Quantum Software Engineering: Something Old, Something New; Something Borrowed, Something Blue

Jose Garcia-Alonso[1], Majid Haghparast[2], Tommi Mikkonen[2,*], Juan Manuel Murillo Rodríguez[1] and Vlad Stirbu[2]

[1] *University of Extremadura, Spain*
[2] *University of Jyväskylä, Finland*
E-mail: jgaralo@unex.es; majid.m.haghparast@jyu.fi; tommi.j.mikkonen@jyu.fi; juanmamu@unex.es; vlad.a.stirbu@jyu.fi
[*] *Corresponding Author*

Received 15 December 2025; Accepted 05 January 2026

Abstract

Quantum software engineering has gained a lot of attention recently. Multiple traditional software engineering events have introduced a quantum software track, or a co-located quantum related workshop or other side event, indicating that quantum software is becoming a popular research topic, with more and more software engineering researchers contributing to its evolution. In this paper, we address software engineering research that aims at solving problems that emerge when quantum programs are used on industry domains. The paper is based on the keynote at the IEEE Symposium on Quantum Software: Quantum Software Engineering 2025, which took place in Helsinki, Finland, Summer of 2025. In particular, we address the state of research in quantum software engineering, its novel aspects as well as its connections to other branches of software engineering. Furthermore, in the light of this research, we also assess the maturity of quantum software engineering in the light of industry expectations.

Journal of Web Engineering, Vol. 25_2, 283–298.
doi: 10.13052/jwe1540-9589.2526

Keywords: Quantum computing, quantum software, quantum software engineering, quantum programming.

1 Introduction

Quantum computing is a new way of computing that uses qubits, the quantum version of bits. Its power comes from quantum mechanics concepts like superposition and entanglement, which may allow it to solve certain problems faster than regular computers. For instance, weather forecasting involves complex calculations that are difficult for classical computers [7]. A fully developed quantum computer could handle some of these hard problems in far fewer steps than traditional ones. The first useful applications of quantum computing are likely to be in solving optimization problems, although there is no general consensus on the topic [12].

On the software side, there are major challenges to compose and run quantum programs [19, 23]. Although programming languages for quantum computing exist, writing actual programs still requires developers to think in terms of quantum circuits, similar to how classical circuits are designed, and the development in general takes place at a low abstraction level. Deploying a quantum program involves several steps: the circuit must first be transpiled into a form that a specific quantum computer can execute, then sent to the quantum processor; it is often controlled by mechanisms like laser pulses; and finally, the results must be read and analyzed for accuracy. Finally, classical software plays a key role in managing these processes, coordinating quantum executions, interpreting results, and handling the computer-specific interfaces needed for communication between classical and quantum systems.

In this paper, we address software engineering research that aims at solving problems that emerge when quantum programs are used on industry domains. The paper is based on the keynote [16] at the IEEE Symposium on Quantum Software: Quantum Software Engineering 2025, which took place in Helsinki, Finland, Summer of 2025.

The rest of this paper is structured as follows. In Sections 2–5, we address the many facets of quantum software engineering, consisting of continuity and connection to the past, something new that represents optimism for the future, something borrowed from other fields, and something blue standing for clarity and focus. In Section 6, we present a call to action to advance both the state of the art and the state of the practice in quantum software engineering.

2 Quantum Software Engineering: Something Old

The foundation of quantum software engineering (QSE) is deeply rooted in established mathematical and algorithmic concepts. Quantum computing, at its core, relies on the principles of linear algebra, utilizing concepts such as Dirac notation and matrix multiplication to describe and manipulate quantum states. This is a continuation of the long-standing practice in computer science where hardware operations are formally underpinned by mathematical structures.

Furthermore, a significant portion of early quantum algorithms are inspired by, or direct extensions of, classical computing techniques. Algorithms like Grover's search and Shor's algorithm are the most famous examples, showcasing how classical computational problems, such as searching and factoring, can be approached with a quantum speed-up. While the underlying physics is new, the problem domain and the goal, to devise efficient algorithms, are well-established challenges in computer science. The current catalogue of practically useful quantum algorithms is still limited, reinforcing the focus on established algorithmic targets. This old perspective emphasizes that QSE must not reinvent the wheel but rather adapt proven algorithmic methods to the quantum paradigm.

The current state of quantum programming today mirrors the early days of classical computing in the 1950s, which is another 'Something Old' challenge. Despite the existence of higher-level quantum programming languages like Qiskit, OpenQASM, Silq, and Q#, developers are still required to primarily think in terms of quantum circuits. This low-level, gate-centric approach to software development poses significant challenges. It needs deep quantum expertise, making the programming process difficult to follow for those without a background in quantum mechanics. For more detailed information regarding such programming languages and their characteristics, the reader is referred to Table 1.

This low-level development paradigm impacts the entire software lifecycle. Debugging quantum programs is inherently difficult, as stepwise execution often breaks fundamental quantum characteristics like superposition. The statistical nature of results from the noisy hardware (NISQ-era systems) [20] complicates iterative development and traditional verification. Moreover, the process of transpilation, converting a high-level circuit into an equivalent, hardware-specific circuit using only the gates supported by a particular quantum computer, introduces a classical-era problem: the need to adapt and optimize code for diverse and rapidly changing hardware

Table 1 Commonly used quantum programming languages and their characteristics.

Language	Intended use	Paradigm and characteristics
Qiskit	Learning quantum computing	Circuit-based, imperative quantum programming tightly integrated with Python
Cirq	Hardware-level research	Low-level, circuit-centric paradigm focused on explicit hardware operations
Q#	Enterprise/cloud integration	High-level, strongly typed quantum programming with clear classical-quantum separation
PennyLane	Quantum machine learning	Hybrid quantum-classical, differentiable programming paradigm for variational algorithms
Silq	Theory and language research	High-level declarative quantum programming with automatic uncomputation

configurations. This reminds the software engineering community that some of the most fundamental challenges of abstraction and tool development have not disappeared but have been reintroduced in a new, complex context.

The overall architecture of a quantum solution is not purely quantum, it is a complex, hybrid system. The success of QSE will rely on its ability to leverage well-established software engineering practices (e.g. [2, 22]) in a new context. The entire process, from user needs and system requirements to validation and operation, remains an essentially classical software engineering endeavor. Quantum components are ultimately just another specialized type of hardware, akin to graphics processing units (GPUs), that must be controlled and commanded by classical software.

Core software engineering concepts such as programming language research, tool chain development, and system architecture principles must be adapted, not abandoned. Even the application of research methods in QSE often borrows directly from the classical domain, including literature surveys and mining software repositories. QSE is, therefore, fundamentally rooted in using the known methods of classical software engineering to address the unique challenges of quantum systems, ensuring that problems like design, verification, and maintenance are addressed by drawing from decades of experience.

3 Quantum Software Engineering: Something New

Fundamental principles of quantum computing have been understood for decades. However, from a software engineering perspective, quantum computing introduces some genuinely new paradigms. Classical software runs on

the classical (digital) computers, using bits as the basic data unit. Of course it is possible to have multiple valued logic in classical software. Also classical software uses classical logic, Boolean operations, branches, loops, and more. Classical software are deterministic, i.e. given the same input array and environment, it will always produce exactly the same output. Classical software engineering is mature and has its ecosystem, tools, compilers, debuggers, operating systems, and optimized libraries. It scales well in several domains and is appropriate for most everyday tasks, e.g. for web and database. Error rates in classical software are low and well manageable via standard software engineering techniques and some hardware redundancies for reliability.

Usually the term quantum software refers to the software designed for exploiting the power of quantum hardware or to simulate it. Quantum software introduces new concepts that require its own attention. Here we explain new facets that are required to be considered in the emerging and promising field of QSE.

Quantum computing systems use qubits (instead of bits) that can be in superposition or become entangled, enabling interference effects. These concepts do not have a classical analog interpretation, as each operation is reversible, whereas in classical computations, each operation is irreversible. Hence, quantum computing can offer speedups for specific problems, such as factoring, unstructured search by exploiting superposition, interference, and entanglement that remain reversible until the result is read, at which point the computation becomes irreversible.

Quantum computing can be realized through different models, including the circuit-based model, as well as alternative paradigms such as quantum annealing and the measurement-based model. In practice, a vast array of contemporary approaches rely on hybrid quantum-classical computation, where the classical computer manages almost all the logical control, optimization loops, and data processing, while the quantum processor performs specific calculations and subroutines that benefit from quantum parallelism. Some of the well-known examples of such hybrid quantum-classical algorithms include the quantum approximate optimization algorithm (QAOA), and the variational quantum eigensolver (VQE). Of these, QAOA is a hybrid iterative method for solving combinatorial optimization problems, which has been a widely studied method in the context of noisy intermediate-scale quantum era (NISQ) devices in particular. In contrast, VQE is a quantum-hybrid algorithm for quantum chemistry, quantum simulations and optimization problems to find the ground state of a given physical system. Given an initial guess, the quantum processor calculates the expectation value of the system with

respect to an observable property, and a classical optimizer is used to improve the guess. The algorithm is based on the variational method of quantum mechanics.

In terms of programming and tooling, the quantum software ecosystem has grown rapidly, and a wide range of quantum languages and quantum software development environments are already available, while new instances are on the way. We have low-level languages such as OpenQASM and Qiskit for direct control over quantum circuits, while higher-level abstractions – including Silq, Q#, and frameworks like PennyLane – offer more expressive constructs for quantum algorithms and hybrid quantum-classical computations. Quantum software development relies on hybrid quantum-classical workflows. In this approach, classical computation orchestrates quantum operations.

Beyond quantum programming languages, an expanding set of toolchains supports advanced QSE practices: continuous integration and continuous delivery (CI/CD) pipelines can be applied to our quantum code, GitHub workflows enable modern, automated pipelines for QSE processes. New approaches to quantum versioning, quantum debugging, and quantum testing strategies are emerging. Quantum developers can leverage platforms and simulators such as IBM Quantum Experience, Amazon Braket, Microsoft Azure Quantum, and various local emulators, and quantum simulators to prototype, test, validate, and deploy their quantum quantum applications. Quantum circuit simulators enable developers to experiment with quantum circuits before sending them to real hardware [10]. Quantum circuit simulators are also indispensable for effective quantum education [11]. The word quantum in 'quantum computing' indicates the quantum mechanics that the system uses to compute the output. In physics, quantum is the smallest unit of any physical entity and generally it refers to atomic or subatomic particles, e.g. electrons, neutrons and photons. QC is a fast growing research field and is expected to bring revolution in various industrial areas (Zhao, 2020). QC is a fast growing research field and is expected to bring revolution in various industrial areas [25]. QC uses the concepts of quantum mechanics to process the information and complete specific tasks much faster than the classical computers. The quantum physics characteristics such as superposition, entanglement, and quantum interference are used and apply for computing purposes. Different studies are published that focus on developing the quantum algorithms [5, 9, 17]. Quantum computers outperform in two different ways as compared to the classical computers: 1) when the problem needs large amounts of parallel computing, e.g. big data analysis, [14, 25],

data security [25], encryption [18] and machine learning [1]. 2) when the problem requires precise and effective computation in natural science e.g. physics [3], material science [8] and chemistry [15].

The basic difference between the classical and quantum computer is the Qubit principle [25]. Different from classical computer bits (0,1), the quantum qubit can assign a state that is a superposition of 0 and 1. It means that a quantum system at a time could be at all possible states rather than one specific state. In quantum computers, the quantum register exists in all possible 0 and 1 superposition states, in contrast to classical, where the register has only one value at a time. Generalizing this, quantum computers offer a significant advantage over the classical where n bits have a fixed single state [25].

Looking ahead, several fundamental software engineering principles must be further developed to enable full support for quantum computation. Core concepts such as *abstraction* remain immature, as current quantum programming languages and frameworks expose an excessive amount of the underlying hardware complexity. *Modularity* is similarly challenging, since composing and reusing quantum components is not as straightforward as in classical software engineering. *Partitioning*, while easy in today's hybrid quantum classical architectures, becomes more complex when quantum algorithms scale. In addition, *testing practices* must be adapted to accommodate probabilistic outputs, noise, faults, errors and variation in hardware. Finally, *interpreting results*, especially in the presence of error, introduces a new layer of complexity that requires specialized tools and methodologies like quantum error correction tools and techniques. Together, these areas represent core requirements, and directions for future research in the field of QSE.

Several pressing challenges remain to be tackled as the field of QSE evolves. One major need is the development of a hardware abstraction layer (HAL) that functions as a true hardware abstraction, shielding developers from hardware constraints while still enabling high-performance execution. The emergence of next-generation tools for quantum algorithm design is equally important. The quantum software community also needs a comprehensive and easy-to-use algorithm library, enabling practitioners to build reliable, modular components rather than reinventing foundational routines. At the hardware level, limitations in coherence time, connectivity, and error rates still shape what is practically achievable. Furthermore, the no-cloning theorem imposes intrinsic constraints on debugging and testing – quantum states cannot be duplicated for inspection or replay – forcing the development of new testing methodologies that do not rely on classical paradigms.

Table 2 Key new concepts in quantum software engineering compared to classical software engineering

Concept	Classical software	Quantum software	Notes/examples
Information unit	Bit (0 or 1)	Qubit (superposition of 0 and 1)	Qubits enable parallelism
Key phenomena	None	Superposition, entanglement, interference	Core quantum advantages
Computational models	Classical logic circuits, Turing machine	Circuit-based, annealing, measurement-based	Hybrid combines classical and quantum parts
Task division	Classical CPU handles all	Classical CPU handles classical tasks; Quantum processor handles specific subroutines	None
Algorithm examples	Classical algorithms	Variational quantum eigensolver (VQE), quantum approximate optimization algorithm (QAOA)	Quantum-classical optimization

Addressing these challenges is essential for scaling QSE into a mature discipline.

Table 2 summarizes the key new items in quantum software compare to classical software.

Taken together, these discussions highlight two central new themes of the quantum-computing landscape. First, the quantum software ecosystem is expanding with extraordinary speed: new quantum programming languages, hybrid quantum classical frameworks, quantum software development toolchains, quantum simulators, and quantum cloud platforms are appearing at pace, providing quantum software researchers and developers with an abundance of emerging and evolving technologies. Second, this growth is accompanied by substantial limitations and a wide range of diverging research directions. Quantum hardware constraints, the lack of true abstractions, immature modularity and testing practices, the complexity of interpreting results under noise and error, and fundamental quantum principles such as the no-cloning theorem all create challenges that are required to be addressed. The need for next-generation quantum software tools, robust quantum algorithm libraries, and a true quantum hardware abstraction layer further illustrates how the field is simultaneously rich in innovation and constrained by unresolved practical and theoretical issues.

4 Quantum Software Engineering: Something Borrowed

Nowadays, there is no doubts about the interest that QSE is arousing. One of the main lessons learned during the evolution of classic software engineering is that software needs processes and processes need methodologies that help carry out each of the activities of the processes whether it is requirements specification, architectural design, detailed design, implementation or testing. Regarding the process for quantum software development, it could be assumed that the basic strategies for classical software development could be valid. They provide a general guide to approaching the problem with the aim of getting a piece of software that supports the problem. This leads us to conclude that designing the new techniques needed in the field of QSE requires taking into account all the new concepts along with all the lessons learned in classical software engineering development. Therefore, the commercialization and accessibility of quantum computing have largely adopted the "something borrowed" model of quantum-as-a-service (QaaS), directly adapted from the cloud computing paradigm. This approach is a strategic move to manage the immense infrastructure complexity of quantum systems. By abstracting the control layer, the cloud provides a managed infrastructure that handles crucial, non-trivial elements such as cryogenic environments, error correction protocols, and constant hardware and software version upgrades.

This cloud-native integration fits quantum jobs seamlessly into existing hybrid classical-quantum workflows. Users benefit from easy accessibility and collaboration, focusing solely on programming and experimentation rather than infrastructure management. This borrowing of the cloud service model is pivotal, as it lowers the barrier to entry, enables collaboration across disparate teams, and rapidly speeds up the adoption and experimentation necessary for the field to mature, bypassing the initial capital expenditure and maintenance hurdles associated with specialized quantum hardware.

Another major borrowing trend is the synergy with high-performance computing (HPC), forming an HPC-derivative track. This approach recognizes that quantum processors (QPUs) are best utilized as accelerators for sub-tasks that are computationally intractable for classical machines. To maximize this utility, tight coupling is necessary, where QPUs are integrated directly into HPC environments and connected over ultra-low-latency networks.

In this model, the classical HPC resources handle the large-scale classical computation: problem setup, extensive classical simulation, data preprocessing, and, critically, error mitigation techniques that are essential

for today's noisy quantum devices. The QPU then performs the specific, quantum-advantaged calculations. This setup requires the borrowing and specialization of existing HPC software stacks, including the development of new middleware, schedulers, and APIs. These tools must allow current HPC job managers (like Slurm or PBS) to seamlessly handle both classical and quantum computations, treating the QPU as another type of specialized computing resource in the HPC cluster.

The reality of modern quantum computation is that the quantum part is often 'only a drop in the ocean'. The vast majority of the system that constitutes a quantum application is classical, leading to a heavy reliance on borrowed tools and software artifacts. The entire tool chain, including the editor, classical compiler, transpiler, driver for quantum equipment, simulators, result interpretation, and post-processing tools, is a testament to this borrowing.

The need for a robust orchestration with classical software is paramount. Classical software must manage the entire execution lifecycle, coordinating the different steps and components of the hybrid computation. This structure allows QSE to leverage mature, sophisticated, and well-understood software engineering methodologies for the majority of the system, minimizing risk and complexity while focusing the limited resources on the truly novel quantum components.

Finally, QSE can borrow from the rich history of specialized software engineering (SE) sub-fields to define its own standards and body of knowledge. The existing literature is filled with successful roadmaps for SE in domains like security, safety, machine learning (SE4AI), automotive systems, and embedded systems. QSE does not need to start from scratch when defining its best practices.

By studying and adapting the established principles from these areas, such as techniques for managing high-assurance systems, the process for evolving tool chains, or methods for integrating specialized hardware, the QSE community can rapidly crystallize its own body of knowledge (Q-SWEBOK). Furthermore, the academic study of QSE often borrows research methodologies, such as porting cloud-related research problems to the quantum domain. This strategic borrowing prevents fragmentation and accelerates the maturation of QSE from an esoteric field to a mainstream, reliable discipline.

5 Quantum Software Engineering: Something Blue

The downside of the rapid advancement in some areas of QSE is the overplay of engineering and associated infrastructure. We are in the middle of a

chasm where infrastructure for QSE is ready for its prime time, but the applications are not [13]. To improve, we need more practical use cases and their implementations to shed light on the true challenges of QSE.

6 Call to Action

To advance the research and practice of software engineering in the field of quantum computing, there are some fundamental aspects to be addressed:

- Experiment: Today, much of the software engineering research takes place in isolation from actual quantum software development. Establishing an intimate connection between the two is an essential step forward to understand the true complexity of QSE.
- Focus: Today, we are missing a prime venue where QSE results can be published. Granted, there are numerous venues where papers can be sent, but a key venue is yet to be formed. Without one, QSE results will be scattered and fragmented, and not necessarily found by the community.
- Creating the QSE body of knowledge will be a step towards meeting the two above goals. This remains an important step forward in advent of true QSE research and practice.

Acknowledgments

This work has been supported by Business Finland projects EM4QS (155/31/2024) and SeQuSoS (112/31/2024), and the Research Council of Finland project DEQSE (349945). This work has also been partially funded by the Spanish Ministry of Science, Innovation and Universities and ERDF (PID2024-155693NB-C41 and PDC2025-165096-C31); by the Regional Ministry of Economy, Science and Digital Agenda of the Regional Government of Extremadura (GR24099) and by the Iberoamerican Science and Technology Program for Development(CYTED – 525RT0174).

References

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

[2] Barry Boehm. A view of 20th and 21st century software engineering. In *Proceedings of the 28th international conference on Software engineering*, pages 12–29, 2006.

[3] Andrew M Childs, Dmitri Maslov, Yunseong Nam, Neil J Ross, and Yuan Su. Toward the first quantum simulation with quantum speedup. *Proceedings of the National Academy of Sciences*, 115(38):9456–9461, 2018.

[4] Barry A Cipra. An introduction to the ising model. *The American Mathematical Monthly*, 94(10):937–959, 1987.

[5] David Deutsch. Quantum theory, the church–turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

[6] R Geoff Dromey. From requirements to design: Formalizing the key steps. In *First International Conference onSoftware Engineering and Formal Methods, 2003. Proceedings.*, pages 2–11. IEEE, 2003.

[7] Frank Gaitan. Finding flows of a navier–stokes fluid through quantum computing. *npj Quantum Information*, 6(1):61, 2020.

[8] Harper R Grimsley, Sophia E Economou, Edwin Barnes, and Nicholas J Mayhall. An adaptive variational algorithm for exact molecular simulations on a quantum computer. *Nature communications*, 10(1):3007, 2019.

[9] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[10] Majid Haghparast, Ronja Heikkinen, Samuel Ovaskainen, Julian Fuchs, Jussi PP Jokinen, and Tommi Mikkonen. Quirk-e quantum circuit simulator: Integrated tool for quantum algorithm development. *SoftwareX*, 32:102383, 2025.

[11] Majid Haghparast, Enrique Moguel, Jose Garcia-Alonso, Tommi Mikkonen, and Juan Manuel Murillo. Innovative approaches to teaching quantum computer programming and quantum software engineering. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 2, pages 251–255. IEEE, 2024.

[12] Nils Herrmann, Daanish Arya, Marcus W Doherty, Angus Mingare, Jason C Pillay, Florian Preis, and Stefan Prestel. Quantum utility–definition and assessment of a practical quantum advantage. In *2023 IEEE International Conference on Quantum Software (QSW)*, pages 162–174. IEEE, 2023.

[13] Petri Liimatta, Pauli Taipale, Kimmo Halunen, Teiko Heinosaari, Tommi Mikkonen, and Vlad Stirbu. Research versus practice in

quantum software engineering: Experiences from credit scoring use case. *IEEE Software*, 41(6):9–16, 2024.

[14] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature physics*, 10(9):631–633, 2014.

[15] Sam McArdle, Suguru Endo, Alán Aspuru-Guzik, Simon C Benjamin, and Xiao Yuan. Quantum computational chemistry. *Reviews of Modern Physics*, 92(1):015003, 2020.

[16] Tommi Mikkonen. Quantum software engineering – something old, something new; something borrowed, something blue. Keynote at IEEE Symposium on Quantum Software: Quantum Software Engineering, Helsinki, Finland, 2025.

[17] Ashley Montanaro and Sam Pallister. Quantum algorithms and the finite element method. *Physical Review A*, 93(3):032324, 2016.

[18] Michele Mosca. Cybersecurity in an era with quantum computers: Will we be ready? *IEEE Security & Privacy*, 16(5):38–41, 2018.

[19] Juan Manuel Murillo, Jose Garcia-Alonso, Enrique Moguel, Johanna Barzen, Frank Leymann, Shaukat Ali, Tao Yue, Paolo Arcaini, Ricardo Pérez-Castillo, Ignacio García-Rodríguez de Guzmán, et al. Quantum software engineering: Roadmap and challenges ahead. *ACM Transactions on Software Engineering and Methodology*, 34(5):1–48, 2025.

[20] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[21] Nauman A Qureshi, Norbert Seyff, and Anna Perini. Satisfying user needs at the right time and in the right place: a research preview. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 94–99. Springer, 2011.

[22] Vaclav Rajlich. *Software engineering: The current practice*. Crc Press, 2016.

[23] Javier Romero-Álvarez, Jaime Alvarado-Valiente, Enrique Moguel, José García-Alonso, and Juan M Murillo. Using open api for the development of hybrid classical-quantum services. In *International conference on service-oriented computing*, pages 364–368. Springer, 2022.

[24] Mahamadou Toure, Patricia Stolf, Daniel Hagimont, and Laurent Broto. Large scale deployment. In *2010 Sixth International Conference on Autonomic and Autonomous Systems*, pages 78–83. IEEE, 2010.

[25] Jianjun Zhao. Quantum software engineering: Landscapes and horizons. *arXiv preprint arXiv:2007.07047*, 2020.

Biographies

Jose Garcia-Alonso is an associate professor at the University of Extremadura in Spain, where he also earned his Ph.D. in software engineering. His research spans quantum software engineering, pervasive and mobile computing, e-Health, and service-oriented systems, yielding numerous publications and leading to significant funded research projects in these areas. He has also played an active role in technology transfer and entrepreneurship, contributing to the founding of technology-based companies, and serves as Director of Technology Transfer and Business Development at the university, fostering collaboration between academia and industry.

Majid Haghparast is an associate professor at the University of Jyväskylä in Finland, where he teaches and conducts research on quantum programming and quantum computing. He served as a research fellow during a sabbatical at Johannes Kepler University in Austria and is an IEEE Senior Member. Haghparast is an associate editor and board member for several international journals, including Cluster Computing and Journal of Computational Electronics. His research spans high-performance and distributed systems to quantum algorithms and software development.

Tommi Mikkonen works as a professor of software engineering at University of Jyväskylä, Finland. He has a strong interest in modern software development practices, software architectures, quantum software, and open source ecosystems. Alongside academic research and teaching, he collaborates closely with industry and open source communities, aiming to bridge theory and real-world software development. He received his doctorate in 1999 at Tampere University of Technology, Finland.

Juan Manuel Murillo Rodríguez is a Spanish computer scientist and full professor of software engineering at the University of Extremadura, where he has spent most of his academic career developing research and teaching excellence in software architecture, distributed systems, and service-oriented computing. His work bridges fundamental research and practical impact, active roles in professional societies, and leadership in technology transfer, including co-founding the tech startup GLOIN S.L. Beyond research, he has held institutional roles at the University of Extremadura and contributes to regional advanced computing initiatives through the COMPUTAEX Foundation.

Vlad Stirbu works as an associate professor of quantum software engineering at University of Jyväskylä, Finland. His current interest includes software development methodologies for software-intensive products and services, as well as software middleware for hybrid quantum-classical computing. He holds more than 15 US patents. He received his doctorate degree in 2012 at Tampere University of Technology, Finland.