Advances in Engineering Software 151 (2021) 102933

Contents lists available at ScienceDirect

Advances in Engineering Software

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

Discussion

Software modernization to embrace quantum technology

Ricardo Pérez-Castillo [a,b,*], Manuel A. Serrano [b], Mario Piattini [b]

[a] Faculty of IT & Social Sciences, University of Castilla-La Mancha, Av. Real Fábrica de Sedas, s/n, 45600, Talavera de la Reina, Spain
[b] Information Technology & Systems Institute, University of Castilla-La Mancha, Paseo de la Universidad, 4, 13071, Ciudad Real, Spain, University of Castilla-La Mancha

ARTICLE INFO

Keywords
Quantum
Software Modernization
Reengineering
KDM
UML

ABSTRACT

Quantum Computing is becoming an increasingly mature area, with a simultaneous escalation of investment in many sectors. Quantum technology will revolutionize all the engineering fields. For example, companies will need to add quantum computing progressively to some or all of their daily operations. It is clear that all existing classical information systems cannot be done away with. Rather than that occurring, it is expected that some quantum algorithms will be added, so that they can work alongside classical information systems. There has been no systematic solution offered to deal with this challenge so far. This research proposes a software modernization approach (model-driven reengineering) designed to restructure classical systems to work in conjunction with quantum systems, thereby providing target environments that combine both of these computational paradigms. The approach proposed is systematic, and based on existing software engineering standards like the Knowledge Discovery Metamodel and the Unified Modelling Language. It could therefore be applied in industry in a way that complies with the existing software evolution processes. The independence of this proposal with respect to quantum programming environments is also guaranteed, making its application feasible in the changing environment in today's quantum industry. The main implication of this approach is technical, but also economic, since it enables the reuse of the knowledge embedded in legacy systems, while at the same time the new quantum-based projects are speeded up.

1. Introduction

At the beginning of the last century, the basis of a new physics theory, "quantum mechanics" was established, its purpose being to describe the behaviour of nature at subatomic levels (photons, electrons, etc.). This led to the first quantum revolution, which made it possible to develop revolutionary technologies. In the eighties, physicists realized that although classical physics governs classical, binary computers, it also restricts the representation and processing of information in computers. Quantum physics thus started to be applied in computer science, and the second quantum revolution began. Quantum physics has been exploited from the nineties onwards in the quest to increase computational power exponentially [1].

The quantum momentum is well-established today. Quantum Computing (QC) is becoming an increasingly mature area, with a simultaneous escalation in investment in both public and private sectors [2, 3]. QC is actually a crosscutting and interdisciplinary opportunity for digital transformation and social impact through the revolution of many engineering areas. These areas include, amongst many others, chemical engineering, biomedical simulations and disease diagnosis, machine learning, optimization problems such as logistics, financial modelling and risk management, cybersecurity and cryptography engineering.

Despite the fact that QC is becoming more and more mature and mainstream, software engineering for quantum software has not been considered in the same depth as software engineering for classical software over the last decades [4-6]. This present work is therefore framed within the new Quantum Software Engineering (QSE) paradigm. In particular in this work, we highlight the problem presented by software modernization (model-driven reengineering) [7]. We believe that most organizations will demand migration of their first quantum algorithms or those they are going to use in the future; meanwhile these functionalities will be integrated into the enterprise's existing classical information systems. In the short term, quantum computers will obviously not be used for everything. Apart from the initial prohibitive cost, there are not many well-known algorithms; some specific tasks might be still carried out by classical computers, and with better performance. It will be more common to use quantum computers to solve certain difficult problems by means of specific calls from classical computers to

* Corresponding author.
E-mail addresses: ricardo.pdelcastillo@uclm.es (R. Pérez-Castillo), manuel.serrano@uclm.es (M.A. Serrano), mario.piattini@uclm.es (M. Piattini).

https://doi.org/10.1016/j.advengsoft.2020.102933
Received 25 December 2019; Received in revised form 14 July 2020; Accepted 27 October 2020

remote quantum ones. In this scenario, software modernization processes have proven to be an effective mechanism in the migration and evolution of software, while still preserving business knowledge [7, 8].

The proposed software modernization approach allows classical systems to be restructured alongside existing or new quantum algorithms to provide target systems that combine both classical and quantum information systems. The solution proposed is systematic, and is based on existing, well-known standards like the Unified Modelling Language (UML) [9] and the Knowledge Discovery Metamodel (KDM) [7]. This paper provides the overall process with the expected requirements, and attempts to explain what role existing standards (KDM and UML) will play in the software modernization of classical-quantum information systems.

The main implication of this contribution is the technical and economic impact derived from the possibility of reusing the knowledge embedded in legacy information systems, while also reducing the effort of developing new quantum information systems. In addition, since this proposal is based on international standards, so as to represent knowledge in a technologically-agnostic way, independence with respect to quantum programming languages and environments is achieved. As a result, the application of this proposal is feasible in the volatile environment that is expected to exist during the first stages of the QC industry.

The remainder of this paper is organized as follows: Section 2 presents the state of the art on quantum technology. Section 3 explains the proposal of this research, a software modernization process for classical-quantum information systems. Section 4 provides some implications for researchers and practitioners. Finally, Section 5 draws conclusions from this work.

2. State-of-the Art

This section attempts to explain the state of the art regarding quantum computing (see section 2.1), as well as the status of specific technology that supports quantum computing (see section 2.2).

2.1. Quantum Computing

Classical (digital) computers are almost everywhere today [10]. This ubiquitous presence ranges from simple controllers in modern appliances and smartphones in our pockets which provide a wide range of everyday services, to powerful supercomputers and large data centres that carry out the most computationally-intensive tasks [1]. All these computers have one thing in common; all of them store and handle information in bits (i.e., 0 or 1).

Classical computation works well for numerous different scenarios, although it presents some limitations in many other contexts. Some important computational problems exist (e.g., NP-complete problems) and are believed to be very difficult to solve, even using the most powerful computers. The resource constraints in these problems (i.e., the size of the machine or the time it takes to finish the task) increase exponentially as a function of the size of the problem [1]. This is the situation, one of limitations and constraints, as it stands at the present time. This is despite the fact that Moore's law [11] has been fulfilled over the last fifty years, thanks to dramatic improvements in manufacturing technology and hardware efficiency.

Quantum computing applying quantum mechanics to computing science. It changes the traditional bits, the smaller unit of information, with qubit. It can be said that a qubit is the quantum version of a bit and it is encoded by the two basis states of the quantum system represented as $|0\rangle$ and $|1\rangle$, using Dirac's bracket notation. A qubit is a multiple status quantum system. In contrast to a classical bit, which may be in a state of 0 or 1, a qubit may exist in a superposition of the two states $|\psi\rangle = x_0 |0\rangle + x_1 |1\rangle$, where x_0 and x_1 are the probability of being 0 or 1. This means that a qubit state might be 0 and 1 at the same time. Those coefficients satisfy the normalization condition $|x_0|^2 + |x_1|^2 = 1$. Qubits can be internally handled by different physics systems behaving as a waves at subatomic level. For example, the spin of the electron, in which the two states can be taken as spin up and spin down; or the polarization of a single photon, in which the two states can be taken to be the vertical polarization and the horizontal polarization [12]. This phenomenon is known as superposition and is the key to the exponential computational power since n qubits can be represented by a superposition state vector in 2^n.

Another counterintuitive principle that quantum computers use is and entanglement, i.e. objects can be deeply linked without any direct physical interaction. An important characteristics derived from superposition and entanglement have to do with how the result is measured for a particular qubit. When something is measured at the quantum level, the quantum object that has been measured is no longer in a superposition of states; it rather collapses to a single classical state bit [13]. This means that once the spin of the electron or photon is measured (i.e. its value is observed), the possible values for the qubit are 0 or 1, and this value does not change afterwards. QC thus initializes and uses qubits that work in a probability space until these values are measured, becoming actual binary values.

In a similar way to the classical logic gate operations, Quantum logic gates compute the output states from the input states, allowing the input states to be an arbitrary Quantum state (such as a superposition state). Quantum logic gates are implemented by designing the time evolution of the input Quantum states to achieve the desired output states, often by manipulating the external control signals. Mathematically, Quantum gates are described by unitary matrices, and their application is accomplished through multiplication of the respective matrix by the state vector. Pauli matrices, X, Y and Z are examples of basic, useful single-qubit quantum state transformations. X, Y, and Z represent values that depend on what state our qubit is on the Bloch sphere. As an example of quantum gates, the controlled-NOT gate (C_{not}) operates on two qubits by changing the second bit if the first one is 1, otherwise leaving this bit unchanged.

Quantum circuits represent a computation model based on the combination of quantum gates. Gates can be applied sequentially and in parallel for various qubits, forming an acyclic-directed graph. It is useful to have graphical representations of quantum state transformations, especially when several transformations are combined.

This quantum computing paradigm has a great deal of potential applications [14], such as privacy and cryptography, optimization in supply chain and logistics, chemistry and material simulation, economics, and financial services, sustainable energy and farming, health care and drug development, etc. The impact of research in industry clear in a field that is expected to expose a significative grow in the next years [15, 16]. Actually, quantum computing is a transversal and interdisciplinary opportunity for digital transformation and social impact.

2.2. Quantum Computing Technology

Some small-scale but already fully programmable and universal quantum computers are available to a broader public via cloud access today, and larger systems with better performance are expected to become available in the coming years. Some examples of those services are IBM Q Experience [17], which makes it possible to compose some Quantum circuits and execute them in some remote IBM Q computers. Similarly, Rigetti provides Quantum Cloud Services. Other huge companies have been announcing quantum computer projects over the last few years. For example, Google presented a 72-qubit device. Intel is advertising the third-generation Tangle Lake quantum processors, which contain 49 superconducting qubits. We can also highlight the appearance of D-Wave, which offers what has been called *quantum annealing* to search for solutions to certain optimization problems. D-Wave technology can combine digital and quantum processing units. Finally, similarly to other governments, China is carrying out a strongly-funded program to develop a quantum computer with more

than 100 qubits.

Although these companies and organizations have anticipated 50 to 100 qubit prototypes in the near future, the QC revolution still depends heavily on the software tools needed to bridge the gap between algorithms and physical machines [4] (e.g., editors, compilers, parsers, etc). These tools will accelerate the success of QC by allowing QC algorithms to be optimized into smarter programs that require fewer qubits and operations.

There is a vast amount of programming languages for classical machines in existence at the present time. Most programmers write their source code in one or more of the high-level programming languages, such as C++, Python, or Java. Developers usually ignore the architecture of the machines they are working with and can thus concentrate on the task in hand; they simply let the interpreter/compiler take care of what goes on "under the hood" [13]. There are some quantum programming languages that can be considered as raw quantum machine language, where the frontier between quantum hardware and quantum software is thin and blurred [13]. Probably the earliest proposal for a formalized quantum programming language, as opposed to a low-level definition, is QRAM (Quantum Random Access Machine) [18]. Although the QRAM model is not defined formally, it consists of a register machine with the ability to perform quantum operations, including state preparation, unitary transformation and measurement, on quantum registers. Other high-level languages based on the existing classical ones have been developed in the last decade. QCL [19] has a C-like syntax, augmented by a new *QReg* type, which lets the programmer access quantum registers (qubits). Q takes the approach of promoting operators to fully-fledged objects, as in C++. While QCL is in a sense self-contained, Q looks like an extension of C++, enriched by *QRegisters* and *QOperators*. There are other high-level quantum programming languages that have appeared as functional languages, such as ML, Miranda, or Haskell. Although these are very powerful and flexible, they are not very popular in the industrial world [13]. One example of this is QPL and its extension cQPL [20]. Another language is QFC [21], which combines quantum data and classical control. In contraposition to QFC, QML [22] claims that both data and control are quantum.

Q#, which is developed by Microsoft, is, similarly to QFC, a .NET compliant language that can be integrated with other .NET languages to manage the control flow of a program. As with the IBM Q Experience, Microsoft Quantum [23] is a platform that offers the possibility of integrating quantum programs into a .NET platform and executing it in a QC simulator or a real Quantum Computer. Although there are many quantum programming languages [20], the use of these languages depends on the available tooling and integration with quantum computers. While some of them are executed in simulators like Q#, others like QisKit [24] (provided by IBM) or Forest (provided by Rigetti) are full-stack libraries to hide hardware details. At the present time, these libraries are usually released as open source code, as are many other available tools. Open source software is in fact becoming crucial in the design and testing of quantum algorithms [25]. Many of the tools are backed by major commercial vendors, aiming to make it easier to develop quantum software (see Figure 1).

These libraries are in turn based on assembly languages as an intermediary point with the quantum hardware. For example, QisKit is based on OpenQASM [27], which makes it possible for quantum circuits to be composed in IBM Q Experience or in external environment/ programming languages like Python. The advantage of the QisKit framework is that quantum programs can be executed in real quantum computers in the cloud (see Figure 1). Another example of a circuit composer is Quirk [28], an open source emulator for quantum gate composition that is able to execute them and to visualize the behaviour of qubits.

3. Software modernization towards quantum systems

Apart from differences between quantum and classical software, new software systems will probably integrate classic and quantum computation, since not all kinds of problems are suitable for addressing from a purely quantum point of view. That being the case, future software will include some pieces of code in classical programming languages which perform calls to quantum algorithms that are executed in quantum computers (see Figure 2).

The new architecture presented in Figure 2 must be taken into account for all the upcoming QSE areas. One of the most critical areas in QSE is undoubtedly reengineering and software modernization. Once QC becomes more mainstream, companies whose goal is to achieve quantum supremacy [29] will probably migrate their existing systems to QC, or will do so for at least part of those classical systems with difficult or demanding algorithms.

3.1. Software modernization of classical systems

Despite the fact that legacy systems may be obsolete, this kind of system usually has a critical mission within the company, and represents a valuable asset for them, since legacy systems embed a large amount of business logic and business rules that are not present elsewhere [30]. As a result, companies cannot discard their legacy systems, in spite of the upcoming quantum revolution.

Reengineering has been a successful practice in the software industry. It consists of three phases: reverse engineering, restructuring and forward engineering. When dealing with specific challenges, more than half of the traditional reengineering projects fail because of a lack of

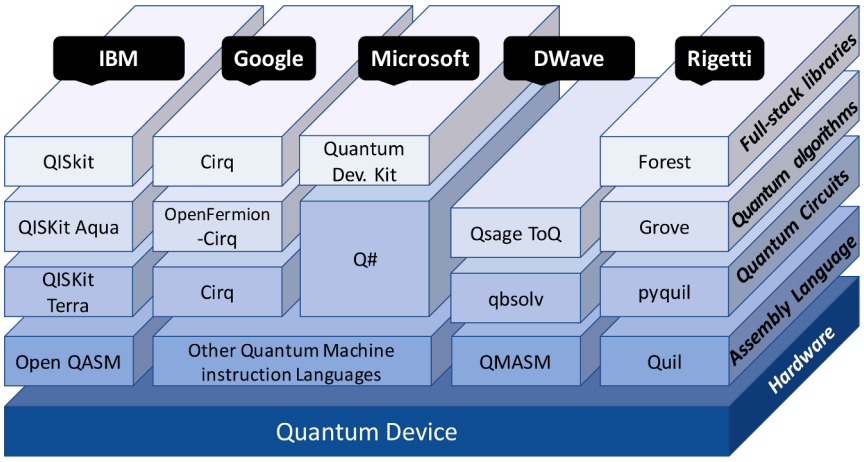

Figure 1. Some of the existing quantum computing languages according to the level of programming (adapted from [26]).

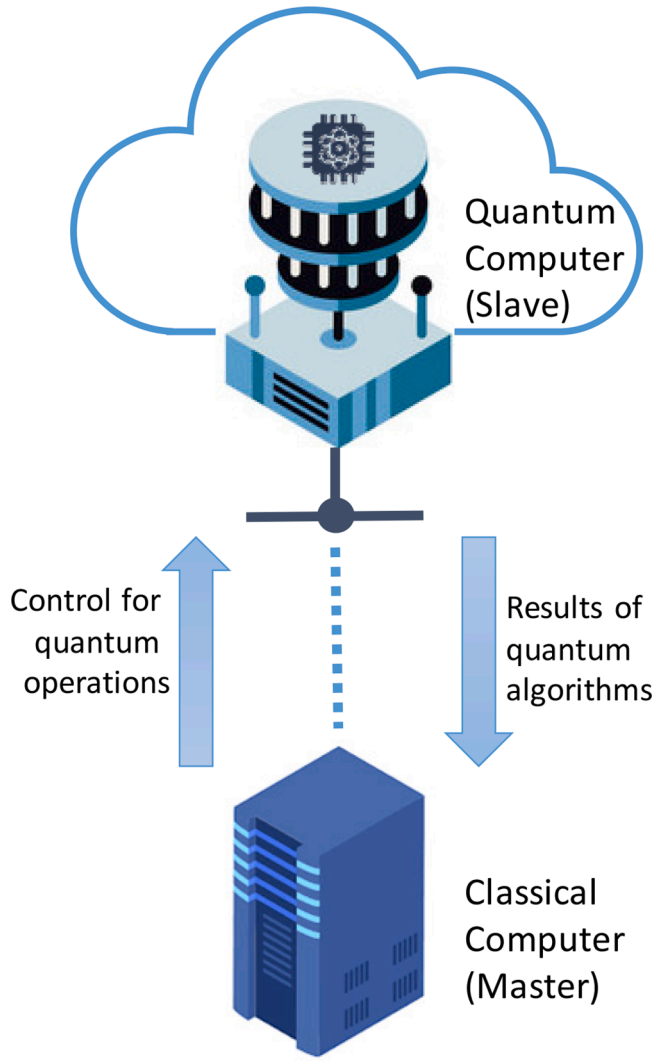

Figure 2. Composition of software systems with quantum computing.

standardized and automated processes [31]. In the first place, standardization is a problem because the reengineering process has typically been carried out in an *ad hoc* manner [8]. This means that reengineering projects must focus their efforts on providing a better definition of the process. Furthermore, the code cannot be the only software asset that the standardization covers, since *"the code does not contain all the information that is needed"* [32]. The reengineering process must be formalized to ensure integrated management of all the knowledge involved in the process, such as source code, data, business rules, and so on. Secondly, automation is also a very important issue. To prevent failure in large complex legacy systems, the reengineering process must be more mature and repeatable [33]. In addition, the reengineering process needs to be aided by automated tools so that companies can handle the maintenance cost [31]. Automation can, moreover, be considered as a problem that derives from the standardization problem, since standardization and the formalization of the process are requirements needed in the endeavour to provide tools that automate the process and which can be reused in several reengineering projects.

The software modernization paradigm, and particularly ADM as defined by the OMG, can be considered as a mechanism for software evolution, i.e., it makes it possible to modernize the legacy information systems and eradicates, or at least minimizes, the software erosion problem in legacy systems. [34]. This approach is aligned with the low-code paradigm [35], the latest trend in enterprises in which sophisticated platforms are employed for generating new code for their applications. There are therefore progressively fewer use cases in which organizations must hand-code anything. This means that only a Platform-Independent Model is necessary, with the details that specify how that system uses a particular type of platform or technology.

ADM facilitates the reverse engineering stage by means of the Knowledge Discovery Metamodel (KDM) [7], since this standard makes it possible to represent all software artefacts involved in a certain legacy system in an integrated and standardized way. The KDM standard is used to represent all the software artefacts involved (i.e., source code, databases, user interactions, etc.), and it achieves this in an integrated and technological-independent manner. This means that it is possible to have a common KDM repository that is gradually completed with knowledge discovered through the analysis of different artefacts in the legacy systems. KDM can be compared with the UML standard (ISO/IEC 19505) [9]: While UML is used to generate new code in a top-down manner, a process involving KDM starts from the existing code and builds a higher level model in a bottom-up way [36].

Eventually, KDM plays the role of an abstraction mechanism to represent legacy information systems as a whole, and not just represent their source code [7, 37]. The KDM metamodel is divided into four layers, which represent both physical and logical software artefacts of information systems at several abstraction levels [7]. Each layer is organized into packages that define a set of metamodel elements; the purpose of these is to represent a specific independent facet of knowledge related to LIS as architectural viewpoints.

3.2. Software modernization for quantum systems

Software modernization and reengineering practices must be brought into the domain of quantum computing. Reengineering and software modernization has therefore to be revisited, if we are to deal with the problems associated with the expected quantum computing migrations and the upcoming coexistence of classical and quantum software (see Figure 2).

We propose a software modernization based on existing standards such as UML and KDM (ISO/IEC 19505 and 19506 respectively). Figure 3 shows the complete view of the software modernization approach proposed.

As regards KDM, we believe that if reverse engineering of classical systems (plus quantum programs, if any) is carried out and the extracted knowledge is represented holistically in a KDM repository, then reengineering and migration towards quantum environments is improved; see (see left path in Figure 3). This means that the previous knowledge and business rules are preserved, and the impact of the integration of quantum programs is limited. As far as UML is concerned, the standard must be extended (through the standard mechanisms) to represent and integrate quantum programs. KDM models can thus be transformed automatically into UML representations, and/or engineers can model quantum aspects manually for new target systems (see right path in Figure 3).

3.2.1. Expected requirements

The software modernization process we propose for hybrid information systems (quantum/classical systems) must meet the following list of 6 requirements. The proposed software modernization process summarized in Figure 3 also presents a mapping with these requirements.

- **Req. 1.**The process considers efficient techniques that should be designed and developed for extracting and populating knowledge from classical information systems into an integrated KDM repository.
- **Req. 2.**A parser is employed for the analysis of quantum code programs and to integrate the analysed information into the KDM repository.

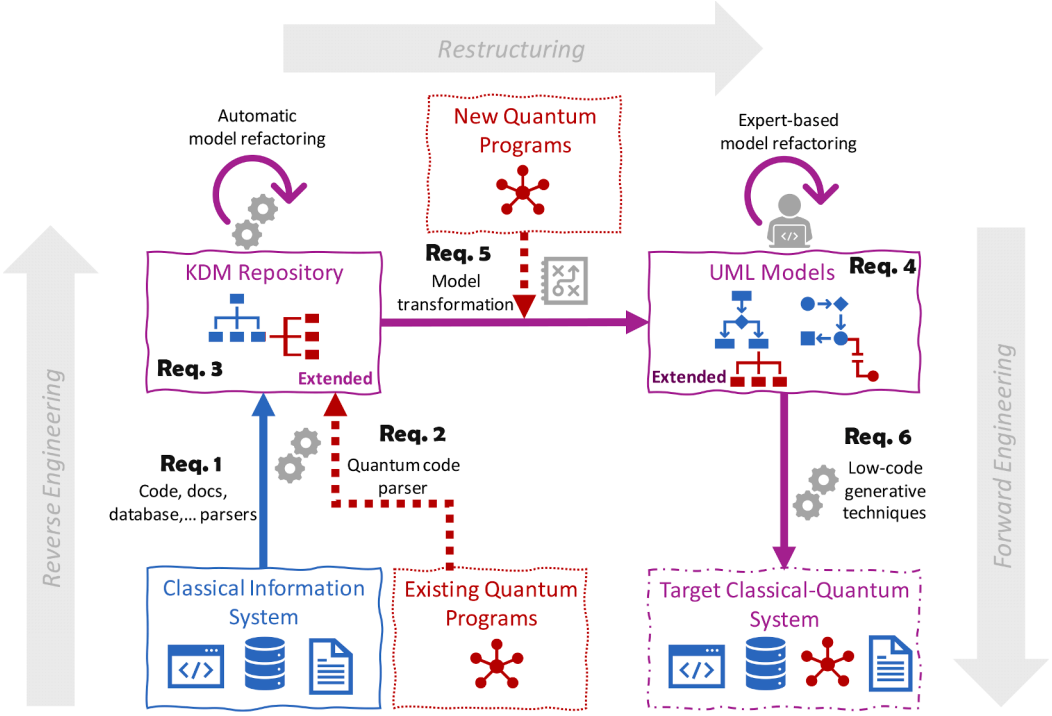

Figure 3. Complete view of the process for software modernization towards quantum computing.

- **Req. 3.**The modernization process takes into account a KDM extension defined through its standard extensibility mechanisms (i.e., extension families) so that the quantum knowledge can be integrated in the KDM.
- **Req. 4.**A UML extension is considered for the representation of high-level models for quantum programs and how those programs are integrated with classical information systems artefacts.
- **Req. 5.**The process employs automatic model transformations to transform KDM models into UML models. These transformations consider both quantum extensions (see previous requirements).
- **Req. 6.**Low-code/generative techniques are implemented to facilitate the link from a high-level description to low-level implementation with quantum gates and the integration of these in the target hybrid system.

3.2.2. *Software modernization phases*

In this section, the three phases of the quantum modernization model in Figure 3 are explained in detail.

3.2.2.1. Reverse engineering from classical/quantum programs to KDM. Several parsers are used to extract and represent knowledge from different classical information systems artefacts such as source code, database schemas, service descriptions, user interfaces information, among other. Such analysed information is then integrated into a common KDM repository according to the different layers and views of KDM [7] (i.e., code, data, events, etc.). Specific parsers must also be designed and developed to analyse quantum code (e.g., QisKit, QASM or Q#) so that KDM elements and relationships are generated in the mentioned KDM repository. The development of these specific parsers can be aided by some tools that generate code from language grammar definitions.

The KDM standard changes the way in which reverse engineering tools are built and used. Traditional reverse engineering tools have been built as silos from which each tool recovers and analyses different proprietary content in a single silo. However, the KDM standard makes it possible to build reverse engineering tools in the KDM ecosystem. Here, the reverse engineering tools recover a range of knowledge related to different artefacts (classical source code or quantum code), but this knowledge is represented and managed in an integrated and standardized manner through KDM. Consequently, other software analysis tools can analyse the KDM repository and generate new knowledge, like high-level architecture or design for classical-quantum systems.

KDM has to be extended to represent quantum programs and circuits in a way that is similar to classical IS artefacts. Two cases are considered: existing quantum programs already implemented by a company in an isolated way, and new quantum programs that are expected to be developed to replace some parts of the legacy systems or to add new functionality (see Figure 3). In addition, these quantum descriptors will be located in the KDM repository with some relationships with classical elements. The extension is carried out through the standard extension mechanisms of KDM (i.e., family extensions, stereotypes and tagged values). Table 1 presents the available stereotypes (left column) as well the KDM elements where these could be applied (right column).

Figure 4 shows a running example to illustrate how KDM can be used to store knowledge discovered from a quantum program using parsers. The quantum algorithm in the example is a basic quantum entanglement algorithm. This algorithm is modelled both as a quantum circuit using the graphical notation according to *Quantum Programming Studio* [38] (see Figure 4 A), and in the respective Q# programming language (see Figure 4 B). The example shows a *Code Model* (see the KDM file to the right in Figure 4 C) as it is usually represented in KDM for other classical systems. However, in addition to this, this KDM model employs a family extension to define stereotypes that are specifically used in most of the quantum programming languages. Those stereotypes and the respective tagged values are then used in the KDM model to represent quantum

Table 1
Example of KDM stereotypes in the quantum KDM extension.

KDM stereotype	KDM element to be applied
quantum program	CompilationUnit
quantum operation	CallableUnit
qubit	DefinedType
quantum gate	ActionElement
qubit measure	
qubit reset	

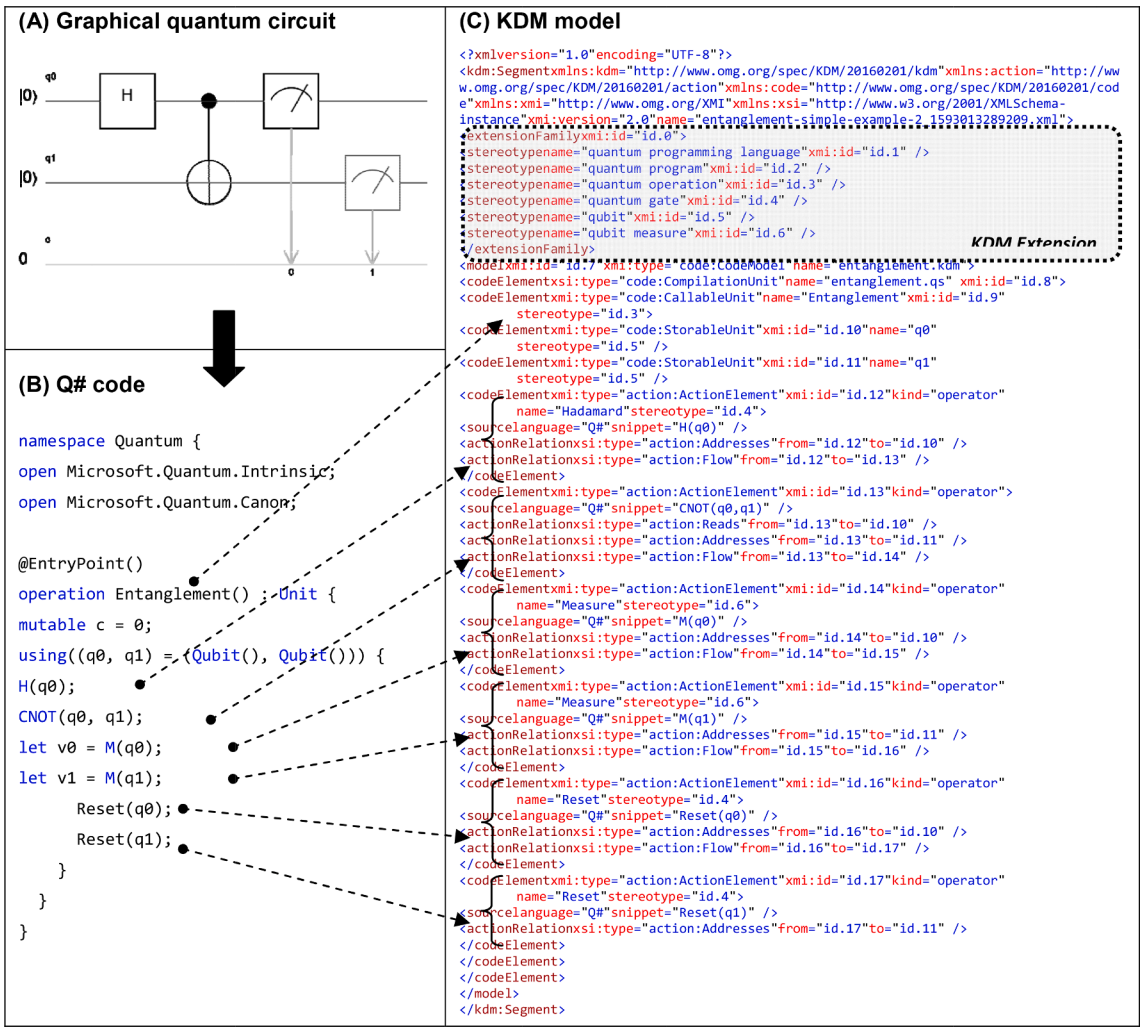

Figure 4. Running Example for an extended KDM model representing a basic quantum entanglement.

gates and other specific quantum knowledge. Sequence flow and operations with qubits are also represented in the KDM model with some *actionRelation* elements and attributes *from* and *to*. It should be noticed that together with the information abstracted in the KDM model, some code snippets might be added to the code elements (see Figure 4 C) in order to reflect the source code that originated each KDM element. In that way, it may prevent the loss of semantics.

3.2.2.2. Restructuring KDM/UML models for quantum computing. If new functionality is added during the restructuring phase (see Figure 3), new quantum computing concerns are added. In that case, extensions for the current UML diagrams, or new ones, should be provided, in order to collect quantum computing information in combination with analysis and design diagrams for classical information systems.

UML might be extended by following two different approaches [39]: i) a new metamodel by derivation of the UML metamodel, or ii) a UML profile, the standard extension mechanism of UML. First option is a heavyweight extension, since new meta-elements can be added, so the expressiveness is higher. However, standardization and conformance are poorer since new meta-elements will not be recognized by existing UML modelling tools. Whilst, An UML profile is a lightweight extension, since it is created by defining stereotypes, tagged values (based on tag definitions) and constraints (similar to what the KDM extension family does). UML profiles specialize UML for a concrete domain. In our proposal, for quantum computing, so-called Quantum UML profile. This profile might contain specific stereotypes to change the semantics of existing UML elements. Quantum UML profile is less expressive in comparison with the heavyweight extension mechanism but, in contrast, it is still UML-compliant, and existing modelling tools can ordinarily operate with this. Also, generative techniques consuming UML models as input will still works with the Quantum UML profile.

For example, we could use the stereotype <<qubit>> to annotate object instances in an object interaction diagram, so that such diagram can represent qubit interactions with regards to a quantum algorithm. Other example regarding UML use case diagrams is presented in Figure 5. The UML profile for this kind of diagram is defined with three stereotypes (see Figure 5 A). First, <<quantum request>> (that is applied for *include* elements) and is used for all the calls that classical (or other quantum) functionalities need to make to quantum algorithms. Second, the <<quantum algorithm>> stereotype that is used to annotate specific *use cases* that consists of functionalities that will be supported by quantum algorithms. Last, <<quantum environment>> might be used to stereotype *system* elements and group in this way all the use case of the hybrid systems that run in a quantum environment.

This phase focuses on the mapping definition between KDM and UML. Although there are some approaches for transforming KDM into UML, these proposals do not consider aspects for quantum programs. This phase considers mappings for the extensions of KDM and UML, together with the ordinary elements already defined in both standards. These mappings must be then implemented as model transformations that can be executed automatically. Model transformation will be eventually implemented according to the mapping provided in the

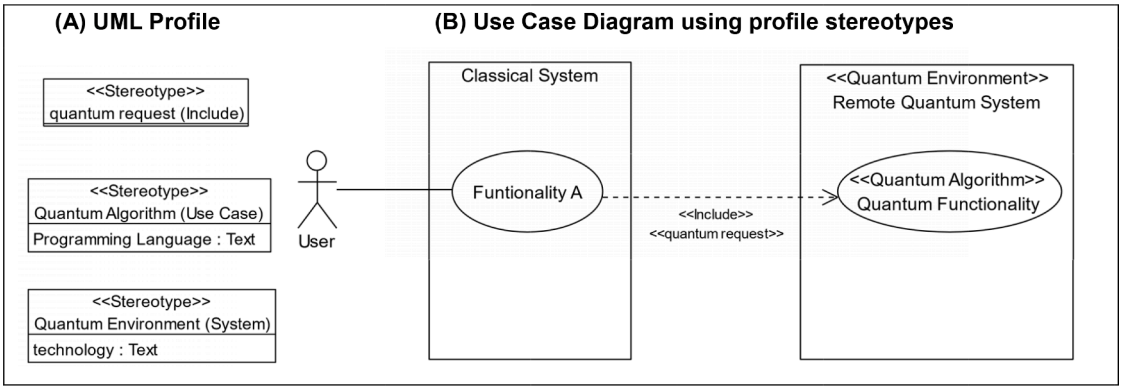

Figure 5. Example of use of Quantum UML profile with use case diagrams.

previous task. This transformation will be integrated with all other tools developed during the reverse engineering phase.

During the restructuring phase, refactoring of the existing quantum software application must be covered as well when those applications are migrated for different platforms [40, 41]. Thus, refactoring is also a mean to improve software quality during the restructuring phase, while the systems are migrated [41]. For example, this phase can detect some refactoring opportunities, some of them are based on antipatterns searches [42], while others look for similarities in models [43].

3.2.2.3. Generative techniques for classical-quantum information systems. This corresponds to the forward engineering phase. At the present time, there are many direct mappings, as well as the respective generators between UML and most common programming languages. This phase carries out further mappings between the UML extension for quantum aspects and the respective target quantum code (e.g., QisKit, QASM, Q#, etc.). The generative techniques consist of the detection of certain structures in some UML diagrams (persisted as XML files) and of building the respective code schema in the target programming language. The information coming from the UML structures and patterns, can be combined with information gathered from quantum circuits (related to UML models). In this sense, there already exist some approaches for transforming a graphical quantum circuit into various possible quantum programming languages [38].

Since the level of abstraction is reduced, further information could be needed to aid the code generation process and complete as many details as possible for the target system. For example, experts might define specific configurations that customizes some mapping between UML and code. This phase is able to generate code from UML specifications, combining classical code with some algorithms coded in quantum programming languages.

4. Implications for Researchers and Practitioners

QC is an ecosystem of recent technologies still under development. That being so, QC "needs a rich and diverse ecosystem: from research, through financing, to marketing—all players need to be active to bring about operational solutions that will have an overall impact" [44]. We provide a list of implications for researchers and practitioners:

4.1. Increasing Importance of Quantum Software

QC is currently at an important crossroads. High-level algorithms for quantum computers have shown considerable promise in the last years, and recent advances in manufacturing of QC devices are leading to quantum computers being used more and more. Nevertheless, a gap still exists between, on the one hand, hardware size and reliability requirements of QC algorithms, and on the other hand, the physical machines which will potentially be developed and deployed within the next few years. To bridge this gap, quantum computers require appropriate software if they are to translate and optimize applications (tool flows) and abstraction layers [4]. The future quantum developer will not be expected to have such in-depth expertise, just as modern-day programmers, for the most part, have a limited knowledge of hardware issues [13]. We believe this proposal will contribute to this regard.

4.2. Importance of using standards

As we stated in our proposal, the use of well-known standards in software engineering can help to bring those best practices and methods to the new QSE field. In our proposal we set forth how KDM and UML can help in the software modernization process by abstracting knowledge and contributing to systematic model-driven reengineering processes. Firstly, KDM helps with the abstraction of technological/vendor details, and collects all the system artefact knowledge in an integrated way. Secondly, UML helps to design and define the architecture of the target systems by combining classical and quantum systems. In addition to the use of KDM and UML, employing other standards to other areas of software engineering could be beneficial.

4.3. Transfer of results

Profitability success will be directly related to the ability to transfer results to the industry and transform the associated technologies into mainstream technology in future markets. The impact of research on industry is of the utmost importance in a field that is expected to show significant growth in the next years [15, 16]. It is also a field in which it seems that no software engineering methods, techniques and practices have been considered for this new scenario [5, 6, 45]. The practical and applied approach of the topics addressed in this proposal will make it easier to transfer the knowledge derived from this research to those companies in the sector who have projects that have to do with QC.

4.4. Lack of skilled professionals for quantum software modernization

Quantum computers not only require different programming languages, they also need a different approximation to the concept of programming [46], hence professionals with new skills are also required. So, the traditional shortage of software engineering professional, and especially "the lack of well-studied, experienced engineers with a formal and deep understanding of software engineering" [47] get worse in the case of quantum software engineers [48]. To reduce this gap, further competencies on quantum computing are required for software engineers [49]. Unfortunately, the majority of international curricula for computer science and software engineering degrees (as well as masters) do not include quantum computing competencies yet [50]. Attaining all these competencies is key to ensure the feasibility of software modernization of classical-quantum information systems.

5. Conclusions

This paper has highlighted the quantum computer science momentum, claiming that there is a need to research and carry out development in the quantum software engineering field. In our view, quantum physics, mathematics, computers, algorithms, and quantum computer science all manifest evident progress. Our opinion is, however, that quantum technologies and programming have not yet been addressed with techniques, good practices, and development methodologies for software engineering to meet the needs of quantum programs.

In an effort to reduce this gap, this paper proposes a software modernization process, i.e., a model-driven reengineering process, designed to cope with the migration of quantum algorithms alongside classical, legacy systems; it also addresses the integration of new quantum software during modernization of classical, legacy systems while knowledge is preserved.

The solution proposed is systematic and based on existing software engineering standards such as KDM and UML. It could therefore be applied in industry in a systematic way and in compliance with the existing software evolution processes.

This proposal focuses on the software modernization process by providing some high-level requirements and providing the steps necessary in each reengineering phase (i.e., reverse engineering, restructuring and forward engineering). This does not therefore provide a specific technique or method that can be applied for a certain technology. We have rather provided a generic process that can be implemented for different quantum technologies and programming languages. Nonetheless, future work should take on the task of conducting empirical validation of the software modernization process through the application of the suggested toolchain to real-life information systems and quantum algorithms.

Despite these limitations, this is probably the first time that the software modernization process has been specifically created and/or adapted for quantum technologies. This proposal will allow companies to reuse the knowledge embedded in legacy information systems while new quantum-based projects are delivered. Thanks to the use of KDM and UML, this proposal is independent of quantum programming languages, making its application feasible in the volatile technological environment that is expected to exist during the quantum computing revolution.

Declaration of Competing Interest

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

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Acknowledgements

This study has been partially funded by the following projects: SMOQUIN (PID2019-104791RB-I00) and ECLIPSE (RTI2018-094283-B-C31) funded by Spanish Ministry of Science and Innovation (MICINN).

References

[1] Maslov D, Nam Y, Kim J. An Outlook for Quantum Computing [Point of View]. Proceedings of the IEEE 2019;107(1):5–10. https://doi.org/10.1109/JPROC.2018.2884353.
[2] Langione, M., C. Tillemann-Dick, A. Kumar, and V. Taneja, Where Will Quantum Computers Create Value—and When? Boston Consulting Group. https://www.bcg.com/en-es/publications/2019/quantum-computers-create-value-when.aspx, 2019: p. 19.
[3] Mohseni M, Read P, Neven H, Boixo S, Denchev V, Babbush R, Fowler A, Smelyanskiy V, Martinis J. Commercialize quantum technologies in five years. Nature News 2017;543(7644):171.
[4] Chong FT, Franklin D, Martonosi M. Programming languages and compiler design for realistic quantum hardware. Nature 2017;549(7671):180.
[5] Clark J, Stepney S. Proposed "Grand Challenge for Computing Research" Quantum Software Engineering in 3rd meeting. Quantum Computing SIG 2002. University of York.
[6] Miranskyy A, Zhang L. In: On Testing Quantum Programs. in 2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER); 2019. https://doi.org/10.1109/ICSE-NIER.2019.00023.
[7] Pérez-Castillo R, de Guzmán IG-R, Piattini M, Knowledge Discovery Metamodel-ISO/IEC 19506. A standard to modernize legacy systems. Computer Standards & Interfaces 2011;33(6):519–32. https://doi.org/10.1016/j.csi.2011.02.007.
[8] Pérez-Castillo R, de Guzmán IG-R, Piattini M. Business process archeology using MARBLE. Information and Software Technology 2011;53(10):1023–44. https://doi.org/10.1016/j.infsof.2011.05.006.
[9] ISO/IEC, ISO/IEC 19505-1:2012. Information technology — Object Management Group Unified Modeling Language (OMG UML) — Part 1: Infrastructure. https://www.iso.org/standard/32624.html. 2012, ISO/IEC. p. 220.
[10] Martino Di, Rak B, M, Ficco M, Esposito A, Maisto S, Nacchia S. Internet of things reference architectures, security and interoperability: A survey. Internet of Things 2018;1:99–112.
[11] Moore, G.E., Progress in digital integrated electronics [Technical literaiture, Copyright 1975 IEEE. Reprinted, with permission. Technical Digest. International Electron Devices Meeting, IEEE, 1975, pp. 11-13.]. IEEE Solid-State Circuits Society Newsletter, 2006. 11(3): p. 36-37. DOI: 10.1109/N-SSC.2006.4804410.
[12] IBM Q, Introduction to Quantum Circuits. https://quantum-computing.ibm.com/support/guides/introduction-to-quantum-circuits. 2019, IBM.
[13] Yanofsky NS, Mannucci MA. Quantum computing for computer scientists. Cambridge University Press; 2008. n.p.
[14] Piattini, M., G. Peterssen, R. Perez-Castillo, J.L. Hevia, M.A. Serrano, G. Hernández, I.G.R.d. Guzmán, C.A. Paradela, M. Polo, E. Murina, L. Jiménez, J.C. Marqueño, R. Gallego, J. Tura, F. Phillipson, J.M. Murillo, A. Niño, and M. Rodríguez, The Talavera Manifesto for Quantum Software Engineering and Programming, in QANSWER 2020. QuANtum SoftWare Engineering & pRogramming, M. Piattini, Editors. 2020, CEUR-WS: Talavera de la Reina. p. 1-5.
[15] European Comission. OpenSuperQ Project. 2018 22/09/ 2019]; Available from: http://opensuperq.eu/project.
[16] Palmer, J., Quantum technology is beginning to come into its own. The Economist https://www.economist.com/news/essays/21717782-quantum-technology-beginning-come-its-own, 2017.
[17] IBM. IBM Q Experience. 2019; Available from: https://quantum-computing.ibm.com/.
[18] Knill E. Conventions for quantum pseudocode. NMUnited States: Los Alamos National Lab.; 1996.
[19] Ömer B. Quantum Programming in QCL. Institute of Information Systems. Technical University of Vienna; 2000. http://tph.tuwien.ac.at/~oemer/doc/quprog.pdf.
[20] Gay SJ. Quantum programming languages: Survey and bibliography. Mathematical Structures in Computer Science 2006;16(4):581–600.
[21] Selinger P. Towards a quantum programming language. Mathematical Structures in Computer Science 2004;14(4):527–86.
[22] Altenkirch T, Grattage J. QML: Quantum data and control. Submitted for publication 2005.
[23] Microsoft. Quantum Development Kit. 2019; Available from: https://www.microsoft.com/en-us/quantum/development-kit.
[24] IBM. QisKit. 2019 20/09/2019]; Available from: https://qiskit.org/.
[25] Fingerhuth M, Babej T, Wittek P. Open source software in quantum computing. PloS one 2018;13(12):e0208561.
[26] Cervera-Lierta, A., Quantum Computing languages landscape. https://medium.com/@quantum_wa/quantum-computing-languages-landscape-1bc6dedb2a35, Quantum World Association, Editor. 2018.
[27] Cross, A.W., L.S. Bishop, J.A. Smolin, and J.M. Gambetta, Open quantum assembly language. arXiv preprint arXiv:1707.03429, 2017.
[28] algassert.com. Quirk emulator. 2019; Available from: https://algassert.com/quirk.
[29] Harrow AW, Montanaro A. Quantum computational supremacy. Nature 2017;549:203. https://doi.org/10.1038/nature23458.
[30] Sommerville I. Software Engineering. 9 th Edition ed. Pearson. n.p; 2010. 792.
[31] Sneed HM. Estimating the Costs of a Reengineering Project. In: Proceedings of the 12th Working Conference on Reverse Engineering; 2005. IEEE Computer Society. n.p. 111 - 119.
[32] Müller HA, Jahnke JH, Smith DB, Storey M-A, Tilley SR, Wong K. Reverse engineering: a roadmap, in Proceedings of the Conference on The Future of Software Engineering. Ireland: ACM: Limerick; 2000. http://doi.acm.org/10.1145/336512.336526.
[33] Canfora G, Penta MD. New Frontiers of Reverse Engineering, in 2007 Future of Software Engineering. IEEE Computer Society 2007. https://doi.org/10.1109/FOSE.2007.15.
[34] OMG. Why do we need standards for the modernization of existing systems? OMG ADM Task Force 2003.
[35] Araujo, C. Is it the end of coding in the enterprise? ITSM & THE DIGITAL ENTERPRISE 2017 23/09/ 2019]; Available from: https://www.cio.com/article/3242253/is-it-the-end-of-coding-in-the-enterprise.html.
[36] Moyer B. Software Archeology. Modernizing Old Systems. Embedded Technology Journal 2009;1:1–4.
[37] Khusidman V. ADM Transformation White Paper. DRAFT V.1. OMG 2008:5. http://www.omg.org/docs/admtf/08-06-10.pdf.

[38] Quantastica. Quantum Programming Studio. 2019 [cited 2020; Available from: https://quantum-circuit.com/.
[39] Ribo JM, Franch Gutiérrez J. A two-tiered methodology to extend the UML metamodel. Universitat Politècnica de Catalunya 2002:1–16. https://upcommons.upc.edu/bitstream/handle/2117/97437/R02-52.pdf.
[40] Santos BM. I.G.d. Guzmán, V.V.d. Camargo, M. Piattini, and C. Ebert, Software Refactoring for System Modernization. IEEE Software 2018;35(6):62–7. https://doi.org/10.1109/MS.2018.4321236.
[41] Pérez-Castillo R, Fernández-Ropero M, Piattini M. Business process model refactoring applying IBUPROFEN. An industrial evaluation. Journal of Systems and Software 2019;147:86–103. https://doi.org/10.1016/j.jss.2018.10.012.
[42] Arcelli D, Cortellessa V, Di Pompeo D. Performance-driven software model refactoring. Information and Software Technology 2018;95:366–97. https://doi.org/10.1016/j.infsof.2017.09.006.
[43] Ghannem A, El Boussaidi G, Kessentini M. On the use of design defect examples to detect model refactoring opportunities. Software Quality Journal 2016;24(4):947–65. https://doi.org/10.1007/s11219-015-9271-9.
[44] Gayrard R, Carrière F, Darde B, Jaunay J. Quantum computing: ready for the huge leap? 2019. A France digitale / Wavestone study 2019.
[45] Computer Science Degree Hub. What is a Quantum Computer? 2019 22/09/2019]; Available from: https://www.computersciencedegreehub.com/faq/what-is-a-quantum-computer/.
[46] Knight W. Serious quantum computers are finally here. What are we going to do with them. MIT Technology Review 2018:30.
[47] Baker, J., 2018′s Software Engineering Talent Shortage—It's quality, not just quantity. 2017.
[48] Hilton J. Building The Quantum Workforce Of The Future, in Forbes Technology Council. Forbes 2019. https://www.forbes.com/sites/forbestechcouncil/2019/06/19/building-the-quantum-workforce-of-the-future/#5dd462bbfa47.
[49] Kiefl, N. and G. Hagel, Software Engineering Education of Classical Computing vs. Quantum Computing: A Competency-Centric Approach, in Proceedings of the 4th European Conference on Software Engineering Education. 2020, Association for Computing Machinery: Seeon/Bavaria, Germany. p. 27–31. DOI: 10.1145/3396802.3396816.
[50] Piattini M. Training Needs in Quantum Computing, in QANSWER'20 - QuANtum SoftWare Engineering & pRogramming, M. Piattini, et al., Editors. Talavera de la Reina. CEUR-WS 2020:23–30.