad, Taylor, Richard N., 2000. A classification and comparison framework for software architecture description languages. IEEE Trans. Softw. Eng. 26 (1), 70–93.
Microsoft, 2021. Quantum computing. URL , (Accessed 05 December 2021).
Miranskyy, Andriy, Khan, Mushahid, Faye, Jean Paul Latyr, Mendes, Udson C, 2022. Quantum computing for software engineering: Prospects. arXiv preprint arXiv:2203.03575.
Moguel, Enrique, Berrocal, Javier, García-Alonso, José, Murillo, Juan Manuel, 2020. A roadmap for quantum software engineering: Applying the lessons learned from the classics.. In: 1st Quantum Software Engineering and Technology Workshop (Q-SET), Co-Located with IEEE International Conference on Quantum Computing and Engineering. QCE, pp. 5–13.
Moin, Armin, Challenger, Moharram, Badii, Atta, Günnemann, Stephan, 2021. MDE4qai: Towards model-driven engineering for quantum artificial intelligence. arXiv preprint arXiv:2107.06708.
Montanaro, Ashley, 2016. Quantum algorithms: an overview. Npj Quantum Inform. 2 (1), 1–8.
Mosca, Michele, 2018. Cybersecurity in an era with quantum computers: Will we be ready? IEEE Secur. Privacy 16 (5), 38–41.
Mourão, Erica, Kalinowski, Marcos, Murta, Leonardo, Mendes, Emilia, Wohlin, Claes, 2017. Investigating the use of a hybrid search strategy for systematic reviews. In: 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ESEM, IEEE, pp. 193–198.
Naveh, Yehuda, 2021. Quantum software development is still in its infancy. https://www.forbes.com/sites/forbestechcouncil/2021/06/23/quantum-software-development-is-still-in-its-infancy/?sh=4f0f7e976ddd, (Accessed 25 December 2021).
Nguyen, Thien, Arya, Daanish, Doherty, Marcus, Herrmann, Nils, Kuhlmann, Johannes, Preis, Florian, Scott, Pat, Yin, Simon, 2022. Software for massively parallel quantum computing. arXiv preprint arXiv:2211.13355.
Paulo, Eduardo Zanni Junior, de Camargo, Valter Vieira, 2021. A systematic mapping on quantum software development in the context of software engineering. arXiv preprint arXiv:2106.00926.
Pérez, Jorge, Díaz, Jessica, Garcia-Martin, Javier, Tabuenca, Bernardo, 2020. Systematic literature reviews in software engineering—Enhancement of the study selection process using Cohen's kappa statistic. J. Syst. Softw. 168, 110657.
Pérez-Castillo, Ricardo, Jiménez-Navajas, Luis, Piattini, Mario, 2021. Modelling quantum circuits with UML. arXiv preprint arXiv:2103.16169.
Pérez-Castillo, Ricardo, Piattini, Mario, 2022. Design of classical-quantum systems with UML. Computing 1–29.
Petersen, Kai, Vakkalanka, Sairam, Kuzniarz, Ludwik, 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. Inf. Softw. Technol. 64, 1–18.
Piattini, Mario, Serrano, Manuel, Perez-Castillo, Ricardo, Petersen, Guido, Hevia, Jose Luis, 2021. Toward a quantum software engineering. IT Prof. 23 (1), 62–66.
Pizard, Sebastián, Acerenza, Fernando, Otegui, Ximena, Moreno, Silvana, Vallespir, Diego, Kitchenham, Barbara, 2021. Training students in evidence-based software engineering and systematic reviews: a systematic review and empirical study. Empir. Softw. Eng. 26 (3), 1–53.
Rebentrost, Patrick, Mohseni, Masoud, Lloyd, Seth, 2014. Quantum support vector machine for big data classification. Phys. Rev. Lett. 113 (13), 130503.
Sajjad, Maryam, Ahmad, Aakash, Malik, Asad Waqar, Altamimi, Ahmed B, Alseadoon, Ibrahim, 2018. Classification and mapping of adaptive security for mobile computing. IEEE Trans. Emerg. Top. Comput. 8 (3), 814–832.
Shepard, Jeff, 2021. Quantum computing system architectures. https://www.microcontrollertips.com/quantum-computing-system-architectures/, (Accessed 16 December 2021).
Sofge, Donald A., 2008. A survey of quantum programming languages: History, methods, and tools. In: Second IEEE International Conference on Quantum, Nano and Micro Technologies (ICQNM 2008). pp. 66–71.
Stepney, Susan, Braunstein, Samuel L, Clark, John A, Tyrrell, Andy, Adamatzky, Andrew, Smith, Robert E, Addis, Tom, Johnson, Colin, Timmis, Jonathan, Welch, Peter, Milnerk, Robin, Partridge, Derek, 2005. Journeys in non-classical computation I: A grand challenge for computing research. Int. J. Parallel Emergent Distrib. Syst. 20 (1), 5–19.
Sunita, Garhwal, Ghorani, Maryam, Ahmad, Amir, 2021. Quantum programming language: A systematic review of research topic and top cited languages. Arch. Comput. Methods Eng. 28 (2), 289–310.
Svore, Krysta M, Aho, Alfred V, Cross, Andrew W, Chuang, Isaac, Markov, Igor L, 2006. A layered software architecture for quantum computing design tools. Computer 39 (1), 74–83.
Tang, Antony, Avgeriou, Paris, Jansen, Anton, Capilla, Rafael, Babar, Muhammad Ali, 2010. A comparative study of architecture knowledge management tools. J. Syst. Softw. 83 (3), 352–370.
Wang, Xinyi, Arcaini, Paolo, Yue, Tao, Ali, Shaukat, 2022. QuSBT: Search-based testing of quantum programs. arXiv preprint arXiv:2204.08561.
Waseem, Muhammad, Liang, Peng, Shahin, Mojtaba, 2020. A systematic mapping study on microservices architecture in devops. J. Syst. Softw. 170, 110798.
Weder, Benjamin, Barzen, Johanna, Leymann, Frank, Vietz, Daniel, 2022. Quantum software development lifecycle. In: Quantum Softw. Eng.. Springer, pp. 61–83.
Wieringa, Roel, Maiden, Neil, Mead, Nancy, Rolland, Colette, 2006. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. Requir. Eng. 11 (1), 102–107.

Wohlin, Claes, 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. pp. 1–10.
Wohlin, Claes, Runeson, Per, Höst, Martin, Ohlsson, Magnus C, Regnell, Björn, Wesslén, Anders, 2012. Experimentation in software engineering. Springer Science & Business Media.
Xu, Xiwei, Weber, Ingo, Staples, Mark, Zhu, Liming, Bosch, Jan, Bass, Len, Pautasso, Cesare, Rimba, Paul, 2017. A taxonomy of blockchain-based systems for architecture design. In: IEEE International Conference on Software Architecture. ICSA, pp. 243–252.
Ying, Mingsheng, 2016. Foundations of Quantum Programming. Morgan Kaufmann.
Zeilinger, Anton, 1999. Experiment and the foundations of quantum physics. Rev. Modern Phys. 71 (2), 482–498.
Zhang, He, Babar, Muhammad Ali, Bai, Xu, Li, Juan, Huang, Liguo, 2011. An empirical assessment of a systematic search process for systematic reviews. In: 15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011). IET, pp. 56–65.
Zhao, Jianjun, 2020. Quantum software engineering: Landscapes and horizons. arXiv preprint arXiv:2007.07047.
Zhao, Pengzhan, Zhao, Jianjun, Miao, Zhongtao, Lan, Shuhan, 2021. Bugs4Q: A benchmark of real bugs for quantum programs. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 1373–1376.
Zhou, Xin, Jin, Yuqin, Zhang, He, Li, Shanshan, Huang, Xin, 2016. A map of threats to validity of systematic literature reviews in software engineering. In: 23rd IEEE Asia-Pacific Software Engineering Conference. APSEC, pp. 153–160.

Arif Ali Khan works as an Assistant Professor with the M3S Empirical Software Engineering Research Unit, University of Oulu, Oulu, Finland. He has expertise in quantum software engineering, software process improvement, software architecture, DevOps, AI ethics, agile software development, requirements change management, soft computing, and evidence-based software engineering. He is professionally active in conducting publication-based research workshops, serving as guest editor in main track software engineering journals, and editing software engineering research books. He has published over 90 articles in peer-reviewed international conferences, workshops and journals.

Aakash Ahmad is currently an Assistant Professor at the School of Computing and Communications, Lancaster University Leipzig, Germany. He has received his Ph.D. in Software Engineering from School of Computing, Dublin City University, Ireland in 2015 respectively. Aakash's research and development expertise are in the area of software architecture, software engineering application to mobile and internet of things systems, and cloud computing. He has published in top-ranked international journals such as IEEE Transactions on Software Engineering (TSE), Journal of Systems and Software (JSS), and IEEE Transactions on cloud computing.

Muhammad Waseem is apostdoctoral researcher in the School of Computer Science, Wuhan University, China. He received his Ph.D. in Software Engineering in 2022 from Wuhan University, China. Before pursuing his Ph.D., he had worked as a lecturer and teaching research associate in the Department of Computer Science and Software Engineering, International Islamic University, Pakistan. He received his master degree in software engineering from International Islamic University, Pakistan. His current research interests include software architecture, microservices architecture, and DevOps.

Peng Liang is a full -professor of Software Engineering in the School of Computer Science, Wuhan University, China. His research interests concern the areas of software architecture and requirements engineering. He is a Young Associate Editor of Frontiers of Computer Science, Springer. He has published more than 100 articles in peer-reviewed international journals, conference and workshop proceedings, and books.

Mahdi Fahmideh is a senior lecturer of cyber security at the University of Southern Queensland. He is working on creating artifacts that help digital transformation. His research outcomes can be in the form of methodological approaches, conceptual models, decision-making frameworks, and software tools. He has published in international venues such as the European Journal of Information System (EJIS), IEEE Transactions on Software Engineering (TSE), Information Sciences, Information Systems, Journal of Computers and Industrial Engineering, Information and Software Technology (IST), and Journal of Systems and Software (JSS). He has 8 years of experience in the software industry as an analyst programmer.

Tommi Mikkonen is a full professor of software engineering at the University of Jyväskylä and a full professor of software systems at the University of Helsinki, both located in Finland. In addition to his academic positions, he has been a principal scientist at Nokia and a visiting professor at Sun Microsystems Research and at Mozilla. His research interests include IoT, software development methods, software architecture, multi-device programming, and software engineering for AI. He is the year 2017 Open World Hero, nominated by the Center of Open Source Software in Finland.

Pekka Abrahamsson is currently a full Professor of software engineering at the Faculty of Information Technology and Communication Sciences, Tampere University, Tampere, Finland. His research is in the area of emerging software technologies, empirical software engineering, software startups, and the ethics of artificial intelligence. He is a pioneer in the field of research on agile software engineering methods and processes. Abrahamsson is the most cited researcher in his field in Finland, and he is the first Professor of Software Engineering at the Finnish Academy of Science and Letters. He has published broadly in his areas of expertise and received many awards and recognitions.