Computing (2025) 107:111
https://doi.org/10.1007/s00607-025-01467-2

SPECIAL ISSUE ARTICLE

Exploring design patterns in quantum software: a case study

Miriam Fernández-Osuna[1] · Ricardo Pérez-Castillo[1] · José A. Cruz-Lemus[2] · Michal Baczyk[2] · Mario Piattini[2]

Received: 31 October 2024 / Accepted: 18 March 2025 / Published online: 11 April 2025

Abstract

Quantum computing holds great promise for solving complex problems that classical computing cannot address, with applications in various industries and sectors. However, developing efficient quantum software remains a challenge. Quantum software engineering (QSE) has emerged to address this, adapting classical software engineering practices for quantum systems. Design patterns, widely used in classical software, can provide reusable solutions for common quantum development issues, but their use in quantum software remains underexplored. This paper presents an empirical study investigating the use of design patterns in 2610 Qiskit programs from GitHub. Using the *QCPD Tool* to detect four design patterns (*initialization, superposition, entanglement, and oracle*) and *QMetrics* to compute software metrics, the study creates a dataset linking patterns with code characteristics. Three research questions guide the study: RQ1 examines the prevalence of design patterns in quantum software, RQ2 explores the relationship between design patterns and code metrics, and RQ3 analyzes the combinations of patterns that occur together. The findings provide insights into quantum software development, offering developers practical guidance on applying specific patterns. The results contribute to QSE by revealing key relationships between patterns and metrics, which can inform future research and tool development. These findings support improved performance, maintainability, and scalability, fostering the broader adoption of quantum computing.

Keywords Quantum software engineering · Quantum computing · Design patterns · Quantum circuit metrics · Code repository analysis

Mathematics Subject Classification 68N01 · 68Q09 · 81P68

Extended author information available on the last page of the article

1 Introduction

Quantum computing has emerged as a transformative technology capable of solving specific types of problems that are intractable for classical computing methods [2]. Its potential to revolutionize a wide range of industries, from pharmaceuticals and materials science to finance and logistics, positions quantum computing as a critical driver of innovation across many sectors Ukpabi et al [47]. This technology holds promise for addressing complex computational challenges, such as optimization, cryptography, and machine learning, which are beyond the reach of conventional software solutions.

As quantum computing continues to advance, its impact on the global digital economy is expected to be profound. Industry analysts anticipate significant growth in the value that quantum computing can create, driven by its ability to accelerate breakthroughs in research, reduce operational costs, and enhance decision-making processes. With high expectations for commercial adoption and widespread integration into critical industries, quantum computing is poised to become a powerful catalyst for the future of the digital economy [25, 48].

Despite such promising claims, [39] pointed out that some challenges to achieving practical applications in quantum computing exist, particularly in the NISQ (Noisy Intermediate-Scale Quantum) era, such as error correction, qubit decoherence, and scalability. Even so, a recent study by Kim et al [24] provide evidence for the utility of quantum computing even before full fault tolerance is achieved.

Although early demonstrations from studies [28, 36, 37] highlight the potential of quantum computing, its full advantages cannot be realized through hardware alone. Quantum software is essential, playing a crucial role in leveraging quantum computers. The field of Quantum Software Engineering (QSE) has emerged to address this need, drawing from classical software engineering practices. QSE focuses on adapting or developing new processes, methods, and techniques to create effective quantum software.

According to Murillo et al [29], QSE, as emerging field, addresses key challenges associated with quantum software, including:

- *Complexity of quantum algorithms*: [46] says "Quantum algorithms are significantly more complex than classical ones and require developers to approach problems in innovative ways". Adapting existing algorithm sets to solve specific problems is not straightforward.
- *Lack of abstraction*: Quantum software is still defined at a low level Baczyk et al [4]. Carleton et al [7] detects the need for greater abstraction in programming languages to facilitate its development, also [32] the techniques to analyze and design quantum software at higher levels of abstraction, along with a specific paradigm oriented toward quantum computing [1], are also crucial.
- *Integration of classical and quantum software*: Quantum software cannot be developed or operate in isolation Pérez et al [33], and Carleton et al [7]

note: "One challenge is to fully integrate these types of systems into a unified environment that combines classical and quantum software within a cohesive development life cycle". As Khan et al [23], Garcia et al [18] state, new architectural paradigms will be required for the effective development of quantum software.

The use of design patterns offers a promising approach to addressing the challenges of quantum software development. Just as design patterns have provided effective solutions for well-known problems in classical software development Gamma et al [17], they can similarly benefit quantum software Khan et al [23], Jiménez et al [22]. Several design patterns have been proposed and thoroughly described for quantum software, such as those outlined by [6, 26]. For instance, the *uniform superposition* pattern initializes qubits using *Hadamard* gates to create an equally weighted superposition, a common requirement in many quantum algorithms. The significance of quantum software patterns lies in their ability to: (i) promote best development practices, (ii) enable the reuse and adaptation of quantum algorithms, and (iii) enhance the quality of quantum software, including aspects such as performance, maintainability, and scalability.

Despite the potential advantages of using design patterns in quantum software, there is still a lack of robust validation and comprehensive characterization of their application across a broad range of quantum programs. To address this gap, this paper presents an empirical study aimed at inspecting the usage of specific quantum software patterns in 2610 quantum programs in GitHub. The research adopts a code repository analysis approach, gathering Qiskit code for analysis. The programs are then examined using the *QCPD Tool* Romero et al [41], designed to recognize four types of quantum design patterns (*initialization, superposition, entanglement and oracle*), and *QMetrics* Cruz-Lemus et al [9], which computes quantum software metrics. Both tools enable a detailed analysis of patterns and code metrics. The result of this analysis is a dataset that links the quantum design patterns identified in each quantum program with a corresponding set of metrics, providing a comprehensive view of the relationship between pattern usage and various software characteristics.

The primary contribution of this research is to provide a comprehensive analysis of how design patterns and code metrics are used in quantum software, addressing three key research questions. Firstly, RQ1 explores the extent to which design patterns are utilized in quantum software, offering a detailed representation of their prevalence. Secondly, RQ2 examines the relationship between the usage of design patterns and specific code metrics, uncovering how certain metrics influence the presence of specific patterns. Thirdly, RQ3 analyzes how is the probability of some design patterns to be applied in combination, which could suggest resource allocation strategies and might contribute to the modular and automated design approaches in quantum software. By answering these research questions, this study offers new insights into the structure and development of quantum software, contributing to the evolving discipline of quantum software engineering.

The implications of this study are significant for both researchers and practitioners in the field of quantum computing. For developers, the findings provide practical guidance on how and when to apply specific design patterns, enhancing the quality

of quantum software in terms of performance, maintainability, and scalability. For researchers, the insights gained from the pattern and metric analysis can help inform the development of new patterns and tools, driving future innovations in quantum software engineering. Furthermore, the study promotes the transfer of knowledge and practices in quantum computing to industry, helping in the transition from experimental to practical, scalable quantum solutions.

The rest of the document is structured as follows. Section 2 discusses everything related to the current state of quantum software and states the motivation for the development of this study; Sect. 3 presents related work on quantum design patterns analytics; Sect. 4 shows the descriptive case study conducted, both the study design and result analysis; finally, Sect. 5 discusses conclusions and future work.

2 Background

This section aims to put into context everything related to this study, such as quantum computing (see Sect. 2.1) and quantum software engineering (see Sect. 2.2), as well as the current state of the art and everything related to metrics and patterns (see Sect. 2.3) being the key factors of this project and Big Data as a promoter in the treatment and management of data.

2.1 Quantum computing

Quantum computing is emerging as a fundamental advancement in data processing capabilities, able to tackle problems that exceed the reach of classical computing. Based on the counterintuitive principles of quantum mechanics, such as superposition and entanglement, quantum computing offers an architecture that, in theory, enables exponentially more efficient calculations in certain scenarios. Problems like large-number factorization, crucial for cryptography, or the simulation of molecular interactions in chemistry and materials science are examples of areas where quantum systems may provide solutions that would be intractable or take excessive time on classical computers. This potential has captured the attention of researchers and developers, who have begun developing quantum programming languages, libraries, and tools to facilitate the implementation and execution of quantum algorithms. To fully leverage these capabilities, Serrano et al [45] indicates that it is essential not only to ensure the technical viability of quantum systems but also to establish rigorous quality standards for the development and evaluation of quantum software. However, to contextualize more in depth on this new technology is not the objective of the article, it would be highly recommended to read [12] for more context.

2.2 Quantum software engineering

Quantum software engineering, which merges classical software engineering with quantum computing, encompasses everything from the creation of quantum

algorithms and circuits to the formalization of quantum applications. The goal is to design, develop, test, and maintain software that functions effectively on quantum computers Piattini et al [36]. Key areas within quantum software engineering include the simulation of quantum systems, the optimization of quantum algorithms, and, in particular, the topics addressed in this study: measuring and evaluating the quality of quantum software through the detection of design patterns and the calculation of metrics.

One key approach to maximizing the potential of quantum computing is hybrid software, which combines classical and quantum software to solve complex problems. These hybrid systems must seamlessly integrate with existing technological information infrastructure in organizations, ensuring interoperability between classical and quantum components. Díaz et al [11], Peterssen et al [35] indicates as quantum computing continues to evolve, it becomes crucial to assess the quality of these hybrid systems to ensure their efficiency and performance.

Two models are used to represent these programs: in the classical model, the entire program, including quantum code (Qiskit), is written in Python, while in the quantum model, quantum circuits are abstracted from the code. This combination allows for valuable relationships to be established between classical and quantum components. This enables the identification of classical data that manages quantum results or classical code that dynamically generates quantum circuits. Furthermore, the analysis of hybrid programs is enhanced by the use of a parser Jiménez et al [21].

2.3 Quantum design patterns

Research on quantum software design patterns has evolved from early circuit-level patterns to encompass higher-level architectural concerns. This evolution noted by Weigold et al [50], Weder et al [49], Bühler et al [6] reflects the growing maturity of quantum software engineering, with contributions spanning from foundational circuit patterns to hybrid system architectures. While there were some early studies about two decades ago from Raussendorf et al [40], they primarily focused on measurement patterns in one-way quantum computing (1WQC). The 1WQC model operates by performing a series of measurements on qubits within a specific multi-qubit entangled state.

A significant milestone in quantum pattern development was [26]'s proposal of a comprehensive pattern language for quantum algorithms, which established foundational patterns like *Initialization and Uniform Superposition*. This work has since been expanded by Khan et al [23] who have contributed additional patterns for data encoding, error handling, and hybrid quantum-classical integration. This work presents a comprehensive and well-documented catalog of software patterns designed to address common challenges in quantum circuit design. It identifies ten core patterns, including *Initialization, Uniform Superposition, Creating Entanglement, Function Table, Oracle, Uncompute, Phase Shift, Amplitude Amplification, Speedup by Verification, and Quantum-Classic Split*. In addition to these, fifteen more patterns are suggested for specialized applications, such as various encoding methods or for use in variation of quantum algorithms [5, 38].

This study focuses on *Initialization, Uniform Superposition, Creating Entanglement and Oracle*, as some of the most fundamental patterns. This design decision is due to two main reasons. Firstly, these patterns are considered some of the most essential patterns that are generally applicable in a broader set of quantum algorithms. Secondly, the set of patterns was limited to a set that can be identified by a sole tool (*QCPD*) to avoid the use of different tools to detect a wider set of patterns. Thus, it prevents to achieve a non-comparable subset of results.

The current study narrows its focus to four of these quantum circuit patterns, which are briefly outlined below:

- *Initialization*: The quantum register input is initialized considering the prerequisites of the subsequent steps of the algorithm. Frequently, the unit vector is used as initialization of a quantum register.
- *Uniform superposition*: This initializes some qubits of a quantum register by applying *Hadamard* gate to obtain an equally weighted (uniform) superposition. This kind of initialization is common in many quantum algorithms where the measurement probability must be initially the same.
- *Creating entanglement*: It enforces a strong correlation between qubits by entangling them, which is usually required to achieve an exponential speedup over classical algorithms.
- *Oracle*: Re-use a computation of a quantum algorithm as a black box. It is used by following a Divide-and-Conquer approach, since the use of oracles can simplify solving a complex problem. Thus, multiple oracles can be used as building blocks to compose larger algorithms. The oracle implementation varies for different algorithms since it is quite problem specific.

All the patterns discussed in this section leverage the unique properties of quantum states. For instance, *amplitude amplification and speedup by verification* use *superposition and entanglement* to enhance both the accuracy and efficiency of quantum algorithms. On the other hand, quantum-classical partitioning integrates classical and quantum resources to optimize the performance and applicability of quantum systems.

The evolution of quantum software patterns has progressed beyond circuit-level concerns to encompass high-level architectural patterns. Recent work has proposed patterns for distributed quantum computing Furutanpey et al [16], quantum resource management Khan et al [23], and hybrid quantum-classical workflows Georg et al [19], addressing the full spectrum of quantum software engineering challenges. For example, Khan et al [23] conducted a systematic literature review in this area. Their study highlighted that while classical architectural patterns like “layered” and “pipe-and-filter” have been adapted for hybrid architectures [32], new quantum-specific architectural patterns have emerged from studies Furutanpey et al [16], Khan et al [23] to address unique challenges in quantum resource management and quantum-classical integration.

Design patterns in quantum computing serve multiple critical functions: they provide reusable solutions for common quantum algorithmic challenges, establish standardized approaches for quantum-classical integration and create a shared

vocabulary for quantum software developers. Additionally, they improve software quality, as these patterns are proven across multiple projects. Finally, they foster clearer communication between engineers and developers by providing a shared language and common concepts.

3 Related work on design patterns analytics

The analysis of classical software repositories is a widely used method in numerous studies Sayago et al [44]. Several works in the literature examine code repositories to explore the use of design patterns in classical software. For example, Aversano et al [3] investigated the evolution of design patterns by analyzing changes in code versioning through domain matching. Their approach compares software project profiles, built using domain-representative keywords from each developer's code.

In the context of quantum software, repository analysis has also been applied for various purposes. Paltenghi and Pradel [31] introduced LintQ, a static analyzer for Qiskit programs. They analyzed 7,568 real-world Qiskit-based quantum programs collected from GitHub to assess the frequency of specific bugs in quantum code. Similarly, Fingerhuth et al [15] evaluated open-source tool chains for quantum software development, assessing aspects like documentation, licensing, programming language choices, adherence to software engineering standards, and project culture.

The article Pérez et al [34], shares some initial similarities as it represents a preliminary study with a limited scope. However, this research expands that line of inquiry with key contributions. Unlike the preliminary study, a customized static analyzer was developed and utilized, specifically designed to enhance the exploration of quantum code and enable more precise and robust detection of design patterns. Furthermore, this study includes a longitudinal analysis of the usage of patterns over the years, offering a historical perspective absent in the preliminary work.

In contrast, Li et al [27] analyzed StackExchange, a Q&A website, to identify the key challenges in Quantum Software Engineering (QSE) as perceived by developers. Additionally, De Stefano et al [10] conducted a survey to assess the current adoption of quantum programming practices, gathering participants by analyzing quantum software repositories on GitHub.

Despite these efforts, no specific studies have been found that analyze or characterize the usage of quantum software design patterns.

The increasing complexity of quantum computing programs leads to a higher probability of errors and greater difficulty in understanding them, a problem that has also been common in classical computing Nayak et al [30]. For this reason, various research efforts, tools, and techniques have been developed for detecting, analyzing, and correcting software errors through the use of static code analysis in quantum programming. This type of analysis involves identifying, classifying, and understanding the most common faults without the need to execute the program. In this project, for the analysis and processing of programming languages, the AST (Abstract Syntax Tree) structure is used, a syntactic representation of programs.

There are specialized libraries like ANTLR4 and *ast* that allow the generation of AST structures for different languages. Although both libraries were used in this study, *ast* was the tool chosen to perform the static code analysis. This Python-native library facilitates the detection of errors and identification of potential improvements in the code without executing it, providing detailed information about the structure of the code, such as class hierarchies, functions, and variables. Additionally, ast allows for the evaluation of code complexity and optimization, ensuring compatibility with the latest Python versions. As a result, better outcomes were obtained using this library.

4 Descriptive case study

This section presents an exploratory case study to depict the actual usage of design pattern in quantum software programs. Section 4.1 outlines the case study design utilizing a Big Data methodology. Section 4.2 presents the analysis performed from the case study. Finally, Sect. 4.3 shows the threats to the validity and limitations of this study.

4.1 Case study design

This study follows the protocol proposed by Runeson et al [43] for designing and conducting case studies. Based on this protocol, data are collected, ensuring that planned decisions and procedures are followed consistently.

4.1.1 Rationale and objectives

As mentioned in related work (see Sect. 3), quantum software development has been mainly handcrafted and lacks a mature methodology due to the novelty and constant evolution of this technology. This motivates an in-depth exploration of its characteristics, including a reflection on the applicability of design patterns in quantum programs available in code repositories (particularly GitHub).

This exploration focuses on a subset of most basic and foundational design patterns (such as *Initialization, Uniform Superposition, Entanglement Creation, and Oracle*). This set of target patterns is selected based on their general applicability and relevance as essential building blocks of quantum algorithms. This selection was also driven by the need to ensure consistent detection criteria and comparable results, which could not be possible in case of using further tools covering a wider set of design patterns. Although this design decision limit the scope up to a certain extent, it provides a systematic and comparable analysis, fully supported by the *QCPD Tool*, while providing a foundation for future studies that may analyzing additional patterns.

It should be noted that this case study is not aimed at validating a specific set of design patterns or the tools used to detect such patterns. Instead, this is a descriptive case study to explain the practical usage of design patterns. The main objective of

this case study is, therefore, to identify and analyses the usage of design patterns and the characterization of quantum programs (through a set of code metrics) in which those patterns are applied.

4.1.2 Research questions

In the presented case study, three main research questions have been identified. These questions relate to the representation of design pattern usage in quantum software, the relationship between design patterns and code metrics, and the combinations of design patterns detected in quantum software.

RQ1. What is the degree of use of design patterns in quantum software?

RQ2. Is there a relationship between the usage of design patterns and code metrics?

RQ3. What is the probability of the application of various design patterns in combination?

The research questions are focused on quantitative analysis because they seeks to measure specific factors, which implies the collection of numerical data. A common feature of a quantitative analysis is that it seeks to be replicable (to some extent); that is why others could conduct studies similar to the one proposed in this project and obtain comparable results. Also, it allows for an in-depth exploration of the contexts, meanings and patterns underlying quantum software. This approach provides a holistic view, facilitating the identification of trends and relationships. In this way, the study not only quantifies relevant metrics, but also interprets the dynamics and behaviors that influence the implementation of quantum design patterns, thus strengthening the validity and applicability of the findings.

In particular, RQ3 analyzes the probability of certain design patterns being applied in combination, offering insights into dependencies between pattern implementations and suggesting strategies to optimize the efficiency and scalability of complex quantum systems. For instance, identifying such combinations can guide resource allocation in QPUs, considering factors like the number of qubits or specific gate sets, and improve the success rates of algorithms tailored to specific applications. Furthermore, detecting combinations of patterns facilitates the design of modular systems where specialized subcomponents, such as oracle blocks or entanglement structures, can be seamlessly integrated. This information could also serve as a valuable foundation for assisted circuit design tools, enabling the tracing of key architectural decisions and enhancing automated design processes.

4.1.3 Measures

The case study considers two main measurable concepts, design patterns (needed for the three research questions) and code quality (specially needed for RQ2). Both measurable concepts defined several metrics. The metrics related to design patterns address the three research questions presented in Sect. 3. These metrics include the number of design patterns (*initialization, superposition, oracle and entanglement*) detected in some of the analyzed quantum circuits. In terms of code quality, the second research question is answered by evaluating quantifiable features in the

circuits, such as circuit size and density, the use of single qubit gates, multiple gates, measurement operations and the implementation of oracles. These metrics are based on those proposed in Cruz-Lemus et al [8], and are then listed in result analysis (cf. Sect. 4.2.2).

4.1.4 Case selection

A quantum program as coded in a single file is the analysis unit for the study of the application of design patterns. Thus, Table 1 shows the selection criteria established to be applied to quantum programs.

C1 refers to GitHub, the most widely used platform for open-source quantum software. The GitHub API allows efficient data extraction and analysis, making it ideal for collecting and studying quantum programs. These features ensure transparency and support large-scale research. The C2 criterion is based on the fact that Qiskit is one of the most widely used programming languages for quantum software development. C3 relates to the date range for data collection, starting from 2017, as this was when Qiskit was released and began to gain popularity. C4 is about the ability to obtain a quantum circuit in RQCR (Reduced Quantum Circuit Representation) format from an original code file IBM Quantum [20]. The RQCR format is an intermediate representation of the quantum circuit best suited for the state machine based pattern detection technique carried out by the *QCPD Tool*. To achieve this, the file must contain quantum circuit functionality, enabling static code analysis with the *ast* library (C5). The resulting data obtained from the application of these criteria can be found in Sect. 4.1.5.

4.1.5 Data collection

This section explains how the dataset was collected to study design patterns. The process was divided into two main stages: the ingestion of quantum software programs from GitHub; and the detection of software patterns for the collected code.

The first phase is dedicated to the ingestion of source code files from GitHub. For this purpose, a Python script was coded to query the GitHub API in a thorough and systematic manner. The process supported by this script is shown in Fig. 1.

Table 1 Criteria for case selection

ID	Criterion for case selection
C1	It must be stored in GitHub
C2	It must be coded in Qiskit (Python)
C3	It must be lastly modified between January 1, 2017 and September 16, 2024
C4	It must be able to be transformed into the RQCR format
C5	It must define, at least, a quantum circuit

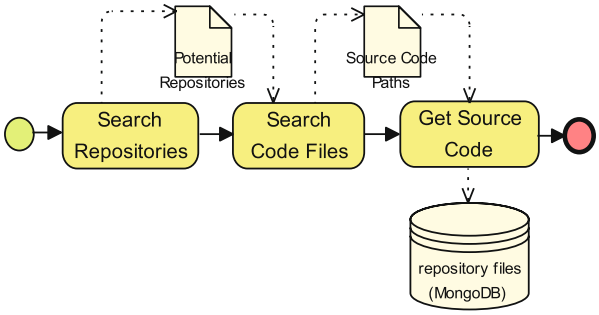


Fig. 1 GitHub Quantum Source Code Ingestion

- First, the script looks at all repositories that may contain quantum source code. This query is parameterized by the programming language (Python) and the Qiskit extension. To find quantum-based repositories using Qiskit library, which is not directly supported by the GitHub API, we search for that string in the repository's name, description and topics, as well as in the repository's *README* file. Although this search strategy will result in lower precision, that is, with many false positives, it nonetheless guarantees the highest recall. Additionally, repositories are filtered by date and progressively consider repositories within the proposed date ranges. Since the date range spans more than one year, requests to the GitHub REST API are made separately for each year, as well as by programming language.
- Having detected repositories with potential quantum programs, all files in the repository are then recursively inspected. The collected repositories do not only contain source code files written in Qiskit. Instead of that, the repositories could be a mixture of files written in different programming languages. Therefore, the second step (see Fig. 1) consists of filtering the source code files using Qiskit (Python). Similarly to the previous step, the files in each repository are queried again by language (Python) plus the search string 'Qiskit' in the content file.
- Once the paths to the source code files are obtained, the content of those files (the source code) is accessed via the respective GitHub operation.

Once the quantum software code has been collected from GitHub, the next step is to analyze the source code and detect design patterns. Figure 2 summarizes that process.

The first step is to load the accessed source code into a Mongo database. After that, in order to filter out quantum source files that could be considered false positives. After applying the search filters mentioned in the previous steps, some source code files might have been picked up without supporting real quantum circuits in Qiskit. This is due to the fact that some matches occurred due to some commented code, or even certain elements of Qiskit leading to false positives. Thus, the script search files containing the expression *'QuantumCircuit'*, which

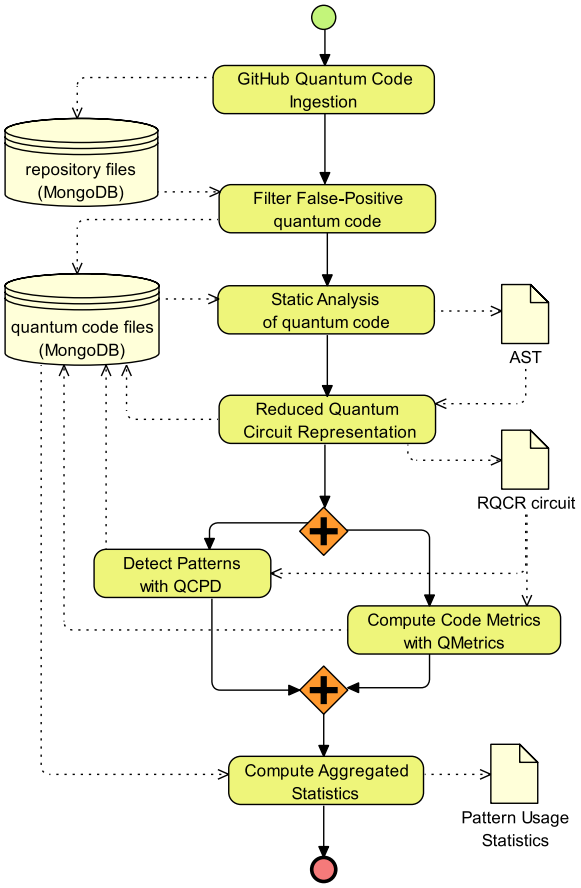


Fig. 2 Quantum software design pattern detection process

is used for the minimal definition of a quantum program. A filtered collection is then loaded in MongoDB with quantum programs.

The next step is to statically analyze the internal structure of the quantum code by means of the Python's *ast* library. This library is able to build the respective Abstract Syntax Tree (AST) for a quantum program. An example of this structure is shown in Listing 2, which has been obtained from a simple quantum circuit (see Listing 1).

```
1 from qiskit.circuit import QuantumRegister, QuantumCircuit
2
3 q = QuantumRegister(2)
4 circuit1 = QuantumCircuit(q)
5 circuit1.h(1)
```

Listing 1 Simple quantum circuit defined in Qiskit.

```
Module(
  body=[
    Assign(
      targets=[Name(id='q', ctx=Store())],
      value=Call(
        func=Name(id='QuantumRegister', ctx=Load()),
        args=[Constant(value=2)],
        keywords=[])),
    Assign(
      targets=[Name(id='circuit1', ctx=Store())],
      value=Call(
        func=Name(id='QuantumCircuit', ctx=Load()),
        args=[Name(id='q', ctx=Load())],
        keywords=[])),
    Expr(
      value=Call(
        func=Attribute(
          value=Name(id='circuit1', ctx=Load()),
          attr='h',
          ctx=Load()),
        args=[Constant(value=1)],
        keywords=[]))
  ]
)
```

Listing 2 AST code representation of a quantum circuit using Qiskit.

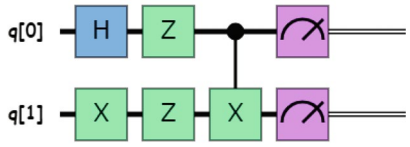
As explained in Sect. 4.1.4 The RQCR format is an intermediate representation of the quantum circuit that is more suitable for the state machine-based pattern detection technique. Thus, the quantum circuit is traversed in columns, and the quantum gates applied to different qubits are summed in a row. RQCR is serialized as a JSON file. In Fig. 3, it is shown as the fourth Bell state encoded in Qiskit (a), which is graphically shown in (b) and its representation in RQCR format (c).

It should be noticed that there are different grammars for Qiskit, as this extension has evolved intensively for the last years. Similarly, Python has changed as well. As a result, programs coded by using different versions of Python and Qiskit are in our repository. Although the construction of the AST representation is direct, the transformation from the AST to the RQCR is not trivial in every case.

For this study, the script support a fully customized AST interpreter designed to accommodate, up to a certain extent, different versions of Qiskit and ensure compatibility with the desired RQCR format. This interpreter modularly and systematically handles the parsing of both simple and complex quantum gates, oracles, and other circuit elements across different versions. The customization includes support for diverse methods of invoking quantum registers, constructing quantum circuits, and interpreting keywords within complex gates and arrays. Additionally, the script allows processing of multiple quantum circuits from a single source file and facilitates the application of quantum gates to specific qubits within arrays

```
q = QuantumRegister(2, 'q')
c = ClassicalRegister(2, 'c')
circuit = QuantumCircuit(q, c)
circuit.h(q[0])
circuit.x(q[1])
circuit.z(q[0])
circuit.z(q[1])
circuit.cx(q[0], q[1])
circuit.measure(q, c)
```

(a)



(b)

```
[["H","Z",{"CONTROL":1},"MEASURE"],["X","Z
","X","MEASURE"]]
```

(c)

Fig. 3 Reduced Quantum Circuit Representation Format

Due to the complexity of the source code to be transformed to the RQCR, not all parsed files are eventually generated in this format. For example, there are some quantum circuits that are dynamically built based on auxiliary code that cannot be directly retrieved through static analysis. As a result, the amount of programs passing through this step is reduced.

The last step of this process is the detection of design patterns. This step is supported by the use of a previously created project, called *QCPD Tool*, by Romero et al [41, 42], the use of this tool is taken as a black box. But at this point, it is necessary to know how it works in order to obtain the results. This project consists of the definition of a state machine that can analyze the quantum circuit, column by column, and process in parallel possible sequences of gates applied to different qubits.

Thus, the technique supported by *QCPD Tool* manages a stack of the gates found and assigns them to the defined state machine. In this way, it finds fragments of the given circuit that can be a possible design pattern. This technique focuses on five patterns, as mentioned in Sect. 2.3 (*Initialization, Uniform Superposition, Entanglement Creation and Oracle*). Thus, the state machine defines the following states for these five patterns:

- *S0-Initial State*: New column found. No indication of any potential pattern is currently being processed.

- *S1-Initialization pattern potentially found*: An exact and deeper analysis for initialization is needed.
- *S2-Uniform Superposition pattern potentially found*: An exact and deeper analysis for superposition is needed.
- *S3-Oracle pattern potentially found*: An exact and deeper analysis of the oracle is needed.
- *S4-Remaining gate found*: Neutral state, just symbol shifting.
- *S5-A set of at least one qubit is superposed, but not all of them*: First out of three atomic components of entanglement found. Notification to the exact checker for further entanglement search is needed.
- *S6-Superposed sub-set of qubits and new column found*: Two out of three atomic components for entanglement were found.
- *S7-Entanglement potentially found*: All atomic components exist. An exact and deeper analysis for entanglement is needed.
- *S8-Final State*: Pattern search finished.

Although the usage of *QCPD Tool* can significantly affect the results of this study, the validation of the *QCPD Tool* is outside of the scope of this study. A previous validation of this tool has been conducted in Romero et al [42] through its application to some quantum circuits tagged by some experts with the expected patterns. The findings revealed a high accuracy rate and a significant level of agreement between the tool's detections and the experts' evaluations, demonstrating that the tool accurately interpreted the circuits regarding pattern detection and recommendations.

QCPD Tool does not have an API to which queries can be made, so it was integrated in this study.

Due to our previous preliminary study, we figure out *QCPD Tool* did not detected, in its current form, the *entanglement* pattern properly. In order to have a dataset suitable for this case study, we coded a isolated script for performing the detection of the *entanglement* pattern.

Quantum *entanglement* detection scans each qubit in the quantum circuit for a *Hadamard* gate. It then scans to see if a control gate appears after it, indicating a possible interaction with another qubit. If that controlled qubit has a *Pauil-X* gate in the correct position and there are no other previous operations, the function concludes that *entanglement* is present. Although there are more situations where the pattern of *entanglement* occurs, the project has considered the most common case [13].

Finally, along with the detection of quantum circuit design patterns, the script computes the code quality metrics (see Table 4) associated with the quantum circuit.

As a result, a dataset with quantum circuits, design patterns and metrics to characterize each circuit is obtained. The whole dataset is available in Fernández et al [14]. To understand the performance of the data collection process, Table 2 provides absolute and relative numbers in every step of the process.

A total of 9646 files were ingested, and 1763 false positive quantum files were discarded, leaving 7883 programs classified as quantum, representing 81.72% of the total. Subsequently, the files were analyzed using the *ast* parsing library, suggesting

Table 2 Summary of data collection

	Python-Qiskit	% of quantum programs
Ingested files	9646	–
False positive discarded	1763	–
Quantum programs	7883	81.72%
Valid AST	7843	81.31%
Files with RQCR circuit	2610	33.11%
Generated RQCR circuit	3719	–
Circuits with metrics	2252	28.57%
Circuits with design patterns	934	11.85%

that the programs were structurally evaluated with high coverage, as 99.5% of the quantum programs were successfully processed. A significant portion of the files contained RQCR circuits, but only 2610 files (33.11% of quantum programs) were directly translated into this type of circuit representation. Anyways, since a program can define more than one circuit, the total number of circuits analyzed were 3719. Additionally, 28.57% of the circuits were accompanied by metrics and in 11.85% of the circuits at least one design pattern was detected.

4.1.6 Analysis method

Various analysis methods are used in the case study in order to attempt answering research questions:

- *Descriptive statistics*: have been used to analyze the data and answer the three research questions (RQ1 to RQ3) regarding the use of design patterns in quantum software. Key statistical measures such as minimum, maximum, mean, and standard deviation were calculated to summarize the data. Additionally, bar charts, line plots, and box plots were utilized to visually represent the distribution and frequency of design patterns across different programs. This approach provides a clear overview of the trends and variations in pattern usage, offering insights into the current state of the art in quantum software design.
- *Correlation analysis*: is used to identify and evaluate the relationship between code metrics and design patterns detected in quantum software (i.e., RQ2). By calculating correlation coefficients, such as Pearson's coefficient, this method quantifies the strength and direction of associations between variables. The correlation values range from -1 to 1, where -1 indicates a strong negative relationship, 1 indicates a strong positive relationship, and 0 implies no correlation. This analysis helps to determine whether certain metrics significantly influence the presence of specific design patterns, offering insights into how these patterns impact software understandability and efficiency.
- *Binary Logistic Regression* (BLR): has been used to answer RQ2. BLR is a statistical method used to model the relationship between one or more independent

variables and a binary outcome. In the context of your study, this technique can be particularly useful where the independent variables are various metrics (e.g., number of gates, density, or depth in quantum circuits), and the dependent variable is whether a specific design pattern was detected (yes/no). This approach allows us to estimate how different metrics impact the likelihood of detecting a particular pattern. Binary logistic regression is well-suited for this study because it directly models the probability of detecting a design pattern (binary outcome) based on various metrics (independent variables). BLR allows for a clear interpretation of how changes in metrics influence pattern detection without requiring strict assumptions about the distribution of the metrics. Logistic regression is ideal for binary results and provides more accurate and interpretable results than alternatives such as linear regression. It handles real-world data flexibly, without the strict assumptions required by other models, making it a reliable choice for analyzing the relationship between quantum metrics and design pattern detection.

- *Clustering*: analysis is employed in RQ2 to explore the occurrence of certain patterns and metrics and, thus, determine specific types of quantum programs. In this study is used the **K-means** clustering, which is an unsupervised machine learning algorithm used to partition a dataset into a predefined number of clusters, where each point belongs to the cluster with the nearest mean. The algorithm iteratively updates cluster centroids by minimizing the within-cluster variance until convergence. To determine the optimal number of clusters, the study uses the *Gap Statistic*: method to determine the optimal number of clusters by comparing the within-cluster variance to a null reference distribution. This approach is more statistically rigorous than alternatives like the *Elbow* method, which can be subjective, and *Silhouette* analysis, which may struggle to differentiate well between clusters of varying density. The *Gap Statistic* strikes a good balance between accuracy and efficiency for this analysis.
- *Bayesian Network* (BN): analysis is a probabilistic graphical model that represents a set of variables and their conditional dependencies through a directed acyclic graph. Each node in the network represents a variable, and each directed edge (arc) represents a probabilistic dependency between variables. The strength of these dependencies is quantified using conditional probabilities that are computed using Bayes' Theorem based on the joint probability distribution, which is factored into conditional probabilities of the nodes given their parents. BN combine principles from graph theory, probability theory, and statistics to model uncertain relationships between variables, making them useful for reasoning under uncertainty, diagnosis, prediction, and decision-making. For RQ3, this method is particularly useful because it helps reveal the extent to which the application of certain design patterns is related.

4.2 Result analysis

This section presents the analysis of the results for each of the research questions (Sects. 4.2.1, 4.2.2 and 4.2.3 respectively), together with the main conclusions drawn.

4.2.1 Design pattern usage

The whole data set considered 3719 programs analyzed (those that comply with the pattern detection tool) (see Table 2). Of these programs, in 934 programs, at least one of the design patterns analyzed was detected. Table 3 shows the absolute numbers of the occurrence of each pattern, as well as the relative occurrence of those total numbers, that is, with respect to the total of the analyzed programs, the programs using at least one pattern.

The most common patterns detected are three, *superposition* (418), *entanglement* (339), and *initialization*. This is not surprising, given that *superposition* and *entanglement* are the basis for quantum computation in order to achieve significant computational advantages over classical software.

The *initialization* pattern was not as common as expected. However, since Qiskit automatically initializes qubits to $|0\rangle$, this pattern probably occurs more frequently than detected, as many initialization steps may be handled implicitly by the framework.

The occurrence of *oracle* patterns was marginal, with only 48 detections (see Table 3), likely due to their association with specialized quantum algorithms, such as Grover's and Deutsch-Jozsa, which are less universally applicable compared to *superposition* and *entanglement*. Furthermore, *oracle* patterns often rely on problem-specific *oracle* functions, making them more complex to implement and less frequent in general experiments. Furthermore, these patterns might be encoded in external functions, preventing their detection in RQCR format. Thus, alternative detection methods may be needed to fully capture and analyze *oracle* patterns.

Figure 4 illustrates a consistent increase in the number of detected quantum design patterns from 2017 to 2024. Notably, the total number of patterns, represented by the black line, shows a steady upward trend, reflecting an expanding use of quantum patterns in recent years. The data reveals not only a broadening adoption of quantum design patterns over time but also hints at the maturation of quantum software, with particular patterns becoming more prevalent as quantum computing techniques advance.

In order to answer RQ1, the analysis confirms that design patterns such as *superposition*, *entanglement*, and *initialization* were indeed detected in a significant number of quantum programs, demonstrating that, these foundational patterns are actively being used in the development of quantum software. This highlights the

Table 3 Summary of design patterns detected

Pattern	Occurrence	Percentage regarding #programs analyzed (5973) (%)	Percentage regarding #programs with patterns (934) (%)
Initialization	218	3.7	23.3
Superposition	418	7.0	44.8
Oracle	40	0.7	4.3
Entanglement	339	5.7	36.3

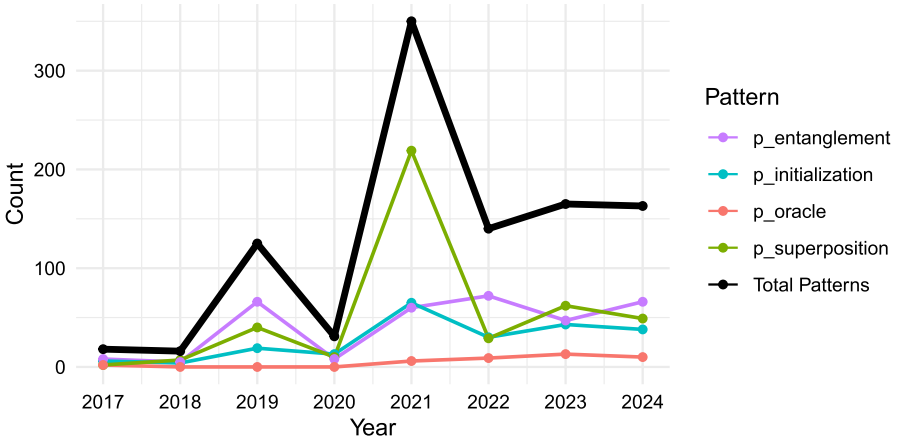


Fig. 4 Evolution of patterns usage in the last 8 years

structured approach to the design of quantum algorithms and the prevalence of key computational strategies. However, the *oracle* pattern was detected less frequently, which may be due to their more specialized or optional use, or limitations in detection tools that may not capture patterns embedded in external functions. This underscores both the current progress in pattern recognition within quantum software and the need for improved detection methods to capture more complex or less conventional patterns.

4.2.2 Relationship between design patterns and code metrics

To answer this question, Table 4 summarizes the values obtained for the code quality measures (minimum and maximum values, as well as average and standard deviation). As highlights, this table shows that the analyzed programs have five gates on average (although there are circuits up to fifty gates), define almost four qubits on average (up to thirty-two for some circuits), and the longest path (depth) is three on average.

Not all measures are likely to influence or be related to the occurrence of design patterns. To analyze this effect, a correlation analysis was performed (as described in Sect. 4.1.6), with the results presented in Fig. 5. For improved clarity, Fig. 5 only includes variables with significant correlations (either positive or negative), specifically those with a coefficient higher than 0.4 or lower than −0.4.

The correlation analysis revealed interesting relationships between the design patterns and various metrics. For the *initialization* pattern, the strongest positive correlations were with the total number of *Pauli* gates ($r = 0.53$) and the number of *Pauli-X* gates ($r = 0.47$). This suggests that circuits utilizing *initialization* tend to involve more *Pauli* operations, particularly *Pauli-X* gates, which may play a role in setting the initial qubit states.

In the case of the *superposition* pattern, the percentage of qubits in initial superposition ($r = 0.73$) and the number of *Hadamard* gates ($r = 0.64$) showed

Table 4 Summary of code metrics in quantum programs analyzed

Description	Metric	Min	Max	Mean	Std. Dev.
Width	m_Width	1	32	3.71	2.6
Depth	m_Depth	1	33	3.15	3.0
Maximum Density	m_MaxDens	1	32	2.25	1.8
Average Density	m_AvgDens	1	14	1.80	1.4
#Pauli-X gates	m_NoP_X	0	16	0.28	0.8
#Pauli-Y gates	m_NoP_Y	0	3	0.03	0.2
#Pauli-Z gates	m_NoP_Z	0	5	0.06	0.3
#Total Pauli gates	m_TNo_P	0	16	0.36	0.9
#Hadamard gates	m_NoH	0	12	1.82	2.2
% Qubits in initial superposition	m_Per_SpposQ	0	1	0.32	0.4
#Other single qubit gates	m_NoOtherSG	0	32	1.21	2.1
#Total Single qubit gates	m_TNoSQG	0	35	3.40	3.0
#Total controlled single qubit gates	m_TNoCSQG	0	8	0.32	0.9
#Total SWAP gates	m_NoSWAP	0	0	0.00	0.0
#Total CNOT gates	m_NoCNOT	0	33	0.97	2.3
% Qubits affected by a CNOT	m_Per_QInCNOT	0	1	0.32	0.4
Average qubits affected by a CNOT	m_AvgCNOT	0	11	0.29	0.7
Maximum CNOTs targeting any qubit	m_MaxCNOT	0	22	0.64	1.2
#Toffoli gates	m_NoToff	0	12	0.16	0.8
% Qubits affected by Toffoli gates	m_Per_QInToff	0	1	0.06	0.2
Average Toffoli gate	m_AvgToff	0	2	0.05	0.2
Maximum Toffoli gates	m_MaxToff	0	5	0.11	0.5
#Total gates	m_NoGates	1	50	4.89	4.1
#Total controlled gates	m_NoCGates	0	33	1.46	2.7
% Single qubit gates in the circuit	m_Per_SGates	0	1	0.72	0.3
#Oracles in the circuit	m_NoOr	0	10	0.03	0.3
#Controlled oracles in the circuit	m_NoCOr	0	1	0.00	0.1
% Qubits affected by oracles	m_Per_QInOr	0	1	0.01	0.1
% Qubits affected by controlled oracles	m_Per_QInCOr	0	1	0.00	0.1
Average depth of an oracle in the circuit	m_AvgOrD	0	1	0.02	0.1
Maximum depth of an oracle in the circuit	m_MaxOrD	0	1	0.02	0.1
#Measurement gates in the circuit	m_NoM	0	32	0.73	1.6
% Qubits measured in the circuit	m_Per_QM	0	1	0.23	0.4

the highest positive correlations. This is expected, as *Hadamard* gates are fundamental in creating *superposition*, and a higher percentage of qubits in *superposition* correlates directly with the use of this pattern. However, a notable negative correlation was observed between *superposition* and the percentage of qubits affected by *CNOT* gates ($r = -0.48$), indicating that circuits focused on *superposition* tend to use fewer entangling operations.

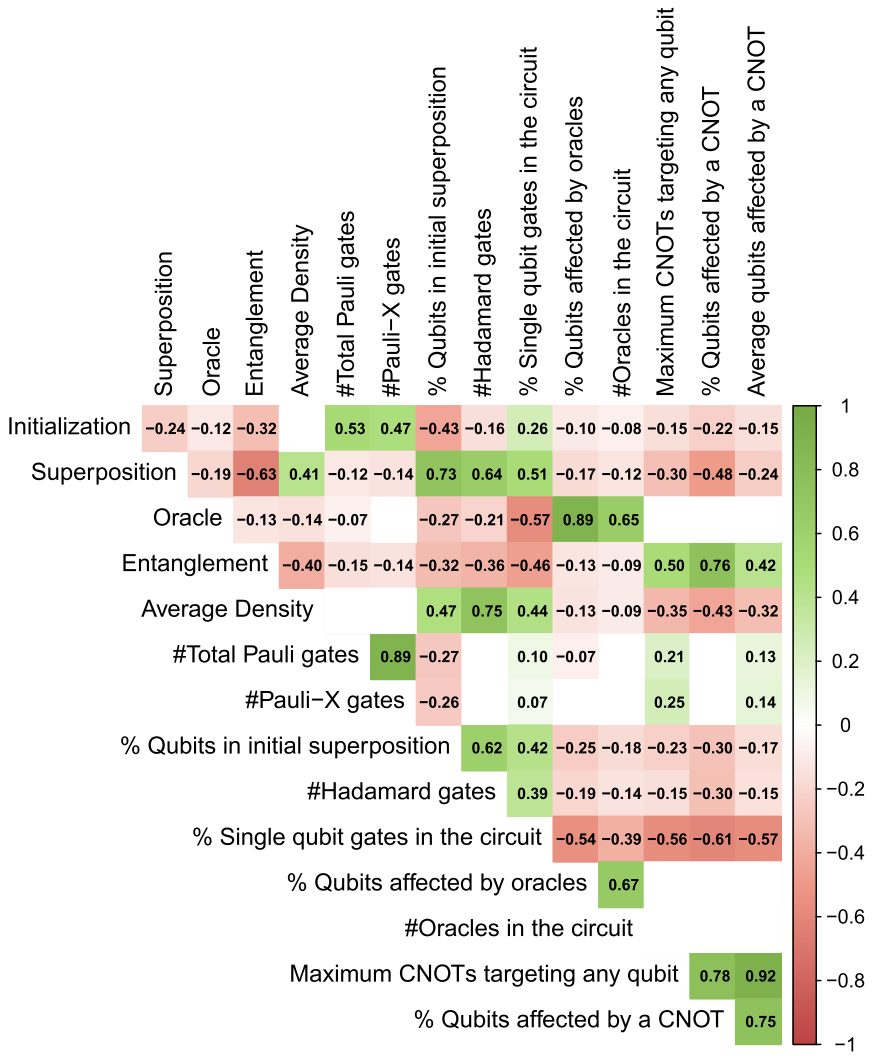


Fig. 5 Correlation of design patterns (*initialization, uniform superposition*, and *oracle*) regarding circuits metrics

The *oracle* pattern showed the highest correlation with the percentage of qubits affected by oracle operations ($r = 0.89$), as well as the number of oracles in the circuit ($r = 0.65$), emphasizing the close relationship between this pattern and the presence of oracle-specific functions. On the other hand, there was a negative correlation with the percentage of single qubit gates ($r = -0.57$), suggesting that circuits using oracles may involve more complex multi-qubit interactions.

Having figured out the most influencing metrics, a Binary Logistic Regression analysis (see Sect. 4.1.6) was conducted for every metrics with high correlation

values. Figure 6 shows the probability curves for all design patterns depending on the values of each metric. In order to note differences in probabilities, it should be noticed that the scale can differ for some plots.

These curves help us to understand how patterns are more likely to be detected when these metrics have certain values. For example, the second plot in the first row in Fig. 6 states that circuits with more than five *X* gates implements the *initialization* pattern with a higher probability. The probability change for the *initialization* patterns occurs abruptly for the number of *X*, and *Pauli* gates in general, while the probability for average of density (another influencing metrics) experiment a gradual rise.

The predicted probability of detecting *superposition* increases sharply with the number of *Hadamard* gates, which is expected because *Hadamard* gates are central to creating superposition states. Furthermore, the percentage of qubits in the initial superposition shows a clear positive relationship with the probability of detecting superposition, reinforcing the role of qubit initialization in this pattern. This indicates that circuits with a higher number of *Hadamard* gates and qubits prepared in superposition are more likely to exhibit the *superposition* pattern.

For the *entanglement* pattern, the probability increases notably as the average number of *CNOT* gates and the percentage of qubits involved in *CNOT* operations rise. This suggests that *entanglement* detection is highly dependent on the presence of *CNOT* gates, which are fundamental in creating entangled pair of qubits. The

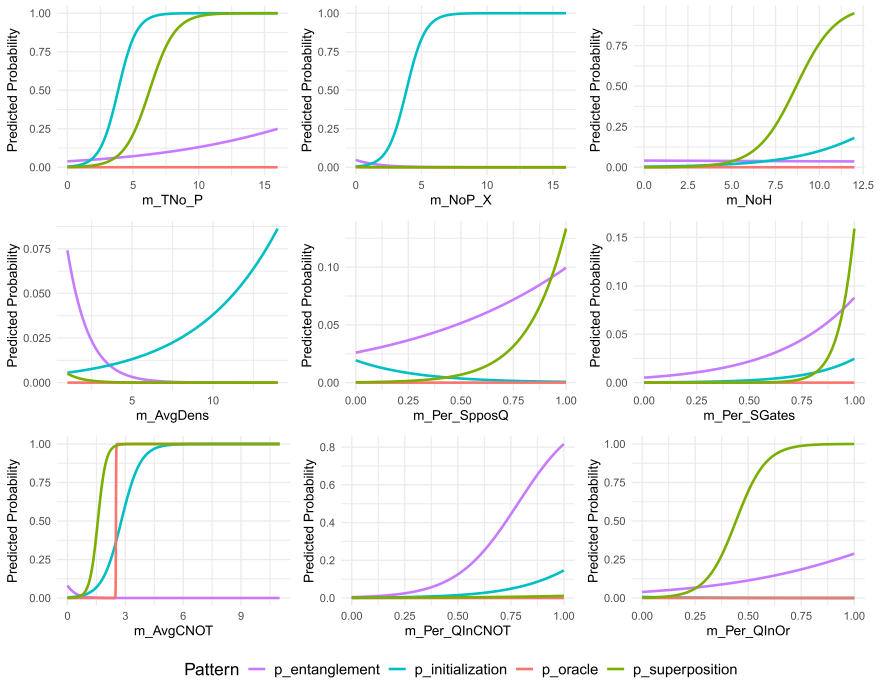


Fig. 6 Predicted probability for patterns (with BLR) for influencing metrics

strong relationship between these metrics and the pattern indicates that circuits with frequent *CNOT* operations are more likely to exhibit *entanglement*.

Finally, regarding the *oracle* pattern, it shows that the predicted probability of detecting the pattern increases as the percentage of qubits involved in oracle operations increases, which confirms the expected results. In addition, the more qubits an oracle operation involves, the higher the likelihood of detecting the *oracle* pattern in the circuit.

Despite observing the correlation between metrics and design patterns, and how the influence of these metrics on pattern detection can be predicted, it is essential to perform the clustering analysis mentioned in Sect. 4.1.6. Clustering will allow us to identify different types of programs based on the combination of patterns they exhibit simultaneously and the specific metrics they present. It should be clarified that the clustering analysis has been made only with the influencing, most correlated, metrics mentioned before.

The optimal number of clusters was four (according to the Gap Statistic method). Each cluster highlights a different combination of design patterns (see Fig. 7). Clustering uncovers specific correlations between patterns and metrics that would otherwise remain hidden, such as the relationship between initialization and average density or superposition and the number of Hadamard gates. This knowledge can support both supervised pattern detection based on code metrics and informed algorithm design decisions to meet specific hardware constraints (e.g., number of qubits or gate sets). Additionally, 8 displays box plots for the key influencing metrics across the clusters. This differentiation reveals how various programs are defined by distinct code metrics, offering a deeper insight into the diversity of quantum software design.

- Cluster 1 primarily shows a high number of *oracle* patterns. Regarding code metrics, it contains a certain levels of qubits in oracles and *CNOT* gates.

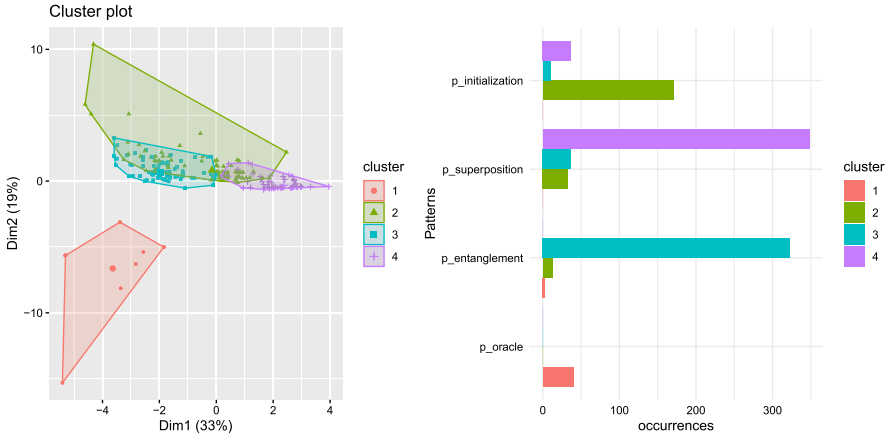


Fig. 7 K-means clustering results (left) and patterns distribution by cluster (right)

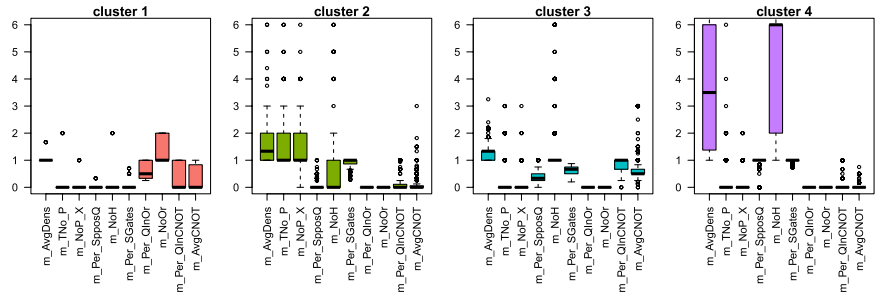


Fig. 8 Box plots for influencing code metrics in every cluster

- Cluster 2 groups programs mainly with *initialization* patterns but also with some *superposition* patterns as well. Relevant code metrics are density average, number of *Pauli* (specially *X*) gates, and *Hadamard* gates, which clearly contribute to the *initialization* pattern.
- Cluster 3 shows a higher concentration of the *entanglement* pattern, although there are some *superposition* and, to a lesser extent, *initialization* pattern. This is a diverse cluster regarding metrics.
- Cluster 4 emphasizes the *superposition* pattern together with some occurrences of the *initialization* pattern. There are two code metrics in this cluster that are much higher than others, the average of density and the number of the *Hadamard* gates.

In order to provide a clear answer to RQ2, the analysis confirms that there is a strong relationship between the usage of design patterns and specific code metrics in quantum software. Although many of these relationships were expected, such as the link between *Hadamard* gates and *superposition* or *CNOT* gates and *entanglement*, the results confirm these associations and highlight that these metrics can be used to accurately predict the occurrence of specific patterns. For instance, metrics like the number of *Pauli-X* gates can reliably forecast the presence of the *initialization* pattern. This predictive power not only validates existing assumptions but also enhances our understanding of how quantum programs are designed, enabling more precise pattern detection based on their operational characteristics.

4.2.3 Relationships in the application of various design patterns

The fact that some patterns appear more frequently than others suggests that certain combinations of patterns are essential in quantum software architecture, while others have more specialized applications. In general, the correlation matrix in Fig. 5 shows that there are no huge correlation between patterns, except for a negative correlation between *superposition* and *entanglement* patterns. However, after the clustering analysis, it is clear that various design patterns can be applied at the same time in some quantum programs. Thus, RQ3 seeks to examine the

likelihood of different design patterns being applied in combination, and whether the presence of one pattern reduces the probability of detecting another.

Figure 9 shows the result of the Bayesian Network (BN) for the four design patterns. By mapping the dependencies between patterns, the BN provides insights into whether specific patterns tend to appear together or if the presence of one pattern decreases the likelihood of detecting another. This approach gives a clear understanding of how design patterns co-exist in quantum software and allows for more precise predictions about pattern combinations.

The BN results (see Fig. 9) shows that there are certain conditional probabilities between some pairs of patterns. However, these probabilities are very low. Even more, the opposite cases, i.e, the apparition of certain patterns preventing other patterns present higher probabilities. Despite neglected probabilities, the net presents the stronger dependency between the *entanglement* and *initialization*, which in somehow is expected. As said before, default initialization mechanisms in Qiskit is a bias in this case study.

The results of the BN analysis (see Fig. 9) indicate that there are conditional probabilities between certain pairs of patterns. However, these probabilities are generally low. Interestingly, the opposite cases (where the appearance of one pattern decreases the likelihood of another) show higher probabilities. Despite the low probabilities, the network highlights the strongest dependency between *entanglement* and *initialization*, which is somewhat expected. As previously mentioned, the default initialization mechanisms in Qiskit introduce a bias in this case study, influencing these results.

Consequently, these results reveals some dependencies between design patterns, but these are very weak. Therefore, the answer to RQ3 must conclude that there is insufficient evidence to confidently assert any conditional relationships or dependencies in the application of design patterns. The weak dependencies observed do not provide strong support for the idea that the presence of one pattern significantly influences the occurrence of another.

To facilitate future research, all data, including the results of the design pattern detection, are available on Fernández et al [14], and the study repository is on Fernández [13].

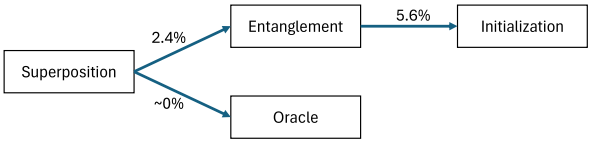


Fig. 9 Bayesian net with conditional probabilities for patterns co-occurrence

4.3 Threats to validity

This section evaluates the validity of the study, analyzes potential threats to validity, and discusses mitigation strategies as well as recommendations for future replications.

4.3.1 Construct validity

This aspect concerns the extent to which the operational measures used in the study accurately represent what the researcher intends to investigate and align with the research questions. In terms of construct validity threats, the metrics used for pattern detection have been binary, indicating whether a pattern was detected or not. However, uncertainty could have been incorporated by assigning a score or probability of pattern occurrence. Future replications of this study could use detection tools that take this factor into account. Regarding the code metrics, while these are relatively new and require further validation, they have been employed in similar previous studies Cruz-Lemus et al [8]. To mitigate this issue, a wide range of metrics was initially considered, and through correlation analysis, the set was refined to focus on the most relevant metrics for subsequent analyses in other research questions.

4.3.2 Internal validity

This aspect of validity concerns external factors that may influence the variables being investigated. Several factors could potentially threaten the integrity of the data.

The data ingestion from GitHub is a challenge, as proprietary quantum programs that are not publicly available might be missing from the analysis. This threat is difficult to mitigate. Additionally, the pattern detection tool, *QCPD Tool*, could introduce bias because it relies on a specific format, and some valid Qiskit programs may be excluded during pre-processing. Despite *QCPD Tool* has been previously validated based on the experts' judgment, the precision and recall of *QCPD Tool* could differ regarding other pattern detection tools. Future replications could address this by using alternative tools to compare results.

The nature of some patterns and features of the programming language (Python/Qiskit) also present threats. For instance, Qiskit's default behavior prevents explicit initialization unless it is to a state other than the default, which could impact the detection of the *initialization* pattern. Similarly, the *oracle* pattern is often dependent on problem-specific oracle functions, making it less common and harder to implement in typical experiments. Furthermore, theoretical definitions of design patterns often specify the use of quantum gates in a particular sequence, but in practice, patterns can be applied in combination, making simultaneous detection more complex.

4.3.3 External validity

This aspect assesses the extent to which the conclusions can be generalized and how applicable they are to other cases with similar characteristics. In this case study, the results can be generalized only to quantum programs written in Qiskit and sourced from GitHub. Although these represent a broad scope (Qiskit is the most widely used language for quantum software and GitHub the largest public code repository) this limits broader generalization. To mitigate this, future studies should expand the scope to include other widely used programming languages, such as OpenQASM, and perhaps explore additional code repositories.

4.3.4 Reliability

This refers to the degree to which the data and analysis may be affected by the specific researchers conducting the study. There are no significant threats, as subjective opinion was not used to determine the value of any metrics. The analysis of the results was conducted with care, limiting the assertiveness of conclusions and providing potential explanations in cases where there was insufficient evidence. This ensures that the findings are based on objective data and that any interpretation is transparently qualified.

4.3.5 Contextual threats

Contextual threats refer to factors within the study's environment or setting that may influence the results. A key contextual threat arises from the rapid evolution of Qiskit. Over the span of the eight years of code analyzed, Qiskit has undergone significant changes, leading to a wide variety of quantum programs with differing coding styles and practices. This variability presents a challenge, as the different versions of Qiskit lead to quantum circuits that are not always comparable and, more importantly, may not be fully analyzable by the pattern detection tool. However, the interpreter modular and systematically handles the parsing of both simple and complex quantum gates, oracles, and other circuit elements across different versions, these make possible to translate into RQCR format, which is essential for applying detection of quantum patterns. Despite this variability, the dataset remains valuable, providing a foundation for comparing current findings with future research, even as quantum computing technologies continue to evolve.

5 Conclusions and future work

The development of quantum software, especially hybrid software that integrates classical and quantum systems, presents numerous challenges, including a lack of abstraction mechanisms and the need for more developed software architectures to support classical-quantum integration. Quantum software design patterns hold promise as solutions to these issues, offering established methods for addressing common problems in quantum software development. However, while some patterns

have been proposed and validated in specific applications, there has been limited analysis of how and when these patterns are applied in practice across a broad range of programs.

This study contributes to filling this gap by providing an empirical analysis of the usage of quantum software design patterns—specifically *initialization, uniform superposition, entanglement, and oracle*—across a large dataset of Qiskit programs. The findings provide practitioners with a clearer understanding of the practical application of these patterns and offer researchers valuable insights that could guide the development of new patterns.

The implications of this study are significant for both researchers and developers. For practitioners, the findings offer practical guidance on how and when to apply specific design patterns, enhancing quantum software quality in terms of performance, maintainability, and scalability. For researchers, insights from the pattern and metric analysis may inform the development of new patterns and tools, supporting advancements in quantum software engineering. This research also supports the broader adoption of quantum computing in industry by promoting knowledge transfer and helping bridge the gap between experimental and scalable quantum solutions.

Future work should address the study's limitations by expanding the code repository analysis to include additional quantum programming languages, such as OpenQASM, and improving pattern detection tools by incorporating scoring mechanisms instead of binary classifications to reduce false negatives. Further analysis of these false negatives would be an essential next step. Additionally, broadening the scope to cover a wider range of design patterns and developing dynamic analysis techniques for capturing circuits as they are constructed, particularly in hybrid environments, could further refine pattern detection.

In addition to future replications of this study that address its current limitations, by characterizing source code with metrics and patterns, researchers could generate valuable datasets for training machine learning models, improving the detection of quantum software design patterns. Comparative studies across various repositories and projects could further illuminate best practices, and machine learning techniques could be leveraged to refactor quantum software, suggesting design patterns that enhance software architecture. These efforts would advance quantum software engineering, facilitating the development of more scalable and efficient quantum applications.

Acknowledgements This work has been supported by projects SMOOTH (PID2022-137944NB-I00), QU-ASAP (PDC2022-133051-I00) funded by MCIU / AEI / 10.13039 / 501100011033 and by the "European Union NextGenerationEU / PRTR", and financial support for the execution of applied research projects, within the framework of the UCLM Own Research Plan, co-financed at 85% by the European Regional Development Fund (ERDF) UNION (2022-GRIN-34110).

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

References

1. Ali S, Yue T (2023) On the need of quantum-oriented paradigm. In: Proceedings of the 2nd international workshop on quantum programming for software engineering, pp 17–20
2. Alsalman A (2023) Accelerating quantum computing readiness: risk management and strategies for sectors. J Quantum Inf Sci 13(2):33–44. https://doi.org/10.4236/jqis.2023.132003
3. Aversano L, Canfora G, Cerulo L, et al (2007) An empirical study on the evolution of design patterns. In: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, pp 385–394
4. Baczyk M, Pérez R, Piattini M (2024) Patterns for quantum software engineering. In: Proceedings of recent advances in quantum computing and technology. association for computing machinery, New York, NY, USA, ReAQCT '24, pp 1-6 https://doi.org/10.1145/3665870.3665871
5. Beisel M, Barzen J, Leymann F, et al (2025) Operations patterns for hybrid quantum applications, to be published
6. Bühler F, Barzen J, Beisel M, et al (2023) Patterns for quantum software development. In: Proceedings of the 15th international conference on pervasive patterns and applications (PATTERNS 2023), pp 30–39
7. Carleton A, Harper E, Robert JE, et al (2021) Architecting the future of software engineering: a national agenda for software engineering research and development. Softw Eng Inst, Pittsburgh, PA, USA, AD1152714
8. Cruz-Lemus JA, Marcelo LA, Piattini M (2021) Towards a set of metrics for quantum circuits understandability. In: International conference on the quality of information and communications technology, Springer, pp 239–249
9. Cruz-Lemus JA, Rodríguez M, Barba-Rojas R et al (2024) Quantum software quality metrics. Springer Nature Switzerland, Cham, pp 125–142. https://doi.org/10.1007/978-3-031-64136-7_6
10. De Stefano M, Pecorelli F, Di Nucci D et al (2022) Software engineering for quantum programming: How far are we? J Syst Softw 190:111326
11. Díaz A, Rodríguez M, Piattini M (2024) Implementing an environment for hybrid software evaluation. Sci Comput Program 236:103109
12. Ezratty O (2021) Understanding quantum technologies 2023. arXiv preprint arXiv:2111.15352
13. Fernández M (2024) Quantumwave. https://github.com/myria75/QuantumWave, accessed: 2024-10-01
14. Fernández M, Pérez R, Cruz-Lemus JA et al (2024). Dataset for exploring design patterns in qiskit programs. https://doi.org/10.5281/zenodo.14017428
15. Fingerhuth M, Babej T, Wittek P (2018) Open source software in quantum computing. PLoS One 13(12):e0208561
16. Furutanpey A, Barzen J, Bechtold M, et al (2023) Architectural vision for quantum computing in the edge-cloud continuum. In: 2023 IEEE international conference on quantum software (QSW), IEEE, pp 88–103
17. Gamma E, Helm R, Johnson R, et al (1995) Elements of reusable object-oriented software. Design Patterns
18. Garcia J, Rojo J, Valencia D et al (2021) Quantum software as a service through a quantum API gateway. IEEE Internet Comput 26(1):34–41
19. Georg D, Barzen J, Beisel M, et al (2023) Execution patterns for quantum applications. In: ICSOFT, pp 258–268
20. IBM Quantum (2024) The history of qiskit: open-source quantum computing framework. https://www.ibm.com/quantum/qiskit/history, accessed: 15-Oct-2024
21. Jiménez L, Pérez R, Piattini M (2024) Reverse engineering of classical-quantum programs. In: ENASE, pp 275–282

22. Jiménez S, Cruz-Lemus JA, Piattini M (2023) A systematic mapping study on quantum circuits design patterns. ICEIS 2:109–116
23. Khan AA, Ahmad A, Waseem M et al (2023) Software architecture for quantum computing systems-a systematic review. J Syst Softw 201:111682
24. Kim Y, Eddins A, Anand S et al (2023) Evidence for the utility of quantum computing before fault tolerance. Nature 618(7965):500–505. https://doi.org/10.1038/s41586-023-06096-3
25. Kshetri N (2024) Monetizing quantum computing. IT Professional 26(01):10–15. https://doi.org/10.1109/MITP.2024.3356111
26. Leymann F (2019) Towards a pattern language for quantum algorithms. In: Quantum technology and optimization problems: first international workshop, QTOP 2019, Munich, Germany, March 18, 2019, Proceedings 1, Springer, pp 218–230
27. Li H, Khomh F, Openja M, et al (2021) Understanding quantum software engineering challenges an empirical study on stack exchange forums and github issues. In: 2021 IEEE international conference on software maintenance and evolution (ICSME), IEEE, pp 343–354
28. Mueck L (2017) Quantum software. Nature 549(7671):171–171
29. Murillo JM, Garcia J, Moguel E, et al (2024) Challenges of quantum software engineering for the next decade: The road ahead. arXiv preprint arXiv:2404.06825
30. Nayak PK, Kher KV, Chandra MB, et al (2023) Q-pac: Automated detection of quantum bug-fix patterns. arXiv preprint arXiv:2311.17705
31. Paltenghi M, Pradel M (2023) Lintq: A static analysis framework for qiskit quantum programs. arXiv preprint arXiv:2310.00718
32. Pérez R, Piattini M (2022) Design of classical-quantum systems with UML. Computing 104(11):2375–2403
33. Pérez R, Serrano MA, Piattini M (2021) Software modernization to embrace quantum technology. Adv Eng Softw 151:102933
34. Pérez R, Fernández M, Cruz-Lemus JA, et al (2024) A preliminary study of the usage of design patterns in quantum software. In: Proceedings of the 5th ACM/IEEE international workshop on quantum software engineering, pp 41–48
35. Peterssen G, Hevia J, Piattini M (2023) Desarrollo profesional de sistemas software híbridos cuántico/clásicos: el ciclo de vida. In: QPath
36. Piattini M, Peterssen G, Pérez R, et al (2020) The Talavera manifesto for quantum software engineering and programming. In: QANSWER, pp 1–5
37. Piattini M, Serrano M, Perez R et al (2021) Toward a quantum software engineering. IT Professional 23(1):62–66
38. PlanQK (2024) PatternAtlas. https://patternatlas.planqk.de/pattern-languages
39. Preskill J (2018) Quantum computing in the NISQ era and beyond. Quantum 2:79
40. Raussendorf R, Browne D, Briegel H (2002) The one-way quantum computer-a non-network model of quantum computation. J Modern Opt 49(8):1299–1306
41. Romero FP, Cruz-Lemus JA, Jiménez S et al (2024) Automata-based quantum circuit design patterns identification: a novel approach and experimental verification. Int J Software Eng Knowl Eng 34(09):1415–1439. https://doi.org/10.1142/S0218194024410031
42. Romero FP, Cruz-Lemus JA, Jiménez-Fernández S et al (2024) Automata-based quantum circuit design patterns identification: a novel approach and experimental verification. Int J Softw Eng Knowl Eng 34(09):1415–1439
43. Runeson P, Host M, Rainer A et al (2012) Case study research in software engineering: guidelines and examples. John Wiley & Sons, Hoboken
44. Sayago J, Pérez R, Piattini M (2021) A systematic mapping study on analysis of code repositories. Informatica 32(3):619–660
45. Serrano MA, Cruz-Lemus JA, Perez R et al (2022) Quantum software components and platforms: overview and quality assessment. ACM Comput Surv 55(8):1–31
46. Swayne M (2023) What are the remaining challenges of quantum computing? The Quantum Insider
47. Ukpabi D, Karjaluoto H, Bötticher A et al (2023) Framework for understanding quantum computing use cases from a multidisciplinary perspective and future research directions. Futures 154:103277. https://doi.org/10.1016/j.futures.2023.103277
48. Vasiliu I (2023) Impact of quantum on the digital economy and society. Coruzant Technologies https://coruzant.com/quantum/impact-of-quantum-on-the-digital-economy-and-society/
49. Weder B, Barzen J, Beisel M, et al (2022) Analysis and rewrite of quantum workflows: Improving the execution of hybrid quantum algorithms. In: Closer, pp 38–50

50. Weigold M, Barzen J, Leymann F, et al (2021) Patterns for hybrid quantum algorithms. In: Symposium and summer school on service-oriented computing, Springer, pp 34–51

Authors and Affiliations

Miriam Fernández-Osuna[1] · Ricardo Pérez-Castillo[1] · José A. Cruz-Lemus[2] · Michal Baczyk[2] · Mario Piattini[2]

✉ Ricardo Pérez-Castillo
ricardo.pdelcastillo@uclm.es

Miriam Fernández-Osuna
miriam.fosuna@uclm.es

José A. Cruz-Lemus
joseantonio.cruz@uclm.es

Michal Baczyk
michal.baczyk@alu.uclm.es

Mario Piattini
mario.piattini@uclm.es

[1] Faculty of Social Sciences and Information Technologies, University of Castilla-La Mancha, Talavera de la Reina, Spain

[2] Escuela Superior de Informática, University of Castilla-La Mancha, Ciudad Real, Spain