Automated Software Engineering (2024) 31:36
https://doi.org/10.1007/s10515-024-00436-x

# Automated quantum software engineering

**Aritra Sarkar**[1,2]

Received: 10 August 2023 / Accepted: 19 March 2024 / Published online: 12 April 2024
© The Author(s) 2024

**Abstract**
As bigger quantum processors with hundreds of qubits become increasingly available, the potential for quantum computing to solve problems intractable for classical computers is becoming more tangible. Designing efficient quantum algorithms and software in tandem is key to achieving quantum advantage. Quantum software engineering is challenging due to the unique counterintuitive nature of quantum logic. Moreover, with larger quantum systems, traditional programming using quantum assembly language and qubit-level reasoning is becoming infeasible. Automated Quantum Software Engineering (AQSE) can help to reduce the barrier to entry, speed up development, reduce errors, and improve the efficiency of quantum software. This article elucidates the motivation to research AQSE (why), a precise description of such a framework (what), and reflections on components that are required for implementing it (how).

**Keywords** Quantum algorithms · Software automation · Program synthesis

## 1 Introduction

Quantum computing (QC) is increasingly gaining focus for stakeholders in high-performance computing. A major research avenue is on maturing the quantum computing hardware in terms of high-fidelity (decoherence, error rates of quantum operations) and scalability (number of qubits, connectivity). While this has proved rather a challenging engineering feat, rapid strides were made in the last decade with a plethora of physical technologies capable of demonstrating controllable processing of quantum information.

✉ Aritra Sarkar
a.sarkar-3@tudelft.nl

[1] Quantum Machine Learning Research Group, Quantum Computing Division, QuTech, Lorentzweg 1, Delft 2628 CJ, The Netherlands

[2] Department of Quantum and Computer Engineering, Delft University of Technology, Mekelweg 4, Delft 2628 CD, The Netherlands

With quantum devices making steady progress, the complementary field of quantum software engineering (QSE) (Zhao 2020; Serrano et al. 2022) is also gaining traction. The field has its roots in the theoretical formulation of quantum information and the earliest quantum algorithms. However, more recently, QSE has been rejuvenated in the light of being integrated within currently available quantum computing pipelines and design methodologies from classical software to be compatible with near-future avatars of quantum processors envisioned in technological road-maps. While the importance of the underlying hardware cannot be understated, it is important that these two fields of hardware and software progress in parallel to prevent a quantum winter scenario where we have large costly quantum devices with no clear understanding of what applications could benefit from it.

Currently, there are three approaches to quantum software development:

$\mathcal{A}_1$: Given the limited capabilities of a specific quantum computing hardware, what useful computing can be implemented on that system?

$\mathcal{A}_2$: Given an industrial use case, how can it be solved using an existing quantum algorithm (with some possible minor tweaking thereof)?

$\mathcal{A}_3$: Designing new quantum algorithms for novel scientific underpinnings (mostly for specific mathematical properties) inspired by the superior (or at least different) computing capabilities of quantum information.

For $\mathcal{A}_1$, the focus is on extracting as much computation power as possible from noisy intermediate-scale quantum (NISQ) (Preskill 2018) devices. The researchers advocate a hardware-software co-design (Shi et al. 2020) approach for the current technology readiness level (TRL) of QC. This involves diluting the abstraction layers of the quantum accelerator stack. These help justify the research funding in quantum computing by demonstrating state-of-the-art proofs-of-concept implementations. However, these highly tuned pipelines become difficult to scale and design, as surveyed in Ezratty (2023). Note that a major focus of experiments involving quantum processors is decoupled from applications and focus on using advanced techniques (like neural networks or pulse shaping) for tuning the building block like coherence time, quantum gates, and control signals. In this article, however, we will not focus on automation for quantum computer engineering, but rather on automating application development on quantum accelerators. In $\mathcal{A}_2$, researchers advocate adhering to strict abstraction layers (Bertels et al. 2020), with separation (Bertels et al. 2021) of concerns between the challenges of hardware (Leymann and Barzen 2020) and software. Thus, the focus is on a proof-of-concept demonstration of the software pipeline, often with a quantum computing simulator as the backend. This principled fashion of organizing the research produces modular designs that are hardware-aware-yet-agnostic and are better aligned with the long-term aims of QSE. While in $\mathcal{A}_3$, the involvement of implementation, either on proof-of-concept QC simulators or real quantum processors, are minimal. The focus is on specific mathematical properties, their proofs of correctness, and derivations of resource complexity bounds.

To make this distinction clear, examples of typical problems addressed by these approaches would be:

$\mathcal{E}_1$: Solving the protein-folding problem on a tetrahedral lattice using a hybrid classical-quantum digitized counterdiabatic algorithm on trapped ions and superconducting quantum processors (Chandarana et al. 2023),

$\mathcal{E}_2$: Implementing a pipeline for satellite image processing using one quantum convolution layer on a neural network architecture (Sebastianelli et al. 2021),

$\mathcal{E}_3$: Proving if quantum computing would provide a superpolynomial speedup in determining the zeta function of a genus curve over a finite field (Kedlaya 2006).

It can be easily appreciated that there exists a considerable gap between these approaches—both in their aim and the domain expertise required to address them.

At this juncture, in this article, we explore promising research directions that will aid in the advancement of QSE. In Sect. 2, the motivation behind automating QSE is elucidated. Section 3 defines the two main perspective of automation: the user interface, or the assessment of quantum advantage. In Sect. 4, theoretical disciplines and tools required to automate QSE are surveyed. Section 5 concludes the article.

## 2 AQSE: Why?

Why do we need to automate quantum software engineering? To understand this, first let's enlist some of the major problems that the field of quantum software engineering faces:

- Quantum mechanics is counter-intuitive to human cognition, even for experts.
- The barrier to entry for quantum algorithm development requires very different training than classical software developers.
- Coverage of statistical testing is not scalable due to exponential state space, and inspecting intermediate states is not feasible due to no-cloning.
- It is currently not possible to deploy realistic use cases on quantum processors, and quantum computing simulators become infeasible at around 50 qubits.
- Similar to data-driven deep learning, the hybrid-quantum-classical algorithms based on variational principles are not interpretable.

In general, there is a need to reduce the barrier to entry for assessing the impact of QC for a use case and develop the quantum accelerated software pipeline.

Barrier to entry can be reduced either with training or by automation. Various educational and industrial institutions are now investing in training the next generation of the quantum workforce (Yakaryilmaz and Delgado 2021; Aiello et al. 2021) via courses, workshops, hackathons, tutorials, popular science articles, etc. The

latter, i.e., automation, is a rather interesting research venture and is the focus of this article.

A primary motivation towards automation is the counter-intuitive nature of the semantic understanding of a quantum algorithm. Typically, graduate-level courses introduce the formalism of quantum mechanics and quantum information. These form the basis for advanced courses and research in quantum algorithms. However, it becomes clear that a phenomenological perspective of quantum algorithms is impossible in the same sense as courses like computer architecture and organization, or Boolean logic design courses are internalized. While superposition can be understood as multiple parallel threads of execution, and measurement can be understood as a weighted random selection of the basis states, (similar to how these are implemented in QC simulators), this is not enough. Gaining quantum advantage depends crucially on orchestrating interference between those threads such that the non-solutions destructively interfere and thereby increase the amplitude of the solution states. Such insights often depend on serendipitous moments for skilled researchers (Shor 2022).

Does that imply that understanding the benefits of QC and building QC-based software solutions would remain the forte of a small circle of researchers? Presenting against this case is the core motivation behind AQSE. AQSE would allow the prototyping and development of software with embedded quantum kernels with little to no knowledge of quantum mechanics. Much like the graphics of a video game automatically gets offloaded by the CPU to the GPU when available, we envision the AQSE framework to utilize quantum computational resources automatically.

In what follows, we will define AQSE and contrast it with similar approaches.

## 3 AQSE: What?

Let us define automated quantum software engineering (AQSE) as: 'a framework capable of synthesizing a quantum computing solution for a given application.' The deliberate vagueness will be discussed and gradually refined in this section. At its finest form, AQSE would take in user requirements and produce a quantum computing implementation that would be a valid solution that the user can plug into an existing software pipeline and reap the benefits. With that moonshot in mind, let us understand two important aspects of AQSE.

### 3.1 Usability of the framework

Based on our motivation, the AQSE framework must conform to ease of use. We will consider two aspects of ease: the user interface and the level of vagueness/rigor in the problem specification.

The barrier to entry to the use of software can be frugally reduced by having a graphical user interface (GUI). The evolution of most software bear testimony to this trend, from operating systems to programming environments. While most application software has GUI, visual programming languages (VPL) have not been as popular.

Tools exist to easily design such interfaces for a code (e.g. in Python) at the back end. Current quantum tools are mostly developed by researchers for fellow researchers with considerable backgrounds in setting up programming platforms. Thus, efforts on these are often considered superfluous. An intuitive user interface would go a long way in lowering the barrier to entry. A few commercial/educational quantum platform providers are considering this more seriously. These include Quirk, IBM Quantum Composer, Quantum Inspire, qBraid, Strangeworks, Elyah, Notate (Arawjo et al. 2022), etc. However, there is a crucial difference between these and the AQSE requirements. These platforms aim to provide a quantum integrated development environment (IDE), aiding researchers in setting up a cloud computing environment, interfacing with various quantum hardware and simulator platforms, visualizing the results, and managing the execution logs. We propose focusing on a Low Code, and eventually a No Code Development Platform (NCDP) for AQSE.

The problem specification interfaces the intent of the user with the automation engine. NCDP alone would not make quantum accelerators more accessible if it involves drag-and-drop unitary gates, as with all current QC VPLs. Thus, this involves a different modality of AQSE. The problem specification should abstract the details of quantum information processing and focus on the functional or behavioral problem definition. Since the quantum details are no longer visible to the user, the interface should not look very different from similar tools on classical computing platforms. Thus, in many aspects, it will be similar to a no-code AI or AutoML. These tools decouple programming languages and syntax from logic and instead take a visual approach to software development to enable rapid delivery. No-code AI with the additional capability of reasoning in quantum logic and synthesizing quantum software is the vision of AQSE.

Do such tools exist? Certainly not in the quantum software engineering space. NCDP is more common for simple situations like web development, mobile apps, and game logic (visual scripting). An intermediate solution would be graphical node/flow-based programming interfaces like Simulink. The blocks can be specified at various levels of abstraction, e.g., a database query application, a quantum search algorithm, a Grover diffusion block, a multi-controlled Z gate, or a native gate/pulse for specific quantum hardware. More recently, the proliferation of large language models (LLM) for coding is considerably boosting the usability (Cai et al. 2023; Romera-Paredes et al. 2023) and efficiency of software engineering. Very likely, such prompt-based interfaces will soon be integrated within the leading quantum software frameworks like Qiskit or Azure Quantum. We will delve more into these levels of interfaces in the next section.

### 3.2 Assessment of applicability

The intentional software (Simonyi et al. 2006) development paradigm, for better or worse, abstracts away the quantization of the desired solution. Thus, it is paramount to understand when quantum computation is useful in the first place, based on the user specification. In the broadest sense, this in itself is the core business idea of many software consultancy companies in the quantum technology space. AQSE

might not be able to be so versatile, and such a feature might only apply to a well-specified problem definition.

There is some well-understood domain knowledge that can aid in this process of applicability analysis. Quantum computation is among the only known violation (Bernstein and Vazirani 1993) of the complexity-theoretical Church-Turing thesis (CTT) that is allowed by our current laws of physics. There exist the complexity class, called bounded-error quantum polynomial time (BQP), that includes problems that are faster on a quantum model of computation (typically proved using a quantum Turing machine) However, there are a few subtleties that need to be unpacked in such theoretical underpinning:

1. The corresponding classical complexity classes are bounded-error probabilistic polynomial time (BPP) and the polynomial time (P) classes that are efficient on a classical probabilistic/deterministic Turing machine, respectively. Thus, the focus of studying BQP problems is to rather identify problems in BQP\BPP or BQP\P region of computational time complexity. Our knowledge of such problems includes only a few examples, although they are the shining gems of quantum algorithms. Some of the early quantum algorithms like Deutsch-Josca, Bernstein-Vazirani, Simon's problems, Forrelation are about mathematical properties. Algorithms with more practical motivations include Shor's discrete logarithm, Shor's factorization, and the HHL algorithm for solving linear equations.
2. Quantum computation does not solve an expanded set of functions, i.e., they are at the same degree of Turing computability. This means it is not a strict violation of the CTT, only of its extended version. This allows any quantum computation to also be expressed at classical computation, which forms the basis of QC simulators.
3. There are many classical universal models of computation, e.g., Turing machines, cellular automata, Post machines, lambda calculus, Wang tiles, etc. These models are equivalent to each other within a polynomial time overhead. Similarly, there are universal models of quantum computation, like quantum Turing machines, quantum cellular automata, quantum lambda calculus, adiabatic quantum computing, measurement-based quantum computing, and the canonical quantum circuit/gate model. These are all related in similar ways to each other.
4. Points (2.) and (3.) mean any requirement specified to AQSE can be translated to both a classical and quantum implementation. The code structure at the computability level cannot guide the choice of a quantum implementation, which makes such a choice difficult. However, it also makes it interesting, as now each code needs to be assessed more intelligently in a broader context to understand its suitability of quantum acceleration. For example, an arithmetic operation would not provide a speedup by translation to reversible logic but becomes imperative if it is part of a quantum algorithm that manipulates superposition states.
5. Many industrial and social computational issues can be formulated as problems that belong to the non-deterministic polynomial time (NP) class (or rather strictly in NP\P class). It is believed that QC will not be able to exactly solve NP problems efficiently (i.e., in polynomial time) under realistic assumptions (e.g., P≠ NP).

6. The quantum Grover search provably provides a quadratic speedup for unstructured database search. Almost any problem can be posed as a search problem over a solution domain, e.g., factoring can be a search over numbers that, on multiplication, equals the result. Similarly, all problems in the NP class can be posed as a search problem based on the constraints' satisfiability (SAT) since SAT is NP-complete.
7. Points (5.) and (6.) are the main reasons why we witness such proliferating attempts to formulate NP-hard problems as quantum algorithms. While these do not aim for an exponential speedup of Point (1.), just solving on a quantum model might allow speedup because it is a different form of computational automata. This latter case is particularly the motivation for quantum annealing (where quantum tunneling can be beneficial for some specific optimization landscapes over thermal fluctuations) and Boson sampling. Thus, understanding the formal model of computation is important, as a quantum search on adiabatic quantum computing (AQC) would perform badly compared to a quantum circuit model, similar to how simulating Game of Life on a Turing machine would perform poorly compared to a cellular automata substrate.
8. Besides these complexity theoretic viewpoints, it is important to realize that there are many problems where time complexity is not the major driver. Such problems are particularly studied in machine learning (ML) and focus on space complexity (Ventura and Martinez 1998), generalization, representation capacity, pattern recognition (Schuld and Killoran 2019), etc. Thus, holistically the quantum solution needs to be assessed against other computational resources and metrics like memory requirements, convergence rate, solution accuracy, etc.

These considerations imply that, though the interface of the No-code AI of AQSE would look welcoming to users, the underlying automation engine needs to be founded in rigorous mathematical principles to even assess the applicability, let alone design the quantum solutions. At a superficial level, this seems as if they are at odds with each other since the functional level description is about abstracting away resource details, while resource estimates are crucial to assess the applicability. This is the core innovation that AQSE needs to address via increasing rigorous levels of abstraction and specification.

In the following section, we will clarify how understanding the resource advantage of quantum software and synthesizing quantum software from requirements are the same problem from two different perspectives.

## 4 AQSE: How?

Having presented the overall goal of AQSE for the external interface and internal engine, in this section, we will delve deeper into components that will be necessary for the internal engine.

This is perhaps the right moment to clarify that AQSE is neither an esoteric nor a novel venture. There have been some attempts in the past to automate quantum algorithm design. As early as 2004, a book (Spector 2004) titled Automatic

Quantum Computer Programming discussed evolutionary approaches for discovering novel quantum algorithms. More recently, in the ongoing quantum computer engineering revolution, a few academic and commercial groups are pursuing this same goal. Three of the most notable toolsets are discussed here, however, there are many individual researchers whose theses are aligned with AQSE. Munich Quantum Toolkit (MQT) includes a set of tools for design automation in QC (Zulehner and Wille 2020). Most relevant to AQSE is the MQTProblemSolver (Quetschlich et al. 2022), aimed at users with little to no quantum computing knowledge. Provided with a problem description (as a constraint satisfaction problem (CSP) or graph optimization like TSP), it offers a selection of implemented quantum algorithms. The user's choice triggers an encapsulated quantum calculation, and the solution is returned in the standard classical format. Horizon Quantum Computing aims to democratize quantum computing applications for businesses by removing the need for quantum algorithms knowledge for software developers. It features a compiler that automatically constructs quantum algorithms from classical code. Their patent (Fitzsimons and Tan 2021) and public presentations reveal a layered approach for various levels of synthesis. The top layer, called Carbon, has many of the desirable AQSE properties of applicability assessment. Another company, Classiq Technologies, aims to revolutionize the process of developing quantum computing software. Their software platform transforms high-level functional models into optimized quantum circuits, allowing quick development of large qubit circuits and execution on any gate-based system. For example, it can generate the quantum circuit for initializing a state given a specific probability distribution while optimizing for circuit depth, width, or precision. They hold a couple of patents (Naveh et al. 2021, 2021, 2021) on their offering that concerns AQSE. The core of their inspiration, like MQT, is to repurpose methods from classical computer-aided design (CAD) in very large-scale integration (VLSI) logic design for quantum circuits. Discussions on the specifics of these tools and others (like AlgebraicJulia (Brown et al. 2022), DisCoCirc (Coecke 2021), SilQ (Bichsel et al. 2020), Qrisp (Seidel et al. xxx), AdaQuantum (Nichols et al. 2019), Klaus (Cervera-Lierta et al. 2022), Wolfram Quantum Framework (Gorard et al. 2021), etc.) will be introduced in the respective components.

The AQSE engine is essentially a stack of abstraction layers connecting an implementation to a user intent. Here we present some components and refinement levels that will be crucial to develop AQSE.

### 4.1 User intent to application specification

Foremost, AQSE requires inputting the user intent. Very broadly, this can either be (i) an extrinsic objective or (ii) an intrinsic motivation. The latter case involves approaches like novelty search, which can eventually aid in the automated discovery of quantum algorithms and their corresponding purpose. This is a rather niche field and has mostly been explored in the context of robotics (Wang et al. 2020; Colas et al. 2022). However, a similar framework can be applied to (quantum) program synthesis. We will not discuss this here in detail and will focus on objective-driven AQSE.

The specification language for the objective determines the level of vagueness allowed. A well-specified software is equivalent in complexity to the program it translates to. Trading off to higher vagueness and abstraction translates to a larger solution space. This also implies a certain degree of freedom, and any solution from the larger space is assumed to satisfy the requirement of the user. In many synthesis frameworks, the specification is iteratively refined by presenting behavioral examples to the user, eventually scoping the correct bounds of the problem space.

The key aspect of interfacing with the user, as discussed in the previous section, is a classical NCDP that abstracts quantum information processing as well as programming syntax.

### 4.2 Formalizing application specification

At the high end of the vagueness spectrum, we already witness the proliferation of natural language-driven coding, e.g., using OpenAI's Codex (Chen et al. 2021) based on a modification of GPT framework. These are based on an enormous corpus of training data, which might not be readily available for quantum computation. However, Codex and Qiskit have already shown some initial promising results. Recently, LLMs like generative pre-trained transformers (GPT) have been used for quantum architecture search (Liang et al. 2023), which potentially will become more ubiquitous in other areas of QSE. A more sustainable and explainable abstraction would be to refine the natural language to a formal specification language which can be further processed downstream in a controlled fashion.

A slightly higher structure is obtained in specification based on pseudo-code or LaTeX. LaTeX to Python code converters already exists for mathematical equations. Such tools can be handy specifically for optimization use cases based on SAT solvers, which can readily be translated to QUBO and thereafter to variational algorithms like QAOA (Bakó et al. 2022) or quantum annealing. These can be integrated with frameworks like SilQ and Qrisp to enhance user accessibility. Similarly, software design frameworks like UML have also been extended to Q-UML (Pérez-Castillo et al. 2021), which can be integrated into AQSE's NCDP.

AQSE should also retain the current level of specification at the QASM or embedded domain-specific language (DSL) level. These include cQASM, OpenQASM, Qiskit, Q#, OpenQL, etc. In the NISQ era, this would also allow advanced users to specify non-functional requirements, like noise level, connectivity topology of hardware, qubit multiplicity, etc. However, the focus on AQSE is on future generation of quantum processors where the end-user is not concerned with these low-level details, and can focus on algorithm development.

The right level of requirement specification is, of course, formal specification languages, like Z notation (Object Z, Z++) (Cartiere 2022) or B method. Another alternative that has a low level of obscure syntax is logic programming languages like Prolog. However, these tools have a steep learning curve and are unknown to most software developers, let alone quantum physics researchers. Thus, the refinement

to this level of abstraction must be encapsulated by AQSE. We need to derive two things at this level, the functionality, and how to test/qualify it and thus bind the intention and validation aspects. The validation can either be analytical or a set of test examples. The AQSE NCDP would output a classical formal specification of the user requirements.

### 4.3 Formal specification to formal logic

Formal specification languages can be easily refined to 1st order predicate logic, or proof obligations for interactive theorem provers (ITP) like Coq, Aqda, LEAN, etc. The crucial aspect at this stage is to choose the formal logic to express the axioms, theorems, and the validity of proof entailments. While classical or intuitionist logics are typically the default choices, in quantum it is worthwhile to use linear logic (Girard 1987), which nicely captures the no-cloning of quantum information. The corresponding language to express the logic is the dagger-lambda calculus (Atzemoglou 2014). However, this needs further exploration and consideration for other candidates like modal logic, temporal logic, computational tree logic* (CTL*), and many-value logic (e.g., paraconsistent logic (Goertzel 2021)).

### 4.4 Solution representation

The synthesized artifact that is gradually constructed by the AQSE engine needs to be represented and stored. Typically in formal logic, proofs are represented in normal form (natural deduction) or tree form. Based on the logic used, other options like Kripke semantics, sequent form, etc. can also be explored. A natural way to store and explore proofs is via proof nets (Girard et al. 1989), in a graph data structure. This allows easy manipulation, rewriting, probabilistic reasoning, etc. using already well-developed libraries in most programming languages. Proof trees can alternatively be replaced by abstract syntax trees (AST) or abstract semantic graphs (ASG).

### 4.5 Search space representation

Once a potential solution is represented as a proof net (or, AST/ASG), it can be related to other solutions. This can be via a meta-graph structure, where the edges represent the relation between the solution (e.g., one requires a qubit less, while the other requires 5 CNOT gates more). In expressing relations (instead of functions), it is often desirable to represent a group of solutions that has a certain property. Thus, we suggest using a generalized meta graph with hyper-edges, as the search space representation. Similar constructs are used in the Wolfram Quantum Framework (Gorard et al. 2020) and the OpenCog Hyperon (Goertzel 2021) AGI cognitive architecture.

### 4.6 Synthesis method

The synthesis of valid solutions and their corresponding estimation of computational resources is the core of the AQSE engine. The refinement stages presented above, between the user's intent and synthesis, preserve explainability and verifiability. However, a black box solution that skips these stages, e.g., a trained deep neural network, would also qualify as a short-sighted AQSE implementation. There are various methods for program synthesis (or, in this case, proof/AST/ASG synthesis). Some of the most promising methods for quantum algorithms are listed here, in increasing order of sophistication required to implement them. On one end, search-based enumeration incorporates sophisticated heuristics like learning, evolution, and symbolic logic. On the other end, formal logic dilutes to incorporate probabilistic and inductive approximations. A trade-off between these approaches needs to balance tractability and accuracy with interpretability.

#### 4.6.1 Automated program search

Automated program search uses various techniques to generate solutions via heuristic methods and assess their applicability. It can either be exhaustive or pruned to preserve the tractability of the search while often sacrificing the guarantee of a global optimum.

- Exhaustive enumeration: is easiest to implement; however, the entailment graph grows exponentially and becomes intractable beyond small instances. Formalisms like Nielsen geometry (Nielsen et al. 2006) and uncomplexity metric (Brown et al. 2021) need to be incorporated to guide the search process. These techniques are used in quantum optimal control for NISQ devices.
- Template-based meta-programming: can be used for small instances (or, holes in program synthesis) to fine-tune a code this is already very close to an acceptable solution. However, this is not specifically the goal of AQSE.
- Evolutionary approach: genetic programming (Spector 2004) based solutions can be easily integrated with ProofNet/AST/ASG using linear logic. Other evolutionary approaches like novelty search (Lehman et al. 2011) and gene expression programming (Alvarez et al. 2023) might prove useful.
- Artificial neural networks (ANN): and deep reinforcement learning (DRL) has been successfully applied in many cases on program learning, including the recent success of AlphaTensor (Fawzi et al. 2022) and AlphaDev (Mankowitz et al. 2023). These techniques can be readily applied to low-level quantum algorithms.
- Foundational models: based on deep neural networks, are becoming popular tools for program induction. We have already mentioned LLM-based solutions (Liang et al. 2023). Machine learning for synthesizing a desired unitary transformation as a quantum circuit has also been demonstrated in AutoQC (Murakami and Zhao 2022) and QSeed (Weiden et al. 2023).

- Neuro-evolution: algorithms like NEAT (Stanley and Miikkulainen 2002) and its later upgrades decouple the hyperparameter tuning and neural architecture search to an evolutionary heuristic.
- Neuro-symbolic approach: trades off between the explainability of symbolic AI with the efficiency of ANNs, and is specifically suited for symbolic regression tasks like theorem proving.
- Quantum reinforcement learning: allows learning the solution given access to the environment or its model. These techniques are explored in Hamiltonian learning, projective simulation (PS) (Saggio et al. 2021), quantum knowledge seeking agent (QKSA) (Sarkar 2022), and quantum photonics setups like Melvin (Krenn et al. 2016), AdaQuantum (Nichols et al. 2019) and Klaus (Cervera-Lierta et al. 2022). For AQSE, the environment can be a real quantum device, a quantum computing simulator, or the set a corpus of input–output training sets (called programming-by-example). PS, QKSA, and Klaus are particularly interesting as these incorporate symbolic AI that allows interpretable solutions.

#### 4.6.2 Automated theorem proving

Automated theorem proving (ATP) is based on the Curry-Howard correspondence (Baez and Stay 2010) between mathematical proofs and programs on universal automata. Proof refinement creates a procedure as a byproduct while proving the validity of the quantum algorithm. While these techniques have been explored in theoretical circles, the proliferation to QSE is due.

- Deductive proofs: are typically what is common in ATP. Some corresponding quantum solutions for expressing quantum proofs already exist, like QWIRE (Paykin et al. 2017), SQIR (Hietala et al. 2020), CoqQ (Zhou et al. 2022), LQP, QHL, etc.
- Categorical quantum mechanics: is a diagrammatic language for formal reasoning in quantum information. Tools like DisCoCirc (Coecke 2021), ZX-calculus, Quantomatic, and Catlab.jl (Brown et al. 2022) can be used for the refinement of ASG to quantum programs. Research is needed in computational category theory for applied sciences (in contrast to applied category theory, which focuses on formalizing and understanding applied sciences rather than proactive computational development).
- Probabilistic proofs: allows uncertainties (Nori et al. 2015) in the user specification to trickle down to formal synthesis in a controlled manner. Tools like Markov logic networks (Richardson and Domingos 2006) and probabilistic logic programming (Goertzel et al. 2008) (e.g., ProbFOIL (De Raedt et al. 2015)) can be upgraded to incorporate quantum logic.
- Inductive proofs: allow generating solutions from incomplete specifications. Similar concepts have been studied in the quantization (Arunachalam and Wolf 2017) of probably-approximate correct (PAC) in learning theory. However, inductive tactics and approximations (Bornholt et al. 2015; Andriushchenko et al. 2021) need to be incorporated in quantum formal proofs.

We expect that future implementation of an AQSE framework would most likely be a subset of these features. However, it is crucial to comprehensively evaluate (Gulwani et al. 2017) the applicability of at least (and most likely, more of) these techniques in the context of AQSE.

#### 4.6.3 Concept discovery and component-based development

While specification-based synthesis would be required for novel applications, most quantum applications would depend of a small set of quantum kernels/components with resource advantages over corresponding classical components. Thus, incorporating component-based software engineering (CBSE) principles in quantum algorithm automation (Kang and Oh 2023) can lead to more scalable and maintainable quantum software, making it easier to develop, test, and optimize quantum algorithms for various applications. A major step for CBSE is identifying quantum algorithm components that can be reused to develop solutions. Such concepts for quantum compilation passed are often pre-designed, e.g., for circuit optimization, routing, and error correction. Quantum algorithm components are currently available as libraries within programming frameworks or cloud platforms (Martyniuk et al. 2021). Augmenting the set of useful components can be automated via concept discovery. Recent works apply statistical metrics of quantum gates in circuits to define reusable modules (Cruz-Lemus et al. 2021; Heese et al. 2023; Sarra et al. 2023).

### 4.7 Formal verification

Formal verification is baked in the AQSE engine and represents a complementary research direction to stochastic verification (Wang et al. 2021). It is, however, important that the formal proof of correctness also remain inspectable and interpretable to end users. Tools that translate proofs to natural language, e.g. Coqatoo (Bedford 2017), can be used for such purposes.

### 4.8 Hardware specific non-functional requirements

Most available quantum processors are universal, in the sense that they have a defined set of native quantum universal gates. However, the exact implementation cost depends on various factors like decoherence time, gate errors, qubit connectivity topology, control system multiplexing, etc. In this article, we focused on the functional aspects of AQSE, with a theoretical pareto-optimization of quantum computing resources. Low-level cost estimation are available in many available compilers which can be plugged into AQSE's synthesis cost estimator to specialize the framework for target hardware.

## 5 Conclusion

This survey of the current state of quantum software engineering and the need for automation is intended to not only scope out the research field around the topic of automated discovery of quantum algorithms but also a pragmatic research proposal and call-to-action for multidisciplinary researchers working on allied fields.

In our opinion, the correct vision to reduce the barrier to entry for quantum application development requires automation in tandem with education. To achieve this, we need to reevaluate the rich spectrum of techniques that are used in other fields and tune them for quantum logic and physics. We want to highlight the crucial similarity between the quantum-classical interface of controllability and measurement, and that of interpretability and efficiency of AI models. This similarity hints that AQSE would share some of the same successes and drawbacks as of the deep learning revolution and, eventually, would have to focus on hybrid approaches like neuro-symbolic techniques.

AQSE is by no means an easy task. While AQSE can be critiqued as being futuristic and not applicable to current NISQ devices, it is important to look beyond the immediate needs and extrapolate the growth and needs of the quantum software industry a decade from now. We envision that, AQSE will be integrated within quantum software frameworks independent of the advancement of quantum processors. Fascinatingly, AQSE would lead to exploring the limits of intelligence systems compared to what human experts can achieve.

**Author contributions** A.S. wrote the main manuscript text.

**Funding** Not applicable.

### Declarations

**Conflict of interest** Not applicable.

**Consent for publication** All authors consent to the publication.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

## References

Aiello, C.D., Awschalom, D.D., Bernien, H., Brower, T., Brown, K.R., Brun, T.A., Caram, J.R., Chitambar, E., Di Felice, R., Edmonds, K.M.: Achieving a quantum smart workforce. Quantum Sci. Technol. **6**(3), 030501 (2021)

Alvarez, G., Bennink, R., Irle, S., Jakowski, J.: Gene expression programming for quantum computing. arXiv preprint arXiv:2303.08203 (2023)
Andriushchenko, R., Češka, M., Junges, S., Katoen, J.-P., Stupinskỳ, Š.: Paynt: a tool for inductive synthesis of probabilistic programs. In: International Conference on Computer Aided Verification, 856–869. Springer, Cham (2021)
Arawjo, I., DeArmas, A., Roberts, M., Basu, S., Parikh, T.: Notational programming for notebook environments: A case study with quantum circuits. In: Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology, 1–20 (2022)
Arunachalam, S., Wolf, R.: Guest column: a survey of quantum learning theory. ACM SIGACT News **48**(2), 41–67 (2017)
Atzemoglou, P.: The dagger lambda calculus. arXiv preprint arXiv:1406.1633 (2014)
Baez, J., Stay, M.: Physics, topology, logic and computation: a rosetta stone. In: New Structures for Physics, 95–172. Springer, Heidelberg (2010)
Bakó, B., Glos, A., Salehi, Ö., Zimborás, Z.: Near-optimal circuit design for variational quantum optimization. arXiv preprint arXiv:2209.03386 (2022)
Bedford, A.: Coqatoo: generating natural language versions of coq proofs. arXiv preprint arXiv:1712.03894 (2017)
Bernstein, E., Vazirani, U.: Quantum complexity theory. In: Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing, 11–20 (1993)
Bertels, K., Sarkar, A., Hubregtsen, T., Serrao, M., Mouedenne, A.A., Yadav, A., Krol, A., Ashraf, I.: Quantum computer architecture: Towards full-stack quantum accelerators. In: 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), 1–6 (2020). IEEE
Bertels, K., Sarkar, A., Ashraf, I.: Quantum computing: from NISQ to PISQ. IEEE Micro **41**(5), 24–32 (2021)
Bichsel, B., Baader, M., Gehr, T., Vechev, M.: Silq: A high-level quantum language with safe uncomputation and intuitive semantics. In: Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation, 286–300 (2020)
Bornholt, J., Torlak, E., Ceze, L., Grossman, D.: Approximate program synthesis. In: Workshop on Approximate Computing Across the Stack (2015)
Brown, A.R., Freedman, M.H., Lin, H.W., Susskind, L.: Effective geometry, complexity, and universality. arXiv preprint arXiv:2111.12700 (2021)
Brown, K., Hanks, T., Fairbanks, J.: Compositional exploration of combinatorial scientific models. arXiv preprint arXiv:2206.08755 (2022)
Cai, Y., Mao, S., Wu, W., Wang, Z., Liang, Y., Ge, T., Wu, C., You, W., Song, T., Xia, Y., et al.: Low-code llm: Visual programming over llms. arXiv preprint arXiv:2304.08103 (2023)
Cartiere, C.R.: Formal methods for quantum software engineering. In: Quantum Softw. Eng., pp. 85–101. Springer, Cham (2022)
Cervera-Lierta, A., Krenn, M., Aspuru-Guzik, A.: Design of quantum optical experiments with logic artificial intelligence. Quantum **6**, 836 (2022)
Chandarana, P., Hegade, N.N., Montalban, I., Solano, E., Chen, X.: Digitized counterdiabatic quantum algorithm for protein folding. Phys. Rev. Appl. **20**(1), 014024 (2023)
Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.d.O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al.: Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021)
Coecke, B.: The mathematics of text structure. In: J. Lambek (eds) The Interplay of Mathematics. Logic, and Linguistics, pp. 181–217. Springer, Cham (2021)
Colas, C., Karch, T., Sigaud, O., Oudeyer, P.-Y.: Autotelic agents with intrinsically motivated goal-conditioned reinforcement learning: a short survey. J. Artif. Intell. Res. **74**, 1159–1199 (2022)
Cruz-Lemus, J.A., Marcelo, L.A., Piattini, M.: Towards a set of metrics for quantum circuits understandability. In: International Conference on the Quality of Information and Communications Technology, 239–249. Springer, Cham (2021)
De Raedt, L., Dries, A., Thon, I., Broeck, G., Verbeke, M.: Inducing probabilistic relational rules from probabilistic examples. In: Twenty-fourth International Joint Conference on Artificial Intelligence (2015)
Ezratty, O.: Where are we heading with nisq? arXiv preprint arXiv:2305.09518 (2023)
Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., Novikov, A., Ruiz, R., F.J., Schrittwieser, J., Swirszcz, G.: Discovering faster matrix multiplication algorithms with reinforcement learning. Nature **610**(7930), 47–53 (2022)

Fitzsimons, J.F., Tan, S.-H.: Systems and methods for unified computing on digital and quantum computers. Google Patents. US Patent App. 17/337,873 (2021)
Girard, J.-Y.: Linear logic. Theor. Comput. Sci. **50**(1), 1–101 (1987)
Girard, J.-Y., Taylor, P., Lafont, Y.: Proofs and Types, vol. 7. Cambridge University Press, Cambridge, UK (1989)
Goertzel, B.: Paraconsistent foundations for quantum probability. arXiv preprint arXiv:2101.07498 (2021)
Goertzel, B.: Reflective metagraph rewriting as a foundation for an agi" language of thought". arXiv preprint arXiv:2112.08272 (2021)
Goertzel, B., Iklé, M., Goertzel, I.F., Heljakka, A.: Probabilistic Logic Networks: A Comprehensive Framework for Uncertain Inference. Springer, New York (2008)
Gorard, J., Namuduri, M., Arsiwalla, X.D.: Zx-calculus and extended hypergraph rewriting systems i: A multiway approach to categorical quantum information theory. arXiv preprint arXiv:2010.02752 (2020)
Gorard, J., Namuduri, M., Arsiwalla, X.D.: Zx-calculus and extended wolfram model systems ii: fast diagrammatic reasoning with an application to quantum circuit simplification. arXiv preprint arXiv:2103.15820 (2021)
Gulwani, S., Polozov, O., Singh, R., : Program synthesis. Found. Trends® Programm. Lang. **4**(1-2), 1–119 (2017)
Heese, R., Gerlach, T.T., Mücke, S., Müller, S., Jakobs, M., Piatkowski, N.: Explaining quantum circuits with shapley values: Towards explainable quantum machine learning (2023)
Hietala, K., Rand, R., Hung, S.-H., Li, L., Hicks, M.: Proving quantum programs correct. arXiv preprint arXiv:2010.01240 (2020)
Kang, C.G., Oh, H.: Modular component-based quantum circuit synthesis. Proc. ACM Programm. Lang. **7**(OOPSLA1), 348–375 (2023)
Kedlaya, K.S.: Quantum computation of zeta functions of curves. Comput. Complex. **15**, 1–19 (2006)
Krenn, M., Malik, M., Fickler, R., Lapkiewicz, R., Zeilinger, A.: Automated search for new quantum experiments. Phys. Rev. Lett. **116**(9), 090405 (2016)
Lehman, J., Stanley, K.O.: Novelty search and the problem with objectives. In: Genetic Programming Theory and Practice IX, 37–56. Springer, Cham (2011)
Leymann, F., Barzen, J.: The bitter truth about gate-based quantum algorithms in the NISQ era. Quant. Sci. Technol. **5**(4), 044007 (2020)
Liang, Z., Cheng, J., Yang, R., Ren, H., Song, Z., Wu, D., Qian, X., Li, T., Shi, Y.: Unleashing the potential of llms for quantum computing: A study in quantum architecture design. arXiv preprint arXiv:2307.08191 (2023)
Mankowitz, D.J., Michi, A., Zhernov, A., Gelmi, M., Selvi, M., Paduraru, C., Leurent, E., Iqbal, S., Lespiau, J.-B., Ahern, A.: Faster sorting algorithms discovered using deep reinforcement learning. Nature **618**(7964), 257–263 (2023)
Martyniuk, D., Falkenthal, M., Karam, N., Paschke, A., Wild, K.: An analysis of ontological entities to represent knowledge on quantum computing algorithms and implementations. In: Qurator (2021)
Murakami, K., Zhao, J.: Autoqc: Automated synthesis of quantum circuits using neural network. arXiv preprint arXiv:2210.02766 (2022)
Naveh, Y., Naveh, A., Minerbi, N., Kirzner, O., Goldfeld, A., Ur, S.: Quantum circuit modeling. Google Patents. US Patent App. 17/149,326 (2021)
Naveh, A., Ur, S., Naveh, Y., Kirzner, O., Alon, R., Goren, T., Goldfeld, A., Minerbi, N.: CSP-based synthesis of a quantum circuit. Google Patents. US Patent App. 17/499,046 (2021)
Naveh, A., Ur, S., Naveh, Y., Kirzner, O., Alon, R., Goren, T., Minerbi, N.: Re-generation of a gate-level quantum circuit based on gate-level analysis. Google Patents. US Patent App. 17/499,063 (2021)
Nichols, R., Mineh, L., Rubio, J., Matthews, J.C., Knott, P.A.: Designing quantum experiments with a genetic algorithm. Quantum Sci Technol **4**(4), 045012 (2019)
Nielsen, M.A., Dowling, M.R., Gu, M., Doherty, A.C.: Quantum computation as geometry. Science **311**(5764), 1133–1135 (2006)
Nori, A.V., Ozair, S., Rajamani, S.K., Vijaykeerthy, D.: Efficient synthesis of probabilistic programs. ACM SIGPLAN Notices **50**(6), 208–217 (2015)
Paykin, J., Rand, R., Zdancewic, S.: Qwire: a core language for quantum circuits. ACM SIGPLAN Notices **52**(1), 846–858 (2017)
Pérez-Castillo, R., Jiménez-Navajas, L., Piattini, M.: Modelling quantum circuits with uml. In: 2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE), 7–12 (2021). IEEE

Preskill, J.: Quantum computing in the NISQ era and beyond. Quantum **2**, 79 (2018)
Quetschlich, N., Burgholzer, L., Wille, R.: Towards an automated framework for realizing quantum computing solutions. arXiv preprint arXiv:2210.14928 (2022)
Richardson, M., Domingos, P.: Markov logic networks. Mach. Learn. **62**(1), 107–136 (2006)
Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M.P., Dupont, E., Ruiz, F.J.R., Ellenberg, J.S., Wang, P., Fawzi, O., Kohli, P., Fawzi, A.: Mathematical discoveries from program search with large language models. Nature (2023)
Saggio, V., Asenbeck, B.E., Hamann, A., Strömberg, T., Schiansky, P., Dunjko, V., Friis, N., Harris, N.C., Hochberg, M., Englund, D.: Experimental quantum speed-up in reinforcement learning agents. Nature **591**(7849), 229–233 (2021)
Sarkar, A.: Applications of quantum computation and algorithmic information: for causal modeling in genomics and reinforcement learning (2022)
Sarra, L., Ellis, K., Marquardt, F.: Discovering quantum circuit components with program synthesis. arXiv preprint arXiv:2305.01707 (2023)
Schuld, M., Killoran, N.: Quantum machine learning in feature hilbert spaces. Phys. Rev. Lett. **122**(4), 040504 (2019)
Sebastianelli, A., Zaidenberg, D.A., Spiller, D., Le Saux, B., Ullo, S.L.: On circuit-based hybrid quantum neural networks for remote sensing imagery classification. IEEE J. Select. Topics Appl. Earth Observ. Remote Sens. **15**, 565–580 (2021)
Seidel, R., Bock, S., Tcholtchev, N., Hauswirth, M.: *Qrisp*: a framework for compliable high-level programming of gate-based quantum computers. PlanQC-Programm. Lang. Quantum Comput. (2022)
Serrano, M.A., Pérez-Castillo, R., Piattini, M.: Quantum Software Engineering. Springer, Cham (2022)
Shi, Y., Gokhale, P., Murali, P., Baker, J.M., Duckering, C., Ding, Y., Brown, N.C., Chamberland, C., Javadi-Abhari, A., Cross, A.W.: Resource-efficient quantum computing by breaking abstractions. Proc. IEEE **108**(8), 1353–1370 (2020)
Shor, P.W.: The early days of quantum computation. arXiv preprint arXiv:2208.09964 (2022)
Simonyi, C., Christerson, M., Clifford, S.: Intentional software. In: Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications, 451–464 (2006)
Spector, L.: Automatic Quantum Computer Programming: A Genetic Programming Approach, vol. 7. Springer, New York (2004)
Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002)
Ventura, D., Martinez, T.: Quantum associative memory with exponential capacity. In: 1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98CH36227), **1**, 509–513 (1998). IEEE
Wang, X., Arcaini, P., Yue, T., Ali, S.: Quito: a coverage-guided test generator for quantum programs. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), 1237–1241 (2021). IEEE
Wang, R., Lehman, J., Rawal, A., Zhi, J., Li, Y., Clune, J., Stanley, K.: Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions. In: International Conference on Machine Learning, 9940–9951 (2020). PMLR
Weiden, M., Younis, E., Kalloor, J., Kubiatowicz, J., Iancu, C.: Improving quantum circuit synthesis with machine learning. arXiv preprint arXiv:2306.05622 (2023)
Yakaryilmaz, A., Delgado, A.: QWorld: Inviting everyone to be part of the second quantum revolution. APS March Meet. Abstracts **2021**, A29-011 (2021)
Zhao, J.: Quantum software engineering: Landscapes and horizons. arXiv preprint arXiv:2007.07047 (2020)
Zhou, L., Barthe, G., Strub, P.-Y., Liu, J., Ying, M.: Coqq: Foundational verification of quantum programs. arXiv preprint arXiv:2207.11350 (2022)
Zulehner, A., Wille, R.: Introducing Design Automation for Quantum Computing, vol. 11. Springer, Cham (2020)

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.