Quantum Information and Computation, Vol. 24, No. 1&2 (2024) 0071–0088

GUIDELINES TO USE THE INCREMENTAL COMMITMENT SPIRAL MODEL FOR DEVELOPING QUANTUM-CLASSICAL SYSTEMS

RICARDO PÉREZ-CASTILLO[a], MANUEL A. SERRANO[b], JOSÉ A. CRUZ-LEMUS[b], MARIO PIATTINI[b]
University of Castilla-La Mancha (Spain)
{*ricardo.pdelcastillo, manuel.serrano, joseantonio.cruz, mario.piattini*}*@uclm.es*

Received October 3, 2023
Revised January 29, 2024

Quantum computing is the turning point that represents a revolution in software development that will make it possible to solve those problems unsolvable with classical computing. Just as in other milestones in the history of software development, such as the adoption of object-oriented systems, where new software development processes and new life cycles emerged, with the quantum computing revolution, a new life cycle for quantum and hybrid software systems is needed. Although there are some life cycle proposals for quantum software systems, most of them do not comprehensively address the specific needs of these systems. In this paper, a quantum life cycle proposal is presented adapted from the Incremental Commitment Spiral Model (ICSM) and an example of its use is presented.

Keywords: Life cycle model, hybrid systems, quantum software, quantum software processes, Incremental Commitment Spiral Model.

1 Introduction

Many problems that have until now been impossible to solve, in practical terms, might be able to be addressed using quantum computing [1]. "Quantum computing is at an inflection point with significant barriers to cross yet a world of opportunities ahead" [2]. These opportunities are (and will be) taking place in various fields such as cryptography, artificial intelligence, communications, optimization, pharmacology, medicine, chemistry, and materials development, among many other sectors [3, 4, 5].

Notwithstanding preliminary demonstrations of such advances and their potential, the advantages offered by quantum computing cannot be realized through the use of cutting-edge quantum computers in isolation, but rather quantum software is also required, and this will undoubtedly play an important role [6, 7]. Definitely, "software is the invisible writing that whispers the stories of possibility into our hardware" [8].

Quantum software technology has undergone a big-bang approach in the last few years. There is a wide variety of quantum programming languages [9], many quantum development environments [10, 11], and a wide variety of types of quantum simulators and hardware. Thereby, the quantum software programming techniques we have as of today have been experimentally proposed in an ad hoc manner. Consequently, there is still not a specific methodol-

[a]Faculty of Social Sciences & IT, Avda. Real Fábrica de la Seda s/n, 45600 Talavera de la Reina, Spain.
[b]Escuela Superior de Informática, Paseo de la Universidad, 4 13071 Ciudad Real, Spain.

ogy or process for developing high-quality quantum software. This gap is not easy to reliably fill even for those experienced in the design and development of classical software because the design, development, and analysis of quantum software are fundamentally different from classical software development practices prevalent today [12].

One of the most important problems in any information systems department is to define a common reference framework that can be used by all the people involved in the development of the systems, and in which the processes, activities, and tasks to be carried out are defined. Traditionally, the main professional organizations and international bodies have been dealing with the life cycle of systems and software. Thus, for example, both IEEE and ISO/IEC have published over time several standards entitled, respectively, "IEEE Standard for Developing Software Life Cycle Processes" and "Information Technology - Software life-cycle Processes". They currently propose a joint ISO/IEC/IEEE 12207 standard [13], which defines a life-cycle model as "a framework of processes and activities concerning the life cycle, which can be organized in stages, and which serves as a common reference for communication and understanding".

Most of these processes can be adapted for quantum software development. Some processes, for example, such as procurement, will become more important since it is very likely that to develop and offer quantum services, computing resources will have to be bought on various quantum platforms on-demand, since quantum computers will be hardly ever available on-premises. What will change in those processes will be the methods and tools when carrying out a specific process. For example, in the case of the system/software requirements definition, and analysis and design processes, extensions of the UML language for quantum software may have to be used [14, 15].

In the depicted landscape, what will probably change the most will be the quantum software lifecycle model. The lifecycle model adopted in a software development project influences the speed of development, improves quality assurance, control, and monitoring of the project, minimizes costs and risks, and improves customer relationships, among others. A wrong lifecycle selection can be a constant source of work slowdown, repetitive, unnecessary, and frustrating work.

It should also be borne in mind that, as the Talavera Manifesto [7] states: "QSE embraces the coexistence of classical and quantum computing", so the adequate lifecycle model should deal with both types of software. Hence, both classical and quantum software modules must be developed to be integrated and operated together within hybrid information systems [16], i.e., software systems combining quantum and classical software that works together. Hence, a lifecycle model for developing quantum software must also consider that feature. Akbar et al. [17] categorize various QSE challenges, from which this paper specifically deals with two of them, the 14th challenge: "Integration with classical computing", and the 22nd challenge: "Project management issues".

- Regarding classical-quantum software integration, quantum software cannot be developed and operated in isolation. "A key challenge is to, eventually, fully integrate these types of systems into a unified classical and quantum software development lifecycle" [18]. Thus, there are some issues, for example, the interpretation of results by the classical counterpart, since quantum computations are stochastic [19]. Also, new architectural paradigms and design patterns will be necessary to develop quantum software

effectively [20].

- Regarding project management, there are some issues like risk management specifically framed in the development of hybrid software systems. For example, a limited availability of real quantum hardware, which leads to test quantum software in simulators, sometimes leads to different results when the quantum software is executed in the production environment. Other risks with a high impact on the success of quantum software development projects are standardized tools and frameworks, or scalability [21].

The remainder of this paper is organized as follows: Section 2 presents an overview of lifecycle model evolution, while Section 3 discusses some related work about quantum software lifecycle models. Section 4 introduces the ICSM foundations and principles. Section 5 presents the hybrid quantum software lifecycle model proposed, and Section 6 provides the key decision input and some risk pattern examples to illustrate its application. Lastly, some conclusions and future research are detailed.

2 Evolution of software lifecycle models

In the history of software engineering, several breakthroughs have happened at different times [22]. The evolution of Software Engineering is "bottom-up" since it was developed after computer science foundations and computer manufacturing had already been laid. First, and most notably, the diffusion of third-generation languages such as COBOL in the 70s, resulted in the structured design techniques proposed by Myers, Yourdon, and Constantine; subsequently, the E/R model was defined by Chen, Gane and Sarson, DeMarco and Weinberg came up with structured analysis. In the 1980s comprehensive methodologies (Merise, SSADM, Information Engineering, etc.) were published. Grady Booch considers this period to be the first "golden age" of software engineering [8].

Regarding lifecycle models, the original version of the waterfall model of the life cycle was proposed by Royce [23] and since then, numerous refinements and variations of the model have appeared [24, 25]. This model received numerous criticisms [26] because software development is not linear and because it takes a long time to deliver something of value with a waterfall model. Because of this, other models appeared such as the Incremental [27] or the Spiral model [28].

In the 90s, the hot topic became object orientation, which presented object-oriented analysis, design, and development patterns. Frontiers between software lifecycle phases are blurred and, as a result, new software lifecycles emerged: Cluster model [29], Fountain model [30], "Pinball" model [31], etc., and a little later, the Unified Software Development Process [32] with a use-case driven, architecture centered, iterative and incremental software lifecycle model. In these years agile methodologies also began to disseminate, and therefore life cycle models such as the ones used in conjunction with Extreme Programming (XP) [33] and Scrum [34].

In turn, it was de adoption of Object-Oriented adoption that provided the seeds for the later model-driven architecture at the beginning of the 21st century. More agile and automatic development models were also used in the latter part of the first decade of this century, where DevOps or "continuous software engineering" [35] began to be widely adopted. A synthesis

of all these software lifecycle models can be found in the ICSM (Incremental Commitment Spiral Model) proposed by Boehm et al. [36, 37].

In the last 5 years, quantum computing has begun to make inroads into the industrial world [38] and we are convinced that it will be the main driver for a new software engineering golden age during this decade [39]. So, we need to adapt some of these well-known lifecycle models to this new scenario or specifically define a new one for quantum software production.

3 State of the art

There is very little research in the quantum field from a software engineering point of view, particularly concerning the languages and software development phases [40]. We have found only a few academic proposals for quantum lifecycle models.

Zhao [41] proposed a slight adaptation of the waterfall model, with five phases: quantum software requirements analysis, quantum software design, quantum software implementation, quantum software testing, and quantum software maintenance. He also introduced each phase from the perspective of quantum software development. The main problem with this model is that it inherits all the weaknesses of the classical waterfall life cycle that are aggravated (e.g., in the case of risks) with quantum software development.

In [42], a quantum development life cycle (QDLC) inspired by the waterfall model is proposed, with the following stages: feasibility study, system requirement specification, system design, software implementation and coding, system testing, and system maintenance and software quality management. In the quantum software implementation and coding phase the authors propose a first step (high-level programming abstractions) to separate classical and quantum functions. The second step is done by using logical-level schedulers and optimizers, and the third one, with physical-level schedulers and optimizers during the synthesis flow of quantum circuits. The last step of this stage is to get low-level constructs for device control firmware. In this model, in addition to the same previous weakness regarding the cascading life cycle, the quality assurance process is included within one stage of the lifecycle.

Weder et al. [43] proposed an iterative model based on "quantum data provenance", which serves in different phases of the life cycle such as error analysis or quantum hardware selection. This model consists of ten phases: 1) quantum-classical splitting, 2) hardware-independent implementation, 3) quantum circuit enrichment, 4) hardware-independent optimization, 5) quantum hardware selection, 6) readout-error mitigation preparation, 7) compilation & hardware dependent optimization, 8) integration, 9) execution, and 10) result analysis. The first phase decides which parts of the problem are solved on a quantum computer and which parts are solved on a classical computer based on the requirements of the problem description. This is perhaps the life cycle model best suited to the specific characteristics of quantum software, but this model does not manage risks and value-based decisions throughout the lifecycle very well, losing some of the advantages of classical models, especially when developing hybrid information systems.

As far as the main manufacturers of quantum software are concerned, they do not make concrete proposals, limiting them to a few steps, for example, in the case of IBM Qiskit saying that: "A basic workflow using Qiskit consists of two stages: Build and Execute. Build allows you to make different quantum circuits that represent the problem you are solving and execute that allows you to run them on different backends" [44]. The lifecycle presented by

Amazon Braket is somewhat more elaborate distinguishing four phases: Build, Test, Run, and Analyze [45]. QPath provides a toolchain supporting the development of hybrid software systems in a technology-agnostic way [46], which considers four phases: design, construction, testing, and execution of quantum software assets.

4 ICSM foundations and principles

The Incremental Commitment Spiral Model (ICSM) was proposed by [36]. ICSM defines "a process model that extends the scope of the original spiral model for software development to cover the definition, development, and evolution of cyber-physical-human systems" [37]. ICSM proposes a risk-based framework that is based on some principles and is aimed at "defining and evolving project and corporate process assets, avoiding pitfalls and disruption, and leveraging opportunities to increase value" [36].

ICSM uses four essential principles to determine whether, where, and when to use candidate common-case process elements (reuse-based, prototype-based, agile, architected agile, plan-driven, product-line, systems of systems, legacy-based, etc.). The four essential principles are [37]:

- Stakeholder value-based system evolution: ICSM focuses on the value of the system to stakeholders and the evolution of the system to meet their needs.
- Incremental commitment and accountability: ICSM emphasizes incremental commitment and accountability to ensure that the system evolves in a controlled and manageable way.
- Concurrent multi-discipline engineering: ICSM promotes concurrent multi-discipline engineering to ensure that all aspects of the system are considered and integrated.
- Evidence and risk-based decisions: ICSM uses evidence and risk-based decisions to ensure that the system evolves in a way that is consistent with the needs of stakeholders and the goals of the organization 1.

ICSM is not defined to be applied massively to any software development process or company. Instead of this, ICSM is designed to be tailored to the specific needs of an organization.

The overall lifecycle proposed in ICSM, as shown in Figure 1, comprises two major stages, which in turn consider various activities [36]:

- State 1. Incremental Definition. It covers the up-front growth in system understanding, definition, feasibility assurance, and stakeholder commitment. This stage considers three main activities:
  - Exploration, which defines the initial scope of the project.
  - Valuation, devoted to specifying the initial concept of the software systems as well as a viability analysis.
  - Foundations, aimed at defining the architecture baseline and the main system operations. This activity also focuses on specifying increment plans.

- Stage 2. Incremental Development Operations and Production. This stage is aimed at implementing a feasible set of specifications and plans for incremental development and operations of the desired system. This stage iteratively considers increments with the following activities:
  - Development, that provides the software increment previously defined (i.e., exploration, valuation, and foundations).
    Production and Operation, aimed at making the developed increment accessible in the production environment.

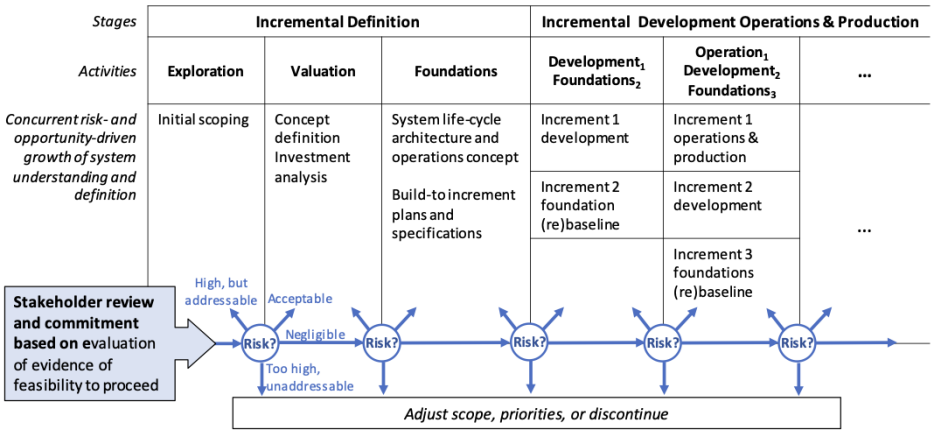

Fig. 1. Stages and activities defined in the incremental commitment spiral model. Adapted from [36].

5 Quantum ICSM Guidelines Proposal

We propose a set of guidelines to adapt the usage of ICSM (Incremental Commitment Spiral Model), proposed by Boehm et al. [36], to hybrid quantum software development. ICSM is based on four principles: stakeholder value-based guidance, incremental commitment and accountability, concurrent multidiscipline engineering, and evidence- and risk-based decisions.

The rationale for considering ICSM as the basis for the proposed lifecycle model lies in the fact that challenges that attempt to address ICSM [36] are almost the same that hybrid quantum software development projects face:

Multi-owner, multi-mission systems of systems (SoS). The nature of hybrid software systems can be considered in that way since new quantum software algorithms have to be integrated with numerous independently evolving legacy or external systems. On the one hand, quantum algorithms could be dynamically built regarding data produced by classical software systems. On the other hand, outputs of quantum software components are used by other existing software systems that probably need to accomplish some adaptations.

- Emergence and human intensiveness with many requirements not pre-specifiable, budgets and schedules not pre-specifiable which lead to a need to manage uncertainty and

risk. Quantum software development project has specific success factors in comparison with classical software. Thus, [21] points out that resources and specialized skills are some of the most critical success factors. Hybrid software development has, generally, genuine risks that classical software does not have. For example, lack of tools for covering specific software development tasks like debugging and testing, the need for involving multidisciplinary teams, and well-founded architectural software patterns, among others.

- Rapid pace of change that claims for incremental developments and adaptability to avoid technology obsolescence and changes in mission priorities. Although classical software development experienced those challenges, hybrid software development must deal with a higher pace of changes in quantum software technology [18]. On the one hand, there is a race with the major quantum hardware vendors developing scalable machines with more and better qubits [47], which in some cases can constrain and impact how quantum software is designed and developed. On the other hand, quantum software technology is in a big-bang approach with plenty of programming languages, and tools that change every day [11].

Figure 2 shows the overview of the ICSM model. It proposes working in iterations that incrementally add activities of the next phases. At the end of each iteration, a new increment is provided, and an evidence- and risk-based commitment review is done. The result of the assessment can determine a risk to be: i) negligible, then the process continues with a new increment and new phases of the process; ii) acceptable, then the process stays working in the same increment, same phases; iii) high, but addressable, then the process backtracks and works in an increment involving past phases; or iv) too high or unaddressable, then the decision could be to discontinue the project.

ICSM differentiates between two stages: i) incremental definition (cf. section 5.1), and ii) incremental development operations & production (cf. section 5.2). Each stage, in turn, defines various phases.

5.1 Incremental Definition Stage

The first stage consists of three phases: exploration, valuation, and foundations. [36] advise that these three phases may be combined in simple cases and depending on the risk assessment. However, we think that in hybrid/quantum information systems is important to keep these three phases separated and get a management and technical foundation to develop a robust and flexible architecture stable and predictable.

5.1.1 Exploration Phase

This phase groups 5 activities: clarify and assess need/potential benefits, conduct a gap analysis against existing capabilities, develop initial concept description, identify potentially feasible alternatives for further analysis, and capture risks and develop mitigation plans. This phase starts with a proposal explaining the need and context for the need and produces several outputs: initial concept description, business case for need, list of feasible alternatives for further analysis, key risks and mitigation plans, and guidelines/need budget for further analysis.

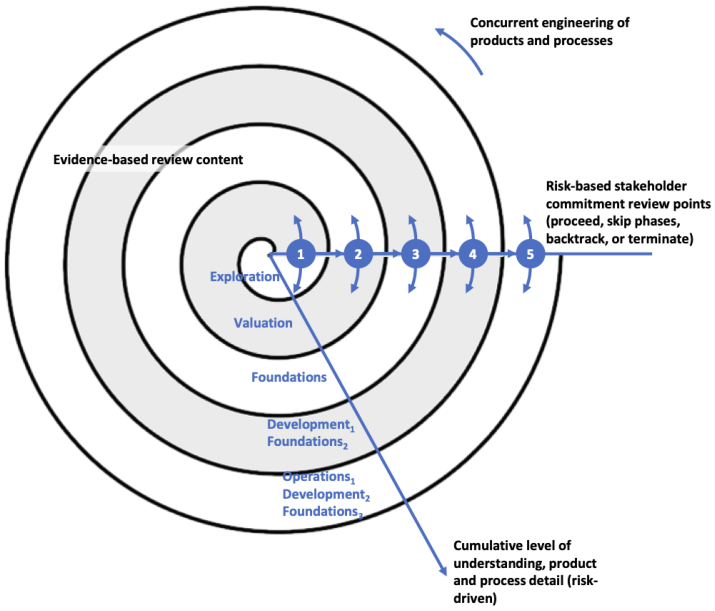

Fig. 2. The incremental commitment spiral model. Adapted from [36].

In our case, it will be necessary to explore and determine the different alternatives. ICSM suggests using decision tables or trees. In this case, these decisions could be:

- Solve the problem with classical software.
- Solve the problem with hybrid systems, i.e., classical and some quantum routines. Then, it must be explored both quantum computing paradigms: (i) gate-based quantum computing; and (ii) annealing quantum computing.

There are some proposals [48] that attempt to determine what functionalities make sense to be developed as quantum software. Similarly, [49] provides a decision-making framework for use cases of quantum and quantum-inspired algorithms. This work focuses on the methodological part to decide if a use case should, or should not, be developed as quantum software.

Apart from considering the different technological alternatives, it is necessary –in the case of using quantum computing– to explore if there is already a quantum algorithm that could be reused to solve the problem, or if it is necessary (and feasible) to create a new one adapted to the problem.

Given that quantum computing is a rather new technology, more detailed feasibility evidence must be required to support the commitment review to proceed further.

5.1.2 Valuation Phase

This phase groups 6 activities: refine and implement the valuation plan developed in the Exploration Phase, monitor changes in needs/opportunities/risks, adapt plans to address

identified changes, evaluate results of valuation activities, develop foundations strategy and plans, and update risks and mitigation risks plans. This phase takes as inputs the outputs of the previous phase and produces the following outputs: analysis of any prototypes, updated risks and mitigation plans, approved foundations strategy plan, and key stakeholder commitments/MOAs (memorandums of agreement).

One of the big issues for quantum software is that it must operate in cooperation with classical software. So, in this phase it is very important to analyze if there exists:

- Any existing information system which could be "reengineered" (totally or partially) to the quantum world. If any, a modernization software approach could be followed in the next stage [16].
- Any existing quantum library that could be used, usually in a specific domain such as Chemistry, Physics, etc.
- Any specific need for the quantum system to interoperate with the classical software counterpart. If any, there are proposals for modeling the integration of quantum software [50], oriented to the orchestration of classical and quantum software computations [43], oriented to the architectural design [14, 15], or graph-based modeling [51]. Such models could help in this phase to comprehend and evaluate the target system.

Regarding the valuation, it should be noted that quantum systems are costly, so perhaps it should be considered a first prototyping and testing phase using a simulator and later the final implementation in a real quantum computer. Analyzing potential quantum simulators is one activity that might be accomplished within the third point discussed.

5.1.3 Foundation Phase

This phase groups 7 activities: ensure technology readiness for needed capabilities, monitor changes in needs/opportunities/risks, prototype and evaluate various alternatives, select acquisition development strategies, prioritize features requirements for development, develop a plan for the development based upon prioritization, and update risks and mitigation risk plans. As outputs of this phase: list of approved features/requirements allocated to components of configuration items, approved development plan, feature allocation to increments, updated risks, and mitigation plans, updated key stakeholder commitments/MOAs, requests for proposals for outsourced development.

Perhaps the most difficult task will be the definition of a hybrid fully developed system architecture, with the planning of all the needed connections between classical and quantum systems and the tailoring needed in quantum algorithms and software libraries. According to the current service-orientation trend, quantum software can be also delivered and consumed as a service. There are preliminary approaches to developing quantum software as a service [52]. Service orientation provides many advantages, but also comes up with additional risks/challenges to be considered, e.g., service composition, configuration management, monitoring, and security, among others.

As current quantum computers are NISQ (Noisy Intermediate-Scale Quantum), i.e., limited with quantum computations disturbed by errors [53], the "quality" tolerance levels needed by the problem must be considered. Another issue is the stability of the quantum vendors,

which (even in the case of the main one) is not guaranteed due to the lack of maturity of the technology.

Another important risk in this stage is the outsourcing contracts with specialized quantum developers and researchers. Due to the lack of workforce in this field [54, 55, 56], this challenge should be managed as well.

5.2 Incremental Development Operations & Production Stage

The second stage consists of two phases, the development and production stages.

5.2.1 Development Phase

This phase takes as inputs: a list of approved features/requirements, problems reported against the currently deployed system, approved changes, approved development plan, feature allocation to increments, updated risks and mitigation plans, updated stakeholder commitments/MOAs, and proposals for outsourced development. As the output of this phase, the model proposed the "system ready for production/deployment". This phase is divided into two types of increments: initial with 4 activities, and follow-up increments with 7 activities.

The initial increment, with 4 activities, sets up development environments; develops detailed design; selects hardware, COTS (Commercial Off-the-Shelf) products, and outsources vendors; and updates foundations as needed based on selections.

Regarding quantum software, the previously explored and evaluated quantum platforms and other needs are eventually selected, and an initial architectural design is provided. Also, a draft of the communications schema between classical and quantum software is defined.

All the subsequent increments have the following 7 activities:

- **Update detailed design**. In this activity, the design of classical and quantum modules is completed. At this point, "hardware-software co-design approaches offer the potential to efficiently and effectively achieve the best mappings of challenging applications onto constrained hardware" [57]. Co-design is important because of the current NISQ devices, and changes in quantum hardware may require revisiting previous phases of ICSM life cycle. Co-design and abstractions between layers of the technological stack (hardware-software) must be used in combination to deliver a detailed design.

- **Procure/develop/ integrate hardware components**. In this activity is especially important to ensure availability and compatibility of all the quantum resources involved in the target system.

- **Develop/procure/integrate software features/requirements according to the development plan**. This activity includes coding tasks to get executable software artifacts, both classical and quantum. At this point, the integration between classical and quantum software is detailed. There exist some design techniques to manage this, for example, the density matrix for classical and quantum software proposed by [58].

- **Monitor changes in needs/opportunities/risks**. Before completing every increment, quantum functionality should be evaluated. At some point, the development of some functionalities, that were decided to be supported by quantum computing, can experience some troubles, or get unexpected results. Or the other way around, new

opportunities can suggest shifting some classical functionalities to quantum. Potential risks should be also analyzed, most of them are related to the current volatility of today's quantum software development technology.

- **Update foundations as needed**. Similar to the previous activity, quantum software development is a boiling field of research.

- **Conduct continuous verification and validation (V&V)**. This is an important activity. The ordinary testing and verification efforts must be done individually for both, classical and quantum software. Of course, quantum software testing is a field to be explored yet [59]. Testing quantum software (gate-based or quantum annealing) is not trivial due to the non-deterministic and probabilistic nature of quantum software. Even so, there are some preliminary testing approaches [60, 61]. Together with unitary testing, integrating and system testing deserve a lot of attention. Here, how quantum software computes results and how these are returned and interpreted by classical software is critical. Another important challenge within this activity is the optimization of quantum software. For example, there are various works for optimizing quantum circuits in gate-based platforms (i.e., reducing the number of operations or qubits), while in quantum annealers the weights and factors of the underlying Hamiltonians can be also optimized. Finally, it is important to conduct acceptance tests that cover whole workflows, i.e., those that include classical and quantum software.

- **Update approved features, risks, and mitigations at the end of each increment.**

5.2.2 Production and Operations Phase

This phase is divided into two types of increments:

- **Production**. It takes as inputs the approved production plans and consists of 5 activities: acquire/establish production or manufacturing facilities and resources, produce systems units under approved production plans/orders, develop maintenance strategies and plans for the system, produce user information/training materials, and produce help desk support materials. We can find outputs like system units ready for operations, user information/training materials, help desk materials and training, and maintenance and logistics plans.
  When quantum software is released and deployed to production, specific concerns about the quantum computers that will run routines of quantum software must be taken into account. As we introduced, most of today's quantum computers are operated in the cloud. Thus, various alternatives for services and billing should be considered. Since, quantum software may have been developed with quantum simulators or emulators, compatibility of the developed software and the target quantum computers should be managed.

- **Operations**. It takes as inputs; approved operations and support plans, system updates, and help desk updates. It consists of 4 activities: distribute and support system units and components under approved plans, distribute and support approved system changes under approved plans, operate system help desk and provide user assistance as

requested, and triage user problem reports and change requests and forward to Development as needed. We have two kinds of outputs: support (problem reports to be resolved and change requests for implementation consideration); and end-of-life (authorization to replace, retire, or dispose of system).

For quantum software development, the operation increment should be aware of quantum computing noise, regarding the current NISQ devices. Every time quantum software is performed, results could slightly vary. During the operation phase, these limitations should be considered. Any change in quantum hardware or software platforms should be considered as well.

6 Key Decision Input and Risk Patterns

As it was introduced, the usage of ICSM implies (re)evaluating risks at the end of each iteration and making four possible decisions according to the risk evaluation: (i) negligible and continue with a new increment and new phases of ICSM; (ii) acceptable, then it stays working in the same increment, same phases; (iii) high, but addressable, then the process backtracks and works in an increment involving past phases; (iv) too high or unaddressable, then the decision could be to discontinue the project. In ICSM, the common key decision inputs are aspects like (i) product and project size and complexity; (ii) requirements volatility; (iii) mission criticality; (iv) nature of commercial off-the-shelf (COTS), services support, etc.; (v) commercial, open source, reused components; (vi) organizational and personnel capability. When ICSM is applied for developing quantum-classical software systems, there are specific, additional decision points to be considered during risk assessments. Table 1 summarizes common decision inputs that can be used as an initial checklist to evaluate risks in each ICSM phase for managing the development life cycle of hybrid software systems.

Although the available stages and phases are the same for all the software development processes that follow ICSM, there could be various risk patterns that can be distinguished. As a result, different risk patterns yield different software development processes. The actual lifecycle can therefore be slightly different in each case. Figure 3 illustrates such different processes in four different scenarios (examples A to D) throughout the first 5 increments. The processes presented for each scenario should be understood as examples that could vary depending on the different risk assessments in each increment.

Example A (see Figure 3) represents the common software development project for a classic information system without any quantum software. Except for the exploration phase, the next phases (valuation and foundations) have a negligible technological risk. Then during the subsequent development and operation phases, the possible risk is acceptable. This is due to the technology stack for building classical software is certainly mature with a low level of uncertainty.

Example B (see Figure 3) provides an exemplar process for the integration of new quantum software (gate-based) into a hybrid information system (probably by following a modernization/reengineering approach). In this example, the exploration phase could even be discarded to develop this project, and the valuation phase could detect a high risk that could lead to exploring other alternative solutions. The same happens in the foundations phase. As we explained before, different vendor solutions suffer from certain limitations (number of qubits, error tolerance, kind of quantum algorithm, etc.) that could lead to a rethink of the

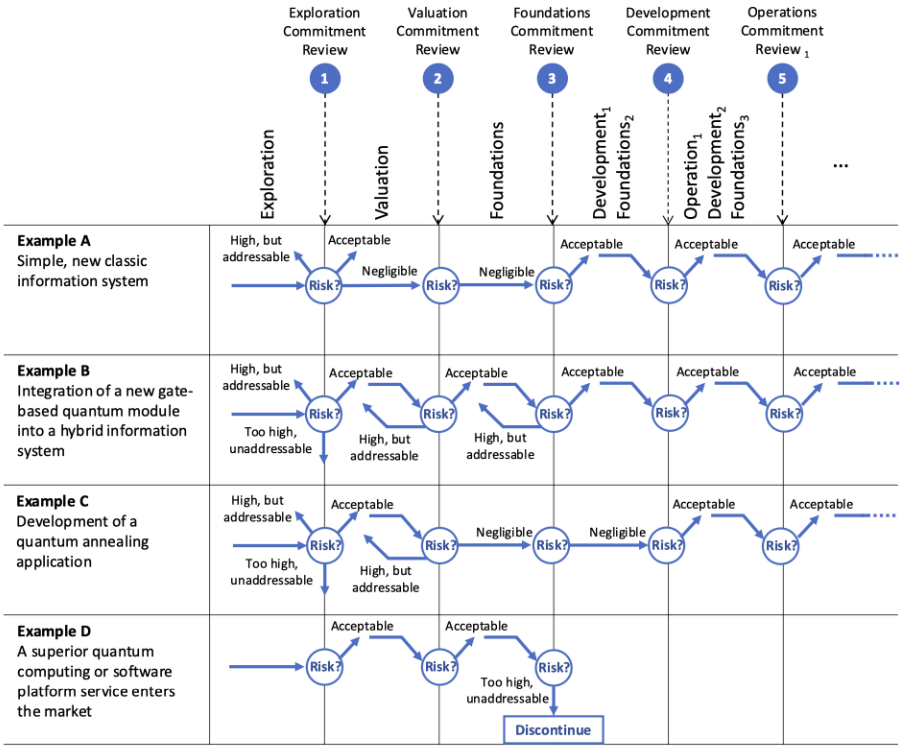

Fig. 3. Examples of different software processes framed in ICSM for different risk patterns.

Table 1. Common decision inputs during risk assessment in ICSM phases for hybrid software.

Stage	**Phase**	**Key Decision Inputs & Risks Assessment**
Definition	Exploration	Solve the problem with classical software
		Solve the problem with some quantum software components
		Existing algorithms to used/adapted
		New algorithms to be developed
	Valuation	Existing information system to be re-engineered (totally or partially) to quantum paradigm
		Specific quantum toolkits for specific domains/sectors
		Needs for workflow operations between classical and quantum
Development Operations & Production	Foundation	Service-Orientation challenges: service composition, configuration management, monitoring, security
		Exploration using simulators instead of actual quantum computers (which are more expensive)
		Error tolerance and power (e.g., number of qubits) of quantum hardware
		Quantum technology stability
		Available quantum software workforce
	Development	Co-design quantum software-hardware
		Unitary testing for quantum software
		Integration testing for quantum algorithms and classical software drivers
		New opportunities can suggest shifting some classical functionalities to quantum
		Volatility of the today's quantum software development technology
	Operation	Error tolerance and power (e.g., number of qubits) of quantum hardware
		Scalability (requests, users, etc.)

exploration and/or valuation phase. After the incremental definition has been passed, the incremental development and operations could be conducted with a controllable risk as the development of classical software found.

Example C (see Figure 3) alternatively shows the development of a simple optimization application based on quantum annealing. In a similar way to example B, exploration and valuation phases could deal with uncertain risk levels. However, the risk found in the foun-

dations and development phase should be negligible since the adiabatic quantum computing technology seems to be stable and helps to solve a narrow set of problems with well-known methods and tools. Then, the risk in the following phases could be higher but, due to the operation phase's challenges, but acceptable in any way.

Finally, Example D (see Figure 3) illustrates a possible issue that could happen during the life cycle of a hybrid information system. Let us imagine that a new, superior (fault tolerance) quantum computing or software platform service is available in the market with clear advantages. This fact could lead to high risk because of a specific commitment review at any phase, due to the technology obsolescence. In that case, if the risk is too high and it is not affordable (from a point of view of the cost, time-to-market, or any other) the project could be discontinued.

Although these risk pattern examples are suitable for many of the common quantum software development scenarios, our proposal is not limited to these examples. Further risk patterns could be considered according to the ICSM.

7 Conclusions

Quantum computing has progressed in the last years beyond academia or research labs and has been embraced by large organizations looking for financial returns and a competitive edge [2]. But to boost large-scale production of high-quality quantum software we need a novel "Quantum Software Engineering" [41, 62, 63, 64], including critical software engineering techniques, such as lifecycle models.

As the evolution of software engineering teaches us, each paradigm shift (structured programming, object-oriented programming, DevOps, etc.) has made new lifecycle models and software development techniques necessary. When defining models for quantum reality, we must be aware that the new software world will be composed of both classical and quantum software interacting with each other.

Software engineers must be aware that classical lifecycle models may not be as effective with the new quantum technology, so they must create models that are best suited to their environment: hybrid information systems, gate-based systems, annealing systems, or a mix of both.

Given this new revolution in software and the need for new quantum/hybrid software development life cycles, our main lines of future research will focus on completing those phases of the life cycle that vary substantially concerning classical computing, such as the development of new algorithms, reuse of quantum algorithms, testing of quantum code or reengineering of classical systems.

Acknowledgements

We would like to thank all the aQuantum members, especially Guido Peterssen and Pepe Hevia, for their invaluable help and support.

This work was partially funded by the 'QHealth: Quantum Pharmacogenomics Applied to Aging' project, the 2020 CDTI Missions Program (Center for the Development of Industrial Technology of the Ministry of Science and Innovation of Spain), as well as by grants PID2022-137944NB-I00 (SMOOTH Project) and PDC2022-133051-I00 (QU-ASAP Project) funded by MCIN/AEI/ 10.13039/501100011033 and by the "European Union NextGenera-

tionEU/PRTR".

References

1. S. Aaronson (2008), *The limits of quantum computers*, Scientific American, Vol.298, no.3, pp. 62-69.
2. B. Lenahan (2021), *Quantum boost: using quantum computing to supercharge your business*, Library and Archives Canada.
3. Interamerican Development Bank (2019), *Quantum technologies. Digital transformation, social impact, and cross-sector disruption.*
4. International Business Machines Co. (2022), *The quantum decade. A playbook for achieving awareness, readiness, and advantage (3rd ed.)*, IBM Institute for Business Value
5. D. Ukpabi, H. Karjaluoto, A. Bötticher, A. Nikiforova, D. Petrescu, P. Schindler, V. Valtenbergs, L. Lennard, and A. Yakaryilmaz (2023), *Framework for understanding quantum computing use cases from a multidisciplinary perspective and future research directions*, arXiv, Vol.2212.13909v2
6. L. Mueck (2017), *Quantum software*, Nature, Vol.549, no. 7671, pp. 171.
7. M. Piattini, G. Peterssen, R. Pérez-Castillo, J.L. Hevia, M. Serrano, G. Hernández, I. García-Rodríguez, C. Andrés, M. Polo, E. Murina, L. Jiménez, J.C. Marqueño, R. Gallego, J. Tura, F. Phillipson, J.M. Murillo, A. Niño, and M. Rodríguez (2020), *The Talavera Manifesto for Quantum Software Engineering and Programming* , Proc. 1st International Workshop on QuANtum SoftWare Engineering & pRogramming (QANSWER 2020), Talavera de la Reina, Spain.
8. G. Booch (2018), *The History of Software Engineering*, IEEE Softw, Vol.35, no.5, pp. 108-114.
9. M. Piattini (2021), *Requirements for a Robust Quantum Software Development Environment*, Cutter Business Tech J, Vol.34, no.4, pp. 12-17.
10. J.L. Hevia, G. Peterssen, C. Ebert, and M. Piattini (2021), *Quantum computing*, IEEE Softw, Vol.38, no.5, pp. 7-15.
11. M. Serrano, J.A. Cruz-Lemus, R. Pérez-Castillo, and M. Piattini (2023), *Quantum software components and platforms: overview and quality assessment*, ACM Comp Surveys, Vol.55, no.8, pp. 1-31.
12. M. Martonosi and M. Roetteler (2019), *Next Steps in Quantum Computing: Computer Science's Role*, arXiv, Vol.1903.10541v1
13. International Organization for Standardization (2017), *ISO/IEC/IEEE 12207:2017 Systems and software engineering – Software life cycle processes.*
14. R. Pérez-Castillo and M. Piattini (2022), *Design of classical-quantum systems with UML*, Computing, Vol.104, no.11, pp. 2375-2403.
15. C.A. Pérez-Delgado and H.G. Pérez-González (2020), *Towards a quantum software modeling language*, Proc. 42nd International Conference on Software Engineering Workshops, pp. 442-444.
16. R. Pérez-Castillo, M. Serrano, and M. Piattini (2021), *Software modernization to embrace quantum technology*, Advances in Eng Softw, Vol.151, 102933
17. M.A. Akbar, A.A. Khan, and S. Rafi (2023), *A systematic decision-making framework for tackling quantum software engineering challenges*, Autom Softw Eng Vol.30, 22.
18. A.D. Carleton, M.H. Klein, J.E. Robert, E. Harper, R.K. Cunningham, D. De Niz, J.T. Foreman, J.B. Goodenough, J.D. Herbsleb, I. Ozkaya, D. Schmidt, and F. Shull (2021), *Architecting the Future of Software Engineering: A National Agenda for Software Engineering Research & Development*, Software Engineering Institute, Carnegie Mellon University.
19. M. Haghparast, T. Mikkonen, J.K. Nurminen, and V. Stirbu (2023), *Quantum Software Engineering Challenges from Developers' Perspective: Mapping Research Challenges to the Proposed Workflow Model*, arXiv Vol.2308.01141v1
20. A.A. Khan, A. Ahmad, M. Waseem, P. Liang, M. Fahmideh, T. Mikkonen, and P. Abrahamsson (2023), *Software architecture for quantum computing systems — A systematic review* J Syst Softw, Vol.201, 111682.
21. M.A. Akbar, A.A. Khan, M. Hameem, and M. Nadeem (2024), *Genetic model-based success*

probability prediction of quantum software development projects, Inform & Softw Tech, Vol.165, 107352.
22. B. Boehm (2006), *A view of 20th and 21st century software engineering*, Proc. 28th International Conference on Software Engineering, Shanghai, China, pp. 12–29.
23. W.W. Royce (1970), *Managing the Development of Large Software Systems: Concepts and Techniques*, Proc. IEEE WESCON, Vol.26, pp. 328-388.
24. B. Boehm (1981), *Software Engineering Economics*, Prentice Hall (Engelwood Cliffs, New Jersey).
25. Y. Sommerville (1985), *Software Engineering*, Addison-Wesley Publishing Company (Wokingham, United Kingdom).
26. D. McCracken and M. Jackson (1982), *Life Cycle Concept Considered Harmful*, ACM SIGSOFT Softw Eng Notes, Vol.7, no.2, pp. 29-32.
27. M.M. Lehman (1984), *A Further Model of Coherent Programming Processes*, Proc. IEEE Software Process Workshop, Eghan, United Kingdom.
28. B. Boehm (1988), *A Spiral Model of Software Development and Enhancement*, Computer, pp. 61-72.
29. B. Meyer (1990), *The new culture of software development*, J Object-oriented Programming, Vol.3, pp. 76-81.
30. B. Henderson-Sellers and J.M. Edwards (1990), *The object-oriented systems life cycle*, Comm ACM, Vol.33, no.9, pp. 142-159.
31. S.W. Ambler (1998), *Building Object Applications That Work – Your Step-by-Step Handbook for Developing Robust Systems with Object Technology*, Cambridge University Press (New York).
32. I. Jacobson, G. Booch, and J. Rumbaugh (2000), *Unified Software Development Process*, Addison-Wesley.
33. K. Beck (1999), *Extreme Programming Explained: Embrace Change*, Addison Wesley Longman, Inc.
34. K. Schwaber and M. Beedle (2001), *Agile Software Development with Scrum*, Prentice Hall.
35. B. Fitzgerald and K.J. Stol (2017), *Continuous software engineering: A roadmap and agenda*, J Syst Softw, Vol.123, pp. 176-189.
36. B. Boehm, J.A. Lane, S. Koolmanajwong, and R. Turner (2014), *The Incremental Commitment Spiral Model: Principles and Practices for Successful Systems and Software*, Addison Wesley.
37. B. Boehm and L. Huang (2015), *The incremental commitment spiral model (ICSM): principles and practices for successful systems and software*, Proc. International Conference on Software and System Process (ICSSP 2015)
38. S. Ali, T. Yue, and R. Abreu (2022), *When software engineering meets quantum computing*, Comm ACM, Vol.65, no.4, pp. 84-88.
39. M. Piattini, G. Peterssen, and R. Pérez-Castillo (2020), *Quantum Computing: A New Software Engineering Golden Age*, ACM SIGSOFT Software Engineering Notes, Vol.45, no.3.
40. P.E. Zanni and W. Vieira de Camargo (2021), *A systematic mapping on quantum software development in the context of software engineering*, arXiv, Vol.2106.00926v1.
41. J. Zhao (2020), *Quantum Software Engineering: Landscapes and Horizons*, arXiv, Vol.2007.07047v2.
42. N. Dey, M. Ghosh, S. Samir, and A. Chakrabarti (2020), *QDLC - The Quantum Development Life Cycle*, arXiv, Vol.2010.08053v1.
43. B. Weder, U. Breitenbücher, F. Leymann, and K. Wild (2020), *Integrating Quantum Computing into Workflow Modeling and Execution*, Proc. IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC), pp. 279-291.
44. International Business Machines Co. (2023), *Get started with Qiskit*, Online available: https://docs.quantum.ibm.com/start/hello-world, accessed 29 January 2024.
45. Amazon Web Services, Inc. (2023), *Amazon Bracket* Online available: https://aws.amazon.com/es/braket/, accessed 29 January 2024.
46. J.L. Hevia, G. Peterssen, and M. Piattini (2022), *QuantumPath: A quantum software development platform*, Softw: Pract Exper, Vol.52, no.6, pp. 1517-1530.

47. S. Witt (2022), *The world-changing race to develop the quantum computer*, The New Yorker.
48. J. Misra, V. Kaulgud, R. Kaslav, and S. Podder (2021), *When to Build Quantum Software?*, arXiv, Vol.2104.09117v1.
49. N. Chancellor, R. Cumming, and T. Thomas (2020), *Toward a standardized methodology for constructing quantum computing use cases*, arXiv, Vol.2006.05846v1.
50. S. Ali and T. Yue (2020), *Modeling quantum programs: Challenges, initial results, and research directions*, Proc. 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software, pp. 14-21.
51. D. Alonso, P. Sánchez, and B. Álvarez (2023), *A Graph-Based Approach for Modelling Quantum Circuits*, Applied Sciences, Vol.13, no.21, 11794.
52. J. García-Alonso, J. Rojo, D. Valencia, E. Moguel, J. Berrocal, and J.M. Murillo (2022), *Quantum Software as a Service Through a Quantum API Gateway*, IEEE Internet Computing, Vol.26, no.1, pp. 34-41.
53. J. Preskill (2018), *Quantum Computing in the NISQ era and beyond*, Quantum, Vol.2, p. 79.
54. J.D. Biamonte, P. Dorozhkin, and I. Zacharov (2019), *Keep quantum computing global and open*, Nature, Vol.573, no.191.
55. J. Hilton (2019), *Building the Quantum Workforce of the Future*, Forbes Technology Council.
56. G. Peterssen (2020), *Quantum Technology Impact: The Necessary Workforce for Developing Quantum Software*, Proc. 1st International Workshop on the QuANtum SoftWare Engineering & pRogramming (QANSWER 2020), Talavera de la Reina, Spain.
57. T. Tomesh and M. Martonosi (2021), *Quantum codesign*, IEEE Micro, Vol.41, no.5, pp. 33-40.
58. I. Exman and A.T. Shmilovich (2022), *Quantum Software Models: Density Matrix for Universal Software Design*, In M. Serrano, R. Pérez-Castillo, and M. Piattini (eds.) Quantum Software Engineering, Springer.
59. A. García de la Barrera, I. García-Rodríguez de Guzmán, M. Polo, and M. Piattini (2023), *Quantum software testing: State of the art*, J Softw Evol Proc, Vol.35, no.4, e2419.
60. M. Polo (2020), *Quantum software testing*, Proc. 1st International Workshop on the QuANtum SoftWare Engineering & pRogramming (QANSWER 2020), Talavera de la Reina, Spain.
61. E. Mendiluze, S. Ali, P. Arcaini, and T. Yue (2021) *Muskit: A Mutation Analysis Tool for Quantum Software Testing*, Proc. 36th IEEE/ACM International Conference on Automated Software Engineering (ASE2021).
62. L.S. Barbosa (2020), *Software engineering for 'quantum advantage'*, Proc. IEEE/ACM 42nd International Conference on Software Engineering Workshops, pp. 427-429.
63. E. Moguel, J. Berrocal, J. García-Alonso, and J.M. Murillo (2020), *A Roadmap for Quantum Software Engineering: Applying the Lessons Learned from the Classics*, Q-SET @QCE, pp. 5-13.
64. M. Piattini, M. Serrano, R. Pérez-Castillo, G. Peterssen, and J.L. Hevia (2021), *Towards a Quantum Software Engineering*, IT Professional, Vol.23, no.1, pp. 62-66.