I. INTRODUCTION

Nowadays, software evolution is a critical aspect of software engineering today [1], [2], since changes are continuous and they have to be performed very quickly due to the trend of software companies adopting agile methodologies [3] or DevOps [4], [5], or caused by new business and technology market requirements. This scenario implies that the decision-making process should be automated in order to satisfy changing needs and to avoid wrong decisions. This issue is critical since suboptimal architecture design decisions may lead to high cost and poor software quality. Therefore, systematic and (semi-)automated mechanisms that help software architects during the decision-making process are required. Architectural patterns are one of the most important features of software applications, but the same pattern can be implemented in different ways, leaving to results of different quality. When an application requires to evolve, knowledge extracted from similar applications is useful for driving decisions, since quality pattern implementations can be reproduced in similar applications to improve specific quality attributes. Therefore, clustering methods are especially suitable for classifying similar pattern implementations. In this paper, we apply a novel unsupervised clustering technique, based on the well-known artificial neural network model Self-Organizing Maps, to classify Model-View-Controller (MVC) pattern from a quality point of view. Software quality is analyzed by 24 metrics organized into the categories of Count/Size, Maintainability, Duplications, Complexity, and Design Quality. The main goal of this work is twofold: to identify the quality features that establish the similarity of MVC applications without software architect bias, and to classify MVC applications by means of Self-Organizing Maps based on quality metrics. To that end, this work performs an exploratory study by conducting two analyses with a dataset of 87 Java MVC applications characterized by the 24 metrics and two attributes that describe the technology dimension of the application. The stated findings provide a knowledge base that can help in the decision-making process for the architecture of Java MVC applications.

INDEX TERMS Model-view-controller, software architectures, artificial neural networks, self-organizing maps (SOM), unsupervised clustering techniques, machine learning, software quality.
automatically in order to reduce complexity, response-time, and the bias that software architects have during selection. To that end, unsupervised learning techniques are presented as suitable mechanisms to address this problem [9].

Clustering helps knowledge to be simplified, analyzed and reused through the classification of source datasets. In this context, software architecture solutions can constitute datasets to be analyzed, although it is important to highlight that architecture solutions have two dimensions: conceptual and technological [6]. The conceptual dimension is related to the architectural patterns, styles or tactics selected; whereas the technological dimension is related to the specific technologies that have been chosen for the implementation of a particular conceptual solution. In addition, it is also important to note that the same conceptual and technological architecture solution can be implemented in different ways, providing results differing quality. Therefore, despite the fact that there are works that apply clustering [9] to analyze different conceptual solutions, it is necessary to study the existing implementations and technologies [6] for designing a particular conceptual solution. This specific and deeper clustering analysis towards a concrete conceptual solution will provide software architects with more accurate knowledge to support their DD-making process. In this paper, we focus on a well-extended conceptual solution: the Model-View-Controller (MVC) Architectural Pattern [10].

This research work consists of an exploratory study to understand architectural solutions based on the MVC architectural pattern. The main goal of this study is to understand the similarities and differences of Java MVC applications in terms of quality without the software architect bias. To that end, we have used a novel unsupervised clustering method, based on the well-known artificial neural network model known as a Self-Organizing Map (SOM) [11]. From the existing unsupervised clustering techniques, this novel approach has demonstrated important advantages in visualizing, preserving, and analyzing complex knowledge [12], [13]. The exploratory study classifies and studies Java MVC applications from a quality perspective using a dataset of 87 MVC applications characterized with quality metrics that measure: Count/Size, Maintainability, Duplications, Code and Design Complexity [14]–[17], and Quality Design [18]–[20].

The exploratory analysis consists of two studies to classify the dataset using the novel SOM based technique. The first was conducted without any technology information of the application, i.e. using only the 24 quality metrics; the second was performed including 2 features that describe part of the technological dimension: the database used and the MVC implementation, the description of which includes the type of application (web, mobile, or desktop), the existence of a database and the implemented design patterns. In order to determine the candidate values of this last feature, a literature study on the different ways of implementing the MVC pattern was done. Conducting these two studies has allowed us to analyze the degree of influence of the technological dimension on quality metrics. As a result, this paper presents: (i) a literature study about the different types of MVC implementations and the identification of their representative features, (ii) the methodology procedure conducted to perform the exploratory study of classifying and analyzing MVC applications based on clustering, (iii) the adoption of a SOM based clustering method for classifying Java MVC applications, and (iv) the results obtained from the study.

The paper is structured as follows: section II introduces the background about software architectures and clustering methods, section III describes the related work about the adoption of unsupervised clustering methods in software architectures, section IV reports the literature study of the MVC pattern and characterizes the different MVC types that have been identified, section V describes the design of the exploratory study and presents its conduction, section VI summarizes the results and key findings and details the threats to validity, and finally, conclusions are presented in section VII.

II. BACKGROUND

This research work combines software architectures and unsupervised clustering techniques. This section gives the background for both research areas.

A. SOFTWARE ARCHITECTURES

Today, software architecture is defined “as the result of a set of design decisions rather than a set of components and connectors” [7]. Consequently, documenting architectural knowledge [21]—i.e., design decisions (DD), rationale, assumptions, constraints, etc.—is required in order to improve the understanding and rationalization of the architectural decision-making process. Architectural knowledge documentation helps software architects to reason about change, thus maintaining and evolving software architectures and avoiding their erosion, drift, or aging [7], [22]–[24]. Several frameworks have been constructed to store architectural knowledge [25], [26] and to visualize DD [27]. The architectural reasoning led by DD knowledge supports software architects to make better requirement-driven decisions about evolution [28]–[30].

For decades of architectural knowledge research [31], [32], it has been maintained that quality attributes and their metrics are critical during the decision-making process [33]. Software architects play an important role in the decision-making practice, but they are biased in the course of the process [34], [35]. This bias must be reduced as much as possible, since design decision-making is the main driver of software architecture construction [6]. However, most techniques are not completely automated, although many of them have been used for different purposes to support the design decision-making process in a (semi-) automated way [25]:

- Modeling and Visualizing DD mechanisms: graphs [36], free text descriptions [37], DD maps [38], tables or templates.
- Decision Mechanisms: agents [39], case-based reasoning, constrain-based reasoning [40], decision trees and statistical models for managing uncertainty [41], multi-criteria hierarchic optimization problem [42], grounded theory [43],
decisions trees combined with reflective component models [44], or fuzzy inference systems [8], [45].
- Storing Elements: DD, DD dependency networks, traceability between architectural elements and DD, labelling DD, code units, or case-based DD.
- Documentation Mechanisms: free text descriptions, templates, meta-models, ADLs, ontologies [46], or alternative patterns as models [42].

B. CLUSTERING METHODS

Clustering methods are algorithms for grouping the elements of a dataset according to a criterion of distance or similarity. In the context of machine learning, clustering algorithms are included within unsupervised learning techniques, where the model is adjusted by looking for relationships between the descriptive variables of the dataset and identifying the natural structure of the data. The adoption of these methods is especially suitable when there is little information or knowledge about the dataset or human exploration of common hidden properties of the data is required.

One of the main challenges of the clustering process is related to the different shapes, sizes, and densities inherent to data groups. Over the years, a large number of clustering techniques have been proposed to address this problem, although there is no definitive clustering algorithm that works for all cases.

In general, clustering methods are based on hierarchical or partitional techniques. Hierarchical algorithms use heuristic techniques of splitting or merging to generate a dendrogram (cluster tree representing a hierarchy of clusters). The computational complexity of these algorithms and erroneous results in the separation of clusters due to the lack of information on the shape or size of the groups, are two of the most significant drawbacks of hierarchical methods [47]. On the other hand, partitional methods divide the dataset into a specific number of clusters with no relation between them. Its simplicity and low computational complexity make the K-means algorithm one of the most popular partitioning clustering methods [48]. However, the initialization of centroids may affect the quality of clustering [49] and requires that the number of clusters be stated in advance, which may be a problem in certain complex datasets [50].

Another type of partitional models is the density-based clustering algorithms, which identify clusters by searching high-density regions surrounded by low-density areas [51]. Its main drawbacks are its dependence on density drops to determine cluster boundaries, weakness in characterizing the intrinsic structure of the data, and the use of difficult-to-adjust parameters [52]. Among partitional models there are also those non-parametric Bayesian clustering models, where clusters are defined as the objects that most likely belong to a distribution. They have been successfully applied to certain domains [53] but involve some inconvenience, such as the precondition of knowing the probabilistic distribution of the data generation and the need to determine the final number of clusters, even through indirect parameters. In the context of artificial neural networks, the Self-Organized Map (SOM) model can be used as a partitional clustering technique. SOM is a model whose operation is inspired by how the cortical somatosensory area is structured in the brain [11]. In this sense, the SOM fulfills two basic functions: (i) vector quantization, by obtaining a reduced subset of prototype vectors that represent the continuous input space of the n-dimensional dataset; and (ii) vector projection, as the prototype vectors are organized in a regular grid, which generates the projection of the n-dimensional input space in the low-dimensional (often two-dimensional) output space.

SOM provides several advantages over the clustering methods described above. First, SOM facilitates the visualization of high-dimensional data into an ordered 2D-map, which helps the discovery of hidden patterns in the dataset [11]. Second, the topology preservation in SOM ensures that similar elements in the input space are mapped to nearby neurons on the map, and vice versa. Third, SOM-based clustering groups multiple prototypes per cluster, thus adjusting to different shapes of complex cluster structures [54]. As a result, SOM helps in identifying hidden patterns in the dataset and in the analysis and visualization of similarities and differences between groups of data, which are features that other clustering methods lack. Therefore, since the goal of this contribution is to study the similarities and dis-similarities of MVC applications in terms of quality, the SOM model was the clustering method selected for the study.

SOM has been successfully proven in data analysis from very different fields, such as ecology [55], biomedicine [56], engineering [57], financing [58], etc. In this paper, we address a new domain of application of SOM, software engineering, and in particular software architectures. This study has been performed by using a SOM prototype-based cluster analysis methodology that adopts a two-phase procedure [12]. The first phase automatically proposes the best number of units for the SOM by quantifying the data samples from the dataset using topology-preserving metrics, whereas in the second phase, the obtained SOM prototypes are clustered through a connectivity analysis exploring the quality of the partitioning with different number of clusters.

The experimental analysis of this SOM prototype-based cluster methodology demonstrates its ability to identify potential segmentations in a dataset, compared to algorithms that only produce a single clustering solution.

III. RELATED WORK

Software clustering has been traditionally used in reverse engineering processes in software engineering [59], [60]. Specifically, it has been mainly used to identify the software architecture of an application from its source code, when its architecture design is unknown. This clustering technique applies different algorithms (k-means, hierarchical clustering among others) considering classes, methods, packages, and relationships among them in order to determine their modularization into components, packages or files [61]–[65] by taking into account static information, design
rules [66], patterns [67], runtime executions [68], evolution [69] or software architectures together with their architects’ knowledge [70], [71]. The literature on the subject indicates that clustering-based architecture extraction is a well-investigated area, although the accuracy of the results depend on: (i) the selected relationships during the architecture [72] and (ii) the characteristics and number of the projects and applications analyzed, which are usually less than 15 projects [69] or datasets are manually generated instead of using real applications [9].

From this mature related work about software architecture extraction, there is a novel research trend of applying clustering techniques to the extracted architectures. They are applied to extract different types of knowledge for different purposes: simplification, software analysis through classification, pattern detection, business market analysis and investments recommendations, etc., [9].

With regard to evolution, the work of Naim et al. [73] takes a step forward in the area to address the evolution problem by classifying new extracted architecture. The work of Raman and D’Souza [74] addresses the evolution of System-of-Systems (SoS) through the uncertainty associated with their emergent behavior. Finally, there are works that allow for the evolution of microservices architectures from monolithic architectures [75].

With regard to DD, the work of Bhat et al. [76] identifies DD from the issue management systems that manage the software architectures. This work has improved by Bhat et al. [77] supporting the DD-making process in terms of quality criteria and clustering the DD using the k-means algorithm.

With regard to software sustainability, the work of Michanan et al. [78] implements a neural network model to classify energy efficient data structures during the execution of CRUD (Create, Read, Update, and Delete) operations.

Finally, with regard to architectural models or patterns, the work of Khan et al. classifies 10 applications with heterogeneous architectural models with this architectural model criterion [9], whereas the work of Kharchenko et al. [79] formalizes the problem of architectural choice by taken into account multi-criteria, especially quality criteria, although it is not demonstrated how the solution works in practice.

IV. MODEL-VIEW-CONTROLLER CATEGORIZATION

Model-View-Controller (MVC) is one of the most extended architectural patterns due to the advantages it provides through its separation of concerns of applications characterized by their user interface and data management. The MVC pattern was defined by Reenskaug in 1979 [80] as a framework for the Smalltalk platform. This pattern decomposes an application into three main components: model, view, and controller [81]. Specifically, the controller is a connector that orchestrates the model and the view by mapping the user’s interactions with the view (GUI) into actions executed by the model. The model manages the business logic and data persistence, and the view presents this data in different ways depending on the user’s needs. As a result, the MVC pattern separates the persistence and presentation concerns through the components model and view. The common implementation of MVC is illustrated in Fig. 1, in which the controller receives the inputs from the user and orchestrates the required interaction with the model and the view, which at the same time interact between them in two different ways: the view receives change notifications from the model, once it has been previously subscribed to these notifications, or the view explicitly asks for changes to the model.

In order to study the MVC, it is necessary to identify the most common implementations and categorize them in order to do a posteriori studies. This work presents an MVC categorization of Java applications based on the kind of application (desktop, web and mobile), design patterns and code smells that has been identified in the research and grey literature. This study is presented in the following subsections.

A. MVC PATTERN IN DESKTOP APPLICATIONS

Desktop Java MVC applications that implement a Graphical User Interface (GUI) are characterized by using graphical libraries such as Java Swing or AWT. In this kind of applications [82], the users’ requests are events of the Java Swing/AWT elements [83]. As a result, the component View receives the user interactions through the graphical elements generating events that are processed by the Controller [84]. The Controller processes the events calling the methods of the view or the model. The Model implements the functionality of the system and the data management. The model interacts with the controller and the view through interfaces preserving the dependency inversion design principle [85]. In addition, when the model needs to store the data in a relational database, it is necessary to introduce the component Persistence in order to use technologies for data management such as JDBC or Hibernate [86], [87] (see Fig. 2).

Several design patterns have been adopted in the literature to implement these components [88]. In the case of the component View, it is possible to implement the pattern Composite for managing the composition of windows and their buttons. The Controller can implement two design patterns: Strategy and Command. The pattern Strategy is
B. MVC PATTERN IN WEB APPLICATIONS

Web applications can be implemented with several variations from the common pattern (see Fig. 3) [84]. The User Interface Components may vary depending on the development framework and use combinations of them: JSP, HTML, CSS, JQuery, etc. In addition, the Model may also vary by being implemented using JavaBeans or Plain Old Java Objects (POJO), etc. As in the case of desktop applications, the Persistence component may be created to implement a DAO for being DBSM independent. Finally, it is important to note that this kind of application may provide or request API Rest services.

On the other hand, web applications also introduce variations through their design patterns implementation. In particular, the View may implement the Composite and Decorator design patterns. The controller may apply three patterns: Strategy, Command, and Mediator. In the pattern Mediator, the View and the Model only interact through the Controller and there are no direct communications between them. In the case of the Model, the Observer and DAO patterns are those that may be implemented as desktop applications [90].

C. MVC PATTERN IN MOBILE APPLICATIONS

Mobile applications are characterized by the absence of a boundary between the View and the Controller components [89]. The View component constructs the presentation conforms an XML layout that establishes the widgets and containers that the view is composed of. The controller wraps these graphical widgets in Activities or Fragments before being visualized and is in charge of the dynamic visualization. The Model is composed by the same main components of web applications (see Fig. 4). With regard to design patterns, the View may implement the Command design pattern [83], the controller may apply the Strategy [90] or Mediator [91] design pattern, and the model may implement the Observer [92] and DAO [91] patterns.

D. MVC CUSTOMIZATION

The MVC pattern implementation can vary in terms of the classes and design patterns that implement it [81] or the kind of GUI application for desktop, web and mobile applications [82]. In addition to these common variations, programmers
TABLE 1. Smells in MVC components.

| Component | Smell                                      |
|-----------|--------------------------------------------|
| View      | The View includes data and/or calculus of the Controller [95] |
|           | The View includes data and/or calculus of the Model [95] |
|           | Promiscuous controller[96]                  |
|           | Brain controller [96]                      |
| Controller| The Controller includes data and/or calculus of the View [95] |
|           | The Controller includes data and/or calculus of the Model [96] |
|           | God class [97][98]                         |
|           | Feature envy [97][98]                      |
|           | Meddling services [99]                     |
|           | Brain repository [99]                      |
|           | Laborious method [99]                      |
|           | Fat repository [99]                        |
| Model     | The Model includes data and/or calculus of the View[99] |
|           | The Model includes data and/or calculus of the Controller [99] |
|           | Ambiguous DAO [99]                        |
|           | Dependency-Based Smell [100][101]          |
|           | Interface-Based Smell [100][101]           |
|           | Ambiguous Interfaces smells [100][101]     |

may introduce customized changes that sometimes imply code smells, even incurring in technical debt [93], [94]. This reveals a wide variation that requires study to determine to what extent MVC applications are similar or different.

TABLE 1 illustrates the most common smells and bad practices that can be found in the literature. These smells modify the results of quality metrics such as Maintainability, Duplications, and Code and Design Complexity. However, the degree of influence of these smells on quality is unknown, so the applications that incur them may be considered similar or not during design-decision making.

V. EXPLORATORY STUDY DESIGN
The main goal of this research work is to analyze the MVC pattern from a quality point of view avoiding software architect bias. To that end, an exploratory study was conducted and reported following the guidelines of the literature [102]–[104] and we have used an automated unsupervised clustering method, based on SOM neural networks, to classify Java MVC applications from their quality metrics, which not require a dataset with a uniform distribution of the patterns per cluster and provide important advantages in visualizing, preserving, and analyzing complex knowledge, not being restricted its clustering discrimination areas to spherical shapes. This exploratory study was decomposed into two different studies: the first study does not have technology information and the second study includes information about the technological solution. This section describes the research objective and questions, the data collection procedure, the analysis and validation procedures, and the methodology to execute the study.

A. RESEARCH OBJECTIVE AND QUESTIONS
The main goal of this research work is to analyze the MVC architectural pattern by classifying different implementation from a quality perspective. This goal is decomposed into the following questions:

**RQ1**: What are the quality metrics (Count/Size, Maintainability, Duplications, Design Quality Code and Design Complexity) that determine the classification similarity of Java MVC applications?

**RQ2**: Is the technology information of architectures the main driver to classify similar Java MVC applications executing a quality metric-oriented classifier?

**RQ3**: Does the technology information of architectures influence the quality classification of Java MVC applications (Count/Size, Maintainability, Duplications, Code and Design Complexity)?

The RQ1 is formulated to determine whether quality metrics play a relevant role in the classification process. The RQ2 aims to study if the similarity that software architects commonly apply to classify an application “two applications with the same type of development (web, mobile, desktop), the same database or MVC implementation imply they are similar”, it is true or, on the contrary, it is a bias in the decision making process. Finally, RQ3 is formulated to compare both studies and to analyze their commonalities and differences.

B. DATA COLLECTION
A problem in software engineering is to find applications to apply clustering techniques because software companies do not provide their source code. This problem has been evidence in related works [69]. Datasets are composed of small sets of applications and sometimes, the applications are intentionally programmed for experimentation. In this study, the application selection of the data collection procedure was designed in order to increase the dataset volume and reduce bias. To that end, we increased the dataset by defining an application selection task (see Fig. 5) based on other systematic processes applied in software engineering such as systematic literature reviews [105] or systematic mappings [106]. These systematic procedures have been tailored to search applications instead of scientific papers. It is composed of 3 steps (see Fig. 6).

The first step consists of searching for the applications in a repository where the software code is accessible. To do this, we searched on Github (https://github.com/) using “MVC” as search string. From the obtained applications, we added a new filter by selecting as programming language “Java”. From this first search, we obtained 112 results (see Fig. 6).

The second step consists of three tasks: (i) to define the inclusion and exclusion criteria, (ii) to open the applications and (iii) to evaluate the accomplishment of the inclusion and exclusion criteria. In the first task, we defined the exclusion criterion “not implementing an MVC pattern”. This exclusion criterion is necessary because despite the fact that applications are registered on Github as MVC applications, some of them do not implement this architectural pattern. The second task requires opening all 112 applications, so the required development frameworks must be installed. In this case, we used NetBeans 8.2 and Spring Tool Suite 4 to add the necessary references and dependences and to compile and run the applications. Finally, the exclusion criterion was evaluated. As a result of this evaluation, we obtained 81 appli-
D. Guamán et al.: Classifying MVC Software Applications Using SOM

FIGURE 5. Data collection procedure formalized using the standard SPEM [107].

FIGURE 6. Selection procedure of applications.

The third step consists in adding other applications from real settings whose source code is accessible and implements the inclusion criteria, i.e., to implement the MVC pattern. In this work, we added 6 applications that are deployed in laboratories of the Universidad Técnica Particular de Loja obtaining a total of 87 applications (see Fig. 6).

Once the applications are selected, the features that will be used to characterize them must be determined. In this work, the main purpose of study is the software quality. To that end, we selected these quality categories: Maintainability, Duplications, Code and Design Complexity [14]–[16], and Quality Design [17]–[19] (see Fig. 5). In addition to these quality categories, the application size must be considered, because a high number of dependencies in a small application it is not the same as in a large one. From these quality categories and size category, we selected as features those quality metrics that can be measured quantitatively in order to be automatically extracted by tools without any kind of bias that a qualitative metric may introduce. From this first selection, we obtained 61 metrics. However, most of them were highly correlated features such as: Lines, Lines of Code, Statements, Comments, Lines of Comments, etc. Since a high number of highly correlated features do not provide additional information and can introduce noise in the analysis of results, the most representative metrics were chosen. Finally, we selected the 24 metrics presented in TABLE 2. To measure the metrics of each of the 87 selected applications, it was necessary to install the tool Sonarqube and the framework Eclipse IDE for Java Developers with the following plugins: Eclipse Metrics, Structural Analysis, JDepend Analysis and Design Pattern Recognition. With the tools and frameworks installed, all 87 applications were executed and the metrics were measured (see Fig. 5). In addition, the architectures of the applications were analyzed in terms of the variations of MVC components (see section IV). From this analysis, features 25 and 26 were obtained (see TABLE 2). Feature 25 describes the used database, and feature 26 was coded using the 13 MVC variations that were found in the selected applications taking into account the analysis performed in section IV, which includes the type of application (desktop, mobile or web), the existence of a database, and the implemented design patterns. These variations are described in TABLE 3.
C. METHODOLOGY

The methodology that we have applied to conduct the study consists of three phases: Feature_Engineering, SOM_Kohonen_Clustering and Clustering_Analysis. They conform to the process that it is formalized in Fig. 7, which was applied to conduct the two studies. The first study followed the process without any technology information of the application, i.e., using only the first 24 quality metrics (see TABLE 2); whereas the second one was executed including the 2 additional features that describe the technological dimension, having 26 features for characterizing the applications.

1) FEATURE ENGINEERING

The phase Feature_Engineering consists of two tasks: Vector_Construction and Data_Z_Normalization (see Fig. 7). This phase departs from the results of the data collection procedure (see section B), i.e., the measurements and values extracted from the 87 applications. In order to train a SOM model, the values that characterize each application must be modeled by means of vectors of numerical components. Therefore, the vector construction consists of representing the collected data in vectors with the same number of components for each application and transforming the alphanumeric values into numerical values.

The dataset used to train the SOM model of the first study consists of 87 vectors (one vector for each application) of 24 components, one component for each one of the metrics. This is possible thanks to the numerical value of the 24 metrics (see TABLE 2). Therefore, in this task, the first study consisted in filling in the 87 vectors with the values of the 24 measured metrics. However, in the case of the second study, the last two features about technology information had to be transformed into numerical values before filling in the vectors (see TABLE 2 and III). Specifically, the transformation of the two features was performed applying the one-hot coding technique, since the values among them are not related. This coding method states that the coded feature must have as many bits as the quantity of values the feature can take. For example, in the case of the feature “Repository”, there exist 10 candidate values (DataBase 4 (for) Objects, HSQLDB, MongoDB, MySQL, H2, Postgres, Google Cloud Datastore (JDO - AppEngine), Cassandra, Apache Derby, and None); therefore, it is encoded with 10 bits, and each value consists of a single bit to 1 and the remaining nine to 0. In this case, the ten values that the feature can take are coded from the first “DataBase 4 (for) Objects” as 1000000000 to the last “None” as 0000000001. Each bit of the coded feature is stored in an individual component of the vector that characterizes the application. As a result, the dataset used to train the SOM model in the second study is composed of 87 vectors with 47 components, which include the required components for coding the values of the two features of the technology information. Specifically, the Repository feature is coded with 10 components, and finally, the MVC feature is coded with 13 components.

The two datasets of 87 vectors with numerical values constitute the input of the second task of feature engineering. This task consists of normalizing data, which is required when the range of values that a feature can have is especially wide or narrow compared with other features. This is especially critical when unsupervised algorithms are applied, and even more in the case of clustering methods based on
## TABLE 2. Features and metrics for quality classification of Java MVC applications.

| No. | Feature/Metric | Formula | Legend | Type | Unit of Measurement | Context |
|-----|----------------|---------|--------|------|---------------------|---------|
| 1   | Lines of Code  | \(\text{LOC} = \sum \frac{L + L - LCM}{L}\) | \(L = \) Non-singly line, \(LCM = \) Comment line | Source code | Number | Application |
| 2   | Statement      | \(\text{Stmt} = \sum \frac{SS + CSS}{CSS}\) | \(SS = \) Simple statement, \(CSS = \) Compound statement | Source code | Number | Application |
| 3   | Functions/Methods | \(\text{Mvm} = \sum M + F\) | \(M = \) Method, \(F = \) Function | Source Code (Method level) | Number | Application, UI/UX, Class Diagram |
| 4   | Classes        | \(\text{Cl} = \sum c\) | \(c = \) Abstract class, concrete class, nested class, interface, enumeration or annotation | Source Code (Class level) | Number | Application, UI/UX, Class Diagram |

### Maintainability

| No. | Code Smells | Formula | Legend | Type | Unit of Measurement | Context |
|-----|-------------|---------|--------|------|---------------------|---------|
| 5   | Smells      | \(\text{Smells} = \sum c\) | \(c = \) Code smell issue | Source code | Number | Application, Design, Documentation |

### Duplications

| No. | Feature/Metric | Formula | Legend | Type | Unit of Measurement | Context |
|-----|----------------|---------|--------|------|---------------------|---------|
| 6   | Duplicates    | \(\text{Duplicates} = \sum b\) | \(b = \) Number of lines involved in duplications | Source code | Number | Application |
| 7   | Duplicated Blocks | \(\text{DuplicatesBlock} = \sum bj\) | \(j = \) Number of duplicated blocks of lines | Source code | Number | Application |
| 8   | Duplicated Files | \(\text{DuplicatesFiles} = \sum f\) | \(f = \) Number of lines involved in duplications | Source code | Number | Application |

### Code Complexity

| No. | Feature/Metric | Formula | Legend | Type | Unit of Measurement | Context |
|-----|----------------|---------|--------|------|---------------------|---------|
| 9   | Cyclomatic Complexity | \(E = \frac{V(G) - N + P + 1}{N}\) | \(V(G) = \) Number of vertices in a graph, \(N = \) Number of nodes in a graph, \(P = \) Number of predicates nodes (node that contain conditions) | Source Code (Class level) | Number | Application |

### Design Quality Metrics

| No. | Feature/Metric | Formula | Legend | Type | Unit of Measurement | Context |
|-----|----------------|---------|--------|------|---------------------|---------|
| 12  | Effort Coupling (Cc) | \(\text{Cc} = \sum \frac{PDC}{CC}\) | \(PDC = \) Fan-Out of Classes, \(CC = \) Number of classes outside the package that classes within the measured package depend on | Source Code (Class level) | Number | Application, UI/UX, Class Diagram |
| 13  | Effort Coupling (Cc) | \(\text{Cc} = \sum \frac{PIC}{PIC}\) | \(PIC = \) Fan-In of Classes, \(CC = \) Number of classes outside the package that classes within the measured package | Source Code (Class level) | Number | Application, UI/UX, Class Diagram |
| 14  | Instability (I) | \(I = \frac{Ce}{Cc + Ce}\) | \(Ce = \) Effort coupling, \(Cc = \) Effort coupling | Source Code (Class level) | Number | Application, UI/UX, Class Diagram |
| 15  | Abstraction (A) | \(A = \sum \frac{Ac}{Ac + A}\) | \(Ac = \) number of abstract classes, \(A = \) abstractness degree | Source Code (Class level) | Number | Application, UI/UX, Class Diagram |
| 16  | Normalized Distance from Mean Sequence (D) | \(D = \sqrt{A + I - 1}\) | \(A = \) Abstractness degree, \(I = \) Instability | Source Code (Class level) | Number | Application, UI/UX, Class Diagram |
| 17  | Concrete Classes Count | \(\text{ConcreteClass} = \sum \text{Conc}\) | \(Conc = \) Number of concrete classes, \(A = \) Abstractness degree | Source Code (Class level) | Number | Application, UI/UX, Class Diagram |

### Design Complexity

| No. | Feature/Metric | Formula | Legend | Type | Unit of Measurement | Context |
|-----|----------------|---------|--------|------|---------------------|---------|
| 18  | Weighted Method per Class (WMC) | \(\text{WMC} = \sum \frac{Ci}{n}\) | \(Ci = \) Complexity of each Method of the Class | Source Code (Class level) | Number | Application |
| 19  | Depth of Inheritance Tree | \(\text{DIT} = \) Depth of inheritance of the class. In classes involving multiple inheritance, the DIT will be the maximum length from the node to the root of the tree | Source Code (Class level) | Number | Application, UI/UX, Class Diagram |
| 20  | Number of Children | \(\text{NDC} = \) Number of immediate subclasses subordinate to a class in the class hierarchy | Source Code (Class level) | Number | Application, UI/UX, Class Diagram |
| 21  | Coupling Between Objects | \(\text{COG} = \) COG for a class is a count of the number of other classes to which it is coupled | Source Code (Class level), Method level | Number | Application |
| 22  | Response for a class (RFC) | \(\text{RFC} = \) RS = the response set for the class | Source Code (Class level), Method level | Number | Application |

### Technology Dimension

| No. | Feature/Metric | Formula | Legend | Type | Unit of Measurement | Context |
|-----|----------------|---------|--------|------|---------------------|---------|
| 23  | Lack of Cohesion in Methods (LCOM) | \(\text{LCOM} = 1 - \frac{M}{F}\) | \(M = \) Number of methods that use a variable, \(F = \) Number of methods in the class | Source Code (Class level), Method level | Number | Application |
| 24  | Type of application | \(T\) | [Web, Mobile, Desktop, Others] | Categorization | Enumeration | Application |
| 25  | Repository | \(DB\) | [SQLServer, MongoDB, DynamoDB, Hadoop, HSQLDB, Others None] | Categorization | Enumeration | Application |
| 26  | MVC | \(MVC\) | [MVC1-MVC2] | Categorization | Enumeration | Application |
Euclidean distances, such as SOM. This means that those features with wider range of values will have more weight than the rest in the distance calculation and, therefore, will distort the result of the clustering. To avoid the resulting imbalance due to the variety of feature ranges, data normalization is required. Normalization of the data was necessary in these two studies, since features such as the lines of code and the number of statements have a wide range of values compared to the rest of features. Specifically, the Z normalization was applied, which consists in scaling the values of each component using its mean value and its standard deviation. Figs. 8 and 9 illustrate this transformation using the first study dataset of 24 components.

2) SOM KOHONEN CLUSTERING

SOM_Kohonen_Clustering phase consists of two tasks: SOM_Size_Selection and OptimalClustering (see Fig. 7). These two tasks conform to the fully automated two-phase process defined by the SOM prototype-based cluster analysis methodology [12]. The first task automatically proposes the optimal number of units in the SOM, whereas in the second task, the obtained SOM prototypes are clustered. This automation is performed by applying the combination of two topology preservation functions and two connectivity indices to guarantee the finest clustering avoiding method and human bias. In addition, this unsupervised two-phase clustering method, based on SOM, is a heuristic process not based on previous probability distributions of the dataset and does not require that groups have a uniform distribution of patterns.

The SOM Size Selection task consists of creating 20 SOM networks with neurons organized in a square neighborhood grid (equal number of rows and columns) for each of the sizes in the range \([r_{min}..r_{max}]\), where \(r\) indicates both the number of rows and columns, and train them using the same dataset and configuration of training parameters, to later select the network with the best results in terms of topology preservation, i.e. similar input vectors are mapped by nearby neurons in the map, and neighboring neurons represent similar input data. The combination of the Kaski–Lagus function \(\Phi_{AC}(0)\) and the topographic function \(\Phi_{PA}(0)\) have been used as the topology preservation metric to weight the topology violations and measure the adaptation of the SOM to the input space, so that the SOM with the minimum value in both functions will be selected for further analysis. The number of vectors in the training dataset is the only factor required to obtain the range of sizes \([r_{min}..r_{max}]\) to analyze as well as the maximum number of clusters to evaluate later [12].

Since both studies depart from the same number of vectors in their datasets, the range of sizes calculated for both cases is \([4..9]\), i.e., 20 SOM networks were trained from the size \(4 \times 4\) to \(9 \times 9\), resulting in a total of 120 SOM networks trained for each study (see Fig. 10). Fig. 10 also shows the selected SOM network for each group of 20 SOM networks trained for a size \(r \times r\). Specifically, the one with the lowest value of the Kaski–Lagus function \(\epsilon_{k_{l}}\) was selected, i.e. the SOM 19 for the size \(4 \times 4\), the SOM 1 form the size \(5 \times 5\), etc. Finally, the process concludes calculating the topographic function \(\Phi_{PA}(0)\) for the selected SOMs in order to identify the SOM network with lower value, i.e., the best one. In addition, the maximum number of clusters \(k_{max}\) was calculated obtaining a value of 6, and therefore the range of clusters to determine the best number of clusters in the second task (OptimalClustering) is \([2..6]\).

TABLE 3 presents the data obtained for each study. The upper graph of TABLE 4 shows the value of the topographic function \(\Phi_{PA}(0)\) calculated for these SOMs (for the sizes \(4 \times 4, 5 \times 5, \ldots, 9 \times 9\)), which in both studies identifies the SOM with the \(4 \times 4\) size as the one that best adapts to its training dataset. TABLE 4 also presents the Unified distance map (U-matrix) and the Direct visualization map of the KOH \(4 \times 4\) networks of the two studies. Both maps display the neurons (circles in U-matrix and rectangles in direct visualization map) as they are arranged in the SOM grid. The U-matrix shows hot and cold areas to illustrate short and long distances, respectively, between the prototype vectors of neurons, so that a compact zone of neurons in cold tones usually identifies a data cluster in the input space and warm zones typically reveal separation between clusters. From the U-matrix graphics of TABLE 4 it is possible to identify 2 potential clusters in both KOH \(4 \times 4\) networks (the two cold areas in the left side of U-matrix of the first study and the two cold areas in the right side of U-matrix of the second study), although it is necessary to apply the second task of this phase in order to ratify them. On the other hand, in the direct visualization map, the prototype vector is shown within each neuron in a graphic format similar to that used to visualize the dataset of the first study (see Fig. 9). TABLE 4 presents the direct visualization maps of the KOH \(4 \times 4\) networks for each study. The prototype vectors share nature and dimension with the
The **Optimal Clustering** task consists in determining which of the potential groupings of the prototype vectors of the SOM network obtained in the previous step is the best. Therefore, the 16 prototype vectors of the KOH $4 \times 4$ selected in the previous step in each study were grouped from 2 clusters to $k_{\text{max}}$ clusters, i.e., from 2 to 6, and the partitioning quality was evaluated through a connectivity analysis based on $\text{CONN}_{\text{Index}}$ [54] and DBI (Davies Bouldin Index) [110].

Following the protocol established in [12] to carry out this task, new 20 SOM networks were trained using a structure of $c$ neurons organized into a single row (being $c$ a value between 2 and 6). The result of each of these trainings achieves the grouping of the 16 prototype vectors of the KOH $4 \times 4$ network in $c$ clusters, selecting for each group of 20 KOH $4 \times 4$ clustered with a specific value of $c$ the one with the lowest DBI (see Fig. 11). After that, the KOH $4 \times 4$ with the neurons grouped in $c$ clusters that obtains the highest $\text{CONN}_{\text{Index}}$ and the lowest DBI is the one selected to perform the classification of the 87 vectors of the dataset, enhancing the highest number of clusters when there is conflict between both indices in order to address a more detailed classification (see Fig. 11). Next, the results of both studies are presented in detail (see TABLE 5).

Despite the fact that the number of clusters ($c$) to group the 16 prototype vectors of the KOH $4 \times 4$ was established from 2 to 6, the second study only presents the results in the range of clusters [2..5] (TABLE 5). This is because after training 20 SOM networks with 6 neurons arranged in a single row, the 16 KOH $4 \times 4$ prototype vectors were grouped in only 5 clusters in all cases.

In the first study, the $\text{CONN}_{\text{Index}}$ and DBI reveal that the best number of clusters are 3 and 5, respectively, whereas in the second study the $\text{CONN}_{\text{Index}}$ and DBI indicate 2 and 3, respectively. Since the goal is to identify the similarity of the applications with a finer grain of accuracy, we selected the higher number of clusters, i.e., 5 clusters in the first study and 3 clusters in the second. This is due to the fact that when the grouped neurons of the KOH $4 \times 4$ in the first study are analyzed, one can observe that the blue and red clusters in the 3 clusters grouping have been decomposed into two clusters each one in the 5 clusters grouping. Specifically, the blue cluster in the 3 clustering grouping is decomposed into the blue and red clusters in the 5 clusters grouping, and the red
cluster in the 3 clustering grouping is decomposed into the green and the violet clusters in the 5 clusters grouping. Also, it is possible to observe the same behavior in the second study, where the red cluster in the 2-clusters grouping is decomposed into the red and green clusters in the 3-clusters grouping.

It should be noted that the two cold areas on the left side of the U-matrix of the first study (TABLE 4) are confirmed as independent clusters in the 5-clustered KOH $4 \times 4$. However, the two cold areas in the right side of the U-matrix of the second study (TABLE 4) have not finally been ratified as two independent clusters in the 3-clustered KOH $4 \times 4$ (they are only separated for the partitions of 4 and 5 clusters, which gave worse results in the connectivity analysis).

Once the 5-clustered KOH $4 \times 4$ for the first study and the 3-clustered KOH $4 \times 4$ for the second study have been obtained, the classification of the 87 applications was executed for both. This execution was performed using the dataset of 87 applications represented by the 24-component vectors (metrics) of the first study and the 26-component vectors of the second study (adding the technology information features). Specifically, this execution consists of classifying each dataset vector into the cluster of the best matching unit, i.e., the neuron which prototype vector has the smallest Euclidean distance to the dataset vector.

3) ANALYSIS
The Analysis phase consists of studying the results provided by the clustering (see Fig. 7). In this section we present the results of the clustering of each study, which are the source of the analysis. The analysis consists in evaluating the similarity of the applications that belong to the same cluster and the differences among clusters.

The results of the first study are illustrated in Fig. 12 and Fig. 13. Fig. 13 presents the distribution of the 87 applications in the neurons of the resulting KOH $4 \times 4$ with 5 clusters. The applications have been distributed as follows: 27 applications in cluster C1, 32 applications in cluster C2, 15 applications in cluster C3, 7 in cluster C4 and 6 in cluster C5. In addition, it also shows the prototype vector of each neuron, i.e., the centroid of the applications that has been mapped by the neuron. It is possible to see the similarity of the prototype vectors that...
TABLE 4. SOM size selection: First and second studies.

| FIRST STUDY | SECOND STUDY |
|-------------|--------------|
| **Topographic function ($\Phi_k^P(0)$)** | **Topographic function ($\Phi_k^P(0)$)** |
| ![Graph](image1.png) | ![Graph](image2.png) |
| **U-matrix (KOH4x4)** | **U-matrix (KOH4x4)** |
| ![Image](image3.png) | ![Image](image4.png) |
| **Direct visualization map (KOH4x4)** | **Direct visualization map (KOH4x4)** |
| ![Image](image5.png) | ![Image](image6.png) |

belongs to the same cluster, e.g., Fig. 12 shows the similarity among the prototype vectors of the neurons that belong to C2, the cluster with a higher number of applications.

The results of the second study are illustrated in Fig. 14 and Fig. 15. Fig. 15 presents the distribution of the 87 application in the neurons of the resulting KOH 4 x 4 with 3 clusters. The applications have been distributed as follows: 72 applications in cluster C1, 10 applications in cluster C2 and 5 in cluster C3. In addition, it also shows the prototype vector of each neuron, i.e., the centroid or prototype vector of the applications that has been mapped by the neuron. In this second study it is also possible to appreciate the similarity of the prototype vectors that belong to the same cluster, e.g., Fig. 14 shows the similarity among the prototype vectors of the neurons that belong to C1, the cluster with a higher number of applications.

VI. RESULTS ANALYSIS

This section reports in detail the results obtained following the last task of the methodology process (see Fig. 7) and the guidelines in the literature [102]–[104]. This last analysis task is presented in the time order in which the studies where executed. In addition, this describes the validation of the study.

From the results of this exploratory study, the analysis to synthesize the information and extract the conclusions was done by answering the research questions defined in the exploratory study design and comparing the different clusters using their max-min column charts. The max-min column chart of a cluster shows the range covered by of the z-normalized values of each feature obtained from the prototype vectors that belong to the cluster. To determine
the similarities and differences among clusters, the range of values of each feature was compared between each pair of clusters. As a result, the set of features in a cluster that have different values (range between minimum and maximum) from the rest of clusters constitute the strict classification criterion that was applied, so the similarity between the applications classified in the same cluster will be those features that make it completely different from the other clusters.
TABLE 5. Optimal clustering: First and second studies.

### FIRST STUDY

| CONN and DBI (Clustered KOH4x4 with c=2, 3, 4, 5 and 6) | CONN and DBI (Clustered KOH4x4 with c=2, 3, 4, 5 and 6) |
|---------------------------------------------------------|---------------------------------------------------------|
| ![Chart](chart1.png)                                   | ![Chart](chart2.png)                                   |

#### Clustered neurons in KOH4x4

- 2 clusters
- 3 clusters
- 4 clusters
- 5 clusters
- 6 clusters

### SECOND STUDY

| CONN and DBI (Clustered KOH4x4 with c=2, 3, 4 and 5) | CONN and DBI (Clustered KOH4x4 with c=2, 3, 4 and 5) |
|-----------------------------------------------------|-----------------------------------------------------|
| ![Chart](chart3.png)                               | ![Chart](chart4.png)                               |

#### Clustered neurons in KOH4x4

- 2 clusters
- 3 clusters
- 4 clusters
- 5 clusters

---

**A. FIRST STUDY: CLASSIFYING MVC APPLICATIONS WITHOUT TECHNOLOGY INFORMATION**

**RQ1:** What are the quality metrics (Count/Size, Maintainability, Duplications, Design Quality, Code and Design Complexity) that determine the classification similarity of Java MVC applications?

The first study aims to determine if it is possible to classify the 87 Java MVC applications by only considering quality metrics, and if it is possible to identify the quality metrics that make this clustering feasible. Fig. 16 presents the obtained max-min column charts from the 5 clusters of the KOH 4 × 4 SOM of the first study, and Fig. 17 shows the max-min column charts grouped by metric. The chart of each cluster has been compared with the rest of charts, looking for those metrics with non-overlapping ranges of values across the 5 clusters (see orange bordered column charts, Fig. 17), in order to identify the main classification criteria on which the KOH 4 × 4 SOM is based on. Next, we present the analysis of each cluster.

1) **CLUSTER C1**

This cluster mapped 27 applications. Fig. 13 shows that cluster C1 does not have a frontier with cluster C4 and, in fact, clusters C1 and C4 do not share common values in
any of the 24 features. Therefore, both clusters characterize applications that are clearly distinguishable from each other.

Cluster C5 has two features similar to cluster C1, Instability and Abstractness, whose value ranges partially overlap. Cluster C3 has five similar features to cluster C1: Density, Duplicated Lines, Duplicated Blocks, Instability and Coupling between Objects. Finally, cluster C2 has the most features in common with cluster C1, because, as it can be seen in the KOH $4 \times 4$ with 3 clusters, these two clusters were merged into a single cluster (see TABLE 5, first study). These features are Code Smell, Density, Duplicated Lines, Duplicated Blocks, Instability and Coupling between Objects, Response for a class (RFC) and Lack of Cohesion in Methods (LCOM). From this analysis is possible
to conclude that cluster C1’s applications are characterized by their size, code complexity, most of the design metrics (Effec-
tent Coupling, Normalized Distance from Main Sequence, Concrete Classes Count and Abstract Class Count), and the

Depth of Inheritance Tree (DIT) feature of design complexity (see Fig. 18).

Specifically, its applications are smaller than other clusters. The size of applications can be classified as XS, if we take
into account the classification of SonarQube for the project size (XS (<1k), S (1k–10k), M (10k–100k), L (100k–500k), XL (> 500k)) [20]. These applications have low levels of complexity (cyclomatic and cognitive), Ce, Ca, Cc, Ac and DIT, with average values of 41.8, 16.9, 0.3, 0.1, 4.2, 0.3 and 0.9, respectively. This fact is evidenced in the z-normalized values of classifying features (see Fig. 18), which all are negative.

2) CLUSTER C2

The C2 cluster mapped 32 applications. Fig. 13 shows that cluster 2 does not have a frontier with clusters C4 and C5, which has been translated into the fact that clusters C2 and C4 only have in common the feature Abstract Class Count and clusters C2 and C5 do not have any. Cluster C3 has eight features similar to Cluster C2: Classes, Efferent Coupling, Instability, Abstractness, Normalized Distance from Main Sequence (D), Abstract Class Count, Number of Children, and LCOM. Finally, cluster C2 has eleven features in common with cluster C1, as it has been described in cluster C1. As a result, the cluster C2’s applications are classified by three of the features of size (Lines of Code, Statements, and Methods), the code complexity, one feature from the design metrics (Concrete Classes Count) and one feature from design complexity (DIT) (see Fig. 19). These applications can also be categorized as XS applications in average, but they are usually bigger than the applications of cluster C1 since the LOC average of cluster C1 is 338.9 and the LOC average of cluster C2 is 651.6. However, taking into account the size increase, cluster C2 applications have proportionally better values of code complexity (cyclomatic and cognitive), Cc, and DIT than cluster C1 applications, with average values of 83.7, 26.1, 13.9, and 0.8, respectively. So, it is possible to conclude that cluster C2 characterizes XS applications with better quality properties than cluster C1. This fact is evidenced in the z-normalized values of classifying features (see Fig. 19), which all are negative, but less negative than C1.
3) CLUSTER C3
There are 15 applications in cluster C3. Fig. 13 shows that cluster C3 has a frontier with all clusters. As has been described before, clusters C1 and C2 have five and eight features similar to cluster C3, respectively. Clusters C4 and C5 have eight features similar to cluster C3. In the case of cluster C4, these features are: Function/Methods, Classes, Instability, Abstractness, Abstract Class Count, DIT, Number of Children, and LCOM. The features of cluster C5 are as follows: Efferent Coupling, Afferent Coupling, Instability, D, Abstract Class Count, DIT, Number of Children, and Coupling between Objects. As a result, applications in cluster C3 are classified by the two size features (Lines of Code and Statements), the Duplicated Files feature of the Duplications category, the code complexity, and two design complexity features (WMC and RFC) (see Fig. 20). These applications are bigger than those in clusters C1 and C2, and their average can be classified as S applications. These applications have higher average values of complexity (cyclo-matic and cognitive), WMC and RFC than clusters C1 and C2, specifically, values of 224.3, 96.8, 6.7 and 7.4, respectively. This fact is evidenced in the maximum z-normalized values of classifying features (see Fig. 20), which are all positive, except in the case of cognitive complexity, by 6 hundredths.

4) CLUSTER C4
The C4 cluster mapped 7 applications. Fig. 13 shows cluster C4 has a frontier with clusters C3 and C5. As it has been described before, cluster C1 has no features in common with cluster C4, and clusters C2 and C3 have one and eight features in common, respectively. In addition, cluster C5 has only two similar features to cluster C4: Efferent Coupling and DIT. From this analysis is possible to conclude the cluster C4’s applications are classified by the two features of size (Lines of Code and Statements), maintenance, duplications, code complexity, the three features of design quality metrics (Afferent Coupling, D and Concrete Classes Count) and design complexity (WMC, Coupling between Objects, and the RFC) (see Fig. 21). Cluster’s C4 applications are in average S applications as cluster C3, but they are usually bigger than the applications of cluster C3, since the LOC average of cluster C3 is 1317.7 and the LOC average of cluster C4 is 2384.7. But if something characterizes applications of cluster C4 is their worst quality, since they increase their values of maintainability, duplications metrics, appearing all their features as classifying values, and also classifying by high values of code and design complexity (see Fig. 20 and Fig. 21). In fact, the average values of cluster C4 applications in maintainability (code smells), duplications (Density, Duplicated Lines, Duplicated Blocks, Duplicated Files), code complexity (cyclo-matic and cognitive) and design complexity (WMC, Coupling between Objects and RFC) are 271, 8.9, 309.3, 17.4, 5.0, 304.8, 151.1, 7.5, 2.8 and 8.4, respectively. So, it is possible to conclude that cluster C4 has S applications with worst quality metrics than cluster C3.

5) CLUSTER C5
There are 6 applications in cluster C5. From the analysis of the other clusters and cluster C5 data, one can conclude that its applications are classified by size, duplications, code complexity, one feature of design quality (Concrete Classes Count) and three features of design complexity (WMC, RFC and LCOM) (see Fig. 22). Cluster C5 applications are mainly characterized by their size M, increases in which are also translated to the rest of the quality features.

6) COMMON CLASSIFICATION OF CLUSTERS
From the analysis of all clusters and the intersection of their classifying features, it is possible to answer RQ1. All the design quality metrics (Count/Size, Maintainability, Duplications, Design Quality, Code and Design Complexity) are valuable to classify Java MVC applications into clusters, but the features that allow the classification of an application into any of the clusters are the size and code complexity metrics. In the case of size, the classifying features are LOC and statements. As a result, it is possible to conclude that the size and code complexity are the main factors determining the similarity of Java MVC applications (see Fig. 23 and Fig. 17-orange bordered column charts-).

RQ2: Is the technology information of architectures the main driver to classify similar Java MVC applications executing a quality metric-oriented classifier?
This first study allows RQ2 to be answered by identifying whether the applications are naturally classified by their technology information without having had this information when applying the clustering method. Therefore, once we have the clustering results it is necessary to determine what kind of applications are classified in each of the 5 clusters...
from an architectural technology information point of view, in order to determine if there is a natural technological classification being unaware of this data. To that end, we have analyzed the two technology dimension features individually and together.

7) REPOSITORY
The results reveal that the database existence is not a classification criterion, since the 5 clusters include applications with and without database. In addition, the specific database systems are not a classification criterion either. Evidence of
this conclusion is the MySQL DBMS, since applications with MySQL are classified in the 5 clusters.

8) MVC
From the MVC feature, it is possible to analyze several characteristics of applications. One of them is the kind of application (web, desktop or mobile). The results reveal that this characteristic is not a classification criterion, since every cluster includes desktop and web applications. With regard to mobile applications, they are part of clusters C2, C4 and C5, which is highly distributed considering that the dataset includes fewer mobile applications than web and desktop ones.

On the other hand, it is possible to analyze the type of MVC. Cluster C1 is characterized to classify desktop and web applications that implements the three components of the MVC: Model, View and Controller, having an exception on the case of Web applications, that also classifies those do not implement the component model, i.e., MVC12 (see TABLE 3). In addition, it also classifies those applications which implement the three components and have a database. In the case of desktop applications, they are characterized by the additional use of libraries or packages; whereas in the case of web applications, it classifies its three components with a database (MVC6), and all the variations with a database in which the functionality is provided as a service, i.e., MVC9 and MVC12. Cluster C2 includes applications of all types of MVC except MVC3 and MVC4. Cluster C3 classifies the same types of MVCs of cluster C1, except for desktop applications, that considers those with database and not with libraries and external packages, and also considers web applications without database and provides functionality as a service (MVC13). Clusters C4 and C5 are the only ones that classify mobile applications of the MVC4, and as cluster C2 they classify applications of MVC8. From these results, it is possible to conclude that the kind of MVC is not a classifying feature.

9) MVC AND REPOSITORIES
From the analysis of the intersection of the specific databases and the MVC, we can conclude that there is not a classifying pattern in the intersection of these two features. MVC7 is a clear example, which is classified by clusters C1, C2 and C3, and all three have applications with MySQL and clusters C2 and C3 have application with database H2. So, the answer to RQ2 in this case is that there is not a natural classification by the technology information of architectures.

B. SECOND STUDY: CLASSIFYING MVC APPLICATIONS WITH TECHNOLOGY INFORMATION

RQ1: What are the quality metrics (Count/Size, Maintainability, Duplications, Code and Design Complexity) that determine the classification similarity of Java MVC applications?

The second study aims to determine if it is possible to classify the 87 Java MVC applications by considering both quality metrics and technology information, and if it is possible to identify the quality metrics that make this clustering feasible. Fig. 24 presents the obtained max-min column charts from the 3 clusters of the KOH 4 × 4 SOM of the second study using 47 features, and Fig. 25 shows the max-min column charts grouped by metric. The chart of each cluster has been compared with the rest of charts, looking for those metrics with non-overlapping ranges of values across the 3 clusters (see orange bordered column charts, Fig.25), in order to identify the main classification criteria on which the HOH 4 × 4 SOM of the second study is based on. Fig. 15 reveals that the three clusters are frontier between them, but it can be seen in Fig. 15 that C1 has more applications than C2 and C3, and C3 has higher values than C1 and C2. Next, we present the analysis of each cluster in detail.

1) CLUSTER 1
C1 is the cluster with the most applications, specifically 72 (see Fig. 15). It has a frontier with clusters C2 and C3 and therefore from the 47 features, it has in common, 30 with C2 and 26 with C3. From this analysis, one can conclude the C1 cluster’s applications are characterized by their maintainability, code complexity, all size features except for the number of classes, 3 features of the design metrics (Efferent Coupling (Ce), Afferent Coupling (Ca) and Concrete Classes Count), 2 features of the Design Complexity (Weighted Method per Class (WMC) and Response for a Class (RFC)) and the MVC4 (see Fig. 26). The size of applications can be classified as XS, since their average of LOC, statements, functions/methods and classes are 685, 227, 77, and 16, respectively. These applications have low levels of code smells, complexity (cyclomatic and cognitive), Ce, Ca, Cc, WMC, and RFC, having their average values 30.8, 97.7, 38.9, 0.6, 0.1, 10.7, 4.7, and 4.1, respectively. This fact is evidenced in the z-normalized values of classifying features (see Fig. 26), in which all minimum values are negatives and the maximum values are close to zero.

On the other hand, it is possible to analyze the type of MVC. Cluster C1 has applications that implement all the MVC patterns except MVC3 and MVC4, but it is the only cluster that does not have applications that implement MVC4, i.e., Java Mobile Model View Controller with the pattern Observer (see TABLE 3).

2) CLUSTER 2
The C2 cluster mapped 10 applications. Fig. 15 shows cluster C2 has frontier with clusters C1 and C3, which has been translated into the fact that it has in common 30 Features with C1 and 21 with C3. The cluster C2’s applications are classified by their maintainability and code complexity, the three features related to the size (Lines of Code, Statements, and Methods/Functions), one feature from the design metrics (Concrete Classes Count), another one from the design complexity (Coupling Between Objects), two databases (DataBase 4 (for) Objects and H2) and one MVC pattern (MVC4) (see Fig. 27). These applications can be
FIGURE 24. Second study: Max-min column charts.
also categorized in average as S application, since the LOC average is 1902 and the average of statements, methods and classes are 815, 154 and 25, respectively. C2’s applications have worse code smell and code complexity values than C1 (cyclomatic and cognitive), having their average values 205.5, 248.2, and 118.8, respectively. This fact is evidenced in the z-normalized values of classifying features (see Fig. 27), which are higher numbers than C1. In addition, it has an average of “Concrete Classes Count” and “Coupling Between Objects” of 39.2 and 2.4.

On the other hand, with regard to DBMS, C2 is characterized by having most applications that use H2 and being the only cluster that has applications with the DBMS DataBase 4 (for) Objects. Finally, it is also characterized by having a minority representation of applications that implement the MVC4, i.e., Java Mobile Model View Controller with the pattern Observer (see TABLE 3).

3) CLUSTER 3
The C3 cluster mapped 5 applications, and despite having the lowest number of classified applications, it has more characterizing features than the other clusters (see Fig. 28). It has a frontier with both clusters C1 and C2, having 26 and 21 features in common, respectively. From this analysis one can conclude that C3 cluster’s applications are characterized...
by their size, maintainability, duplications, code complexity, three design metrics features (Abstractness, Concrete Classes Count, and Abstract Classes Count), one design complexity feature (Lack of Cohesion in Methods (LCOM)) and the MVC4 (see Fig. 28).

The C3 applications are M applications on average (11934 LOC, 5441 Statements, 811 Functions/Methods, and 125 Classes). This increase of size reveals higher numbers of code smells, code complexity and code duplications, being the unique cluster that has duplications as distinctive features. In addition, it also differs from the rest including the design features related to the abstraction and cohesion (see Fig. 28). Finally, the MVC4 is more representative in cluster C3 than in C2 due to C3 has a lower number of applications (see Fig. 27 and Fig. 28).

4) COMMON CLASSIFICATION OF CLUSTERS
From the analysis of all clusters and the intersection of their classifying features, it is possible to answer the RQ1. All the design quality metrics (Count/Size, Maintainability, Duplications, Design Quality, Code and Design Complexity) are valuable to classify Java MVC applications into clusters when the technology information is included as characterization criteria. The quality features that allow the classification of an application into any of the clusters are the three features of size (Lines of Code, Statements, and Methods/Functions), maintainability, code complexity and the quality design feature “concrete classes count metrics” (see Fig. 29 and Fig. 25-orange bordered column charts). As a result, it is possible to conclude that the size, maintainability and code complexity features are the main factors.
determining the similarity of Java MVC applications if we use the 26 features to characterize them (see TABLE 2).

**RQ2: Is the technology information of architectures the main driver to classify similar Java MVC applications executing a quality metric-oriented classifier?**

This second study allows RQ2 to be answered by determining whether the technology information features included in the vectors allow the clustering method classifying the applications to take into account these features and identify those that are not classified by the clusters or only belong to one of them. To that end, we have analyzed the two technology dimension features individually and together.

**5) REPOSITORY**

The results reveal that all of the applications with the database “database 4 (for) objects” are classified in cluster C2 and those with the databases HSQLDB, Google Cloud Datastore (JDO - AppEngine), Cassandra and Apache Derby are classified in the cluster C1. In addition, C3 only classifies SQL databases. However, the database cannot be used as a classification criterion of the Java MVC applications, since a common database as MySql is classified in the 3 clusters and the 3 clusters include applications with database and without it. Finally, non-SQL databases are classified in both clusters C1 and C2.

**6) MVC**

From the MVC feature, it is possible to analyze the kind of application (web, desktop or mobile) and the MVC implementation. The results reveal that this characteristic is not a classification criterion, since every cluster includes desktop, mobile and web applications.

On the other hand, it is possible to analyze the type of MVC. From this study, one can conclude that MVC 3 is only classified in cluster 2 and the rest of them are classified by more than one cluster. As a result, the kind of MVC is not a classifying feature.

**7) MVC AND REPOSITORIES**

From the analysis of the intersection of the specific databases and the MVC, it is possible to identify cases in which the intersection reveals a classification. For example, there are applications using Postgres that are classified in clusters C1 and C3, but they differ in the MVC, since C3 is only used in applications that implement MVC8 and cluster C1 is implemented in applications with MVC7 and MVC12 (see TABLE 3). However, we can conclude that there is not a classifying pattern in the intersection of these two features because there are many combinations of database and MVC that are classified into several clusters, even in the three clusters such as the MVC12 and the MySQL database. So, it is possible to answer RQ2 by concluding that there is not a classification by the technology information of architectures despite having included this information as the vector’s features in the clustering method.

**RQ3: Does the technology information of architectures influence the quality classification of Java MVC applications (Count/Size, Maintainability, Duplications, Code and Design Complexity)?**

In this second study, the technology information has been included into the vectors of the clustering method to validate whether they are critical in determining the similarity of Java MVC applications. After analyzing RQ1 and RQ2 in the present study, it is possible to conclude that the kind of

| Feature                  | Lines of Code | Statement | Functions/Methods | Code Smell | Cyclomatic Complexity | Cognitive Complexity | ConcreteClasses Count | MVC4 |
|--------------------------|---------------|-----------|-------------------|------------|-----------------------|----------------------|-----------------------|------|
| MIN                      | -0.34         | -0.32     | -0.34             | -0.29      | -0.33                 | -0.22                | -0.30                 | -0.15|
| MAX                      | 0.03          | 0.00      | 0.02              | 0.02       | -0.04                 | -0.11                | -0.03                 | -0.15|
| MIN                      | 0.04          | 0.06      | 0.04              | 0.07       | -0.01                 | -0.10                | 0.24                  | 0.10 |
| MAX                      | 0.26          | 0.29      | 0.19              | 0.35       | 0.15                  | -0.04                | 0.45                  | 0.32 |
| MIN                      | 1.24          | 1.26      | 0.84              | 1.35       | 1.11                  | 1.09                 | 0.66                  | 0.47 |
| MAX                      | 2.80          | 2.79      | 2.25              | 2.75       | 2.62                  | 2.36                 | 2.02                  | 0.95 |
application, MVC, and database are not key factors determining the similarity among applications. In order to determine if this technology influences the quality classification, it is necessary to compare the results of both studies.

The influences are evidenced from the clustering results of both studies, since the first study obtained 5 clusters of classification, whereas the second one determined 3 clusters. In addition, the second study provides more features to classify the applications: Lines of Code, Statements, and Methods/Functions), maintainability, code complexity, the quality design feature “concrete classes count metrics” and the MVC 4 (see Fig. 29). Having more features to classify provides more accuracy, especially taking into account that the classifying features of the first study (Lines of Code, Statements, and code complexity) are a subset of the classifying features of the second.

C. DISCUSSION

Having conducted two studies using 87 Java MVC applications characterized by 24 and 26 features, that have been persisted in vectors of 24 and 47 components respectively, have allowed for: (i) a deep study of the similarity of this kind of applications in applying the SOM unsupervised clustering method to avoid bias and (ii) a conclusion determining to what extent quality and technology information is critical in identifying their similarity. The results of both studies can be summarized as findings enumerated as follows.

F. 1 The main factors by which similar XS, S and M Java MVC applications are classified are size, maintainability and code complexity.

F. 2 Information about the kind of application (desktop, web and mobile), MVC implementation and database technology are not classification features that determine the similarity of Java MVC applications.

F. 3 Information about the kind of application (desktop, web and mobile), MVC implementation and database technology is relevant to characterize the application in order to obtain more classifying features and thus be more specific.

F. 4 There is no correlation between the quality of a Java MVC application and its technology information.

F. 5 It is possible to use the clustering results to improve the maintainability and code complexity of applications. From two applications X and Y of the same cluster that use the same DBMS and implement the same MVC, if X has worst quality metrics than Y, Y can be used as reference example for X.

D. THREATS TO VALIDITY

1) INTERNAL VALIDITY

To improve the internal validity of the presented results, the selection of applications was based on evidences described in the code or documentation that the Github repository, or the application itself, in the case of 6 real setting applications, provides. In addition, the metrics characterization criteria were collected using the tools Sonarqube, Eclipse Metrics, Structural Analysis, JDepend Analysis and Design Pattern Recognizer to avoid human intervention. The technology information dimension features were also extracted directly from evidence in the code and documentation of the applications, thus avoiding interpretation. In fact, those features that were not found evidence were categorized as “None” instead of forcing the categorization. Finally, the results of clustering were performed by the SOM tool. As a result, the automatic information management has avoided human bias.

2) CONSTRUCT VALIDITY

Construct validity is concerned with the procedure used to collect data and with obtaining the right measures for the concept being studied. Construct validity was ensured by creating a systematic process for the experimental study as well as for the collection data following the guidelines proposed by [103]–[106]. With regard to the kind of clustering method, we have selected one based on the neural network model Self-Organizing Map (SOM) [11] because it is more rigorous in its construction than common unsupervised clustering techniques, since it has two automated validity steps: the automatic selection of the SOM size, based on topology preserving metrics, and the automated determination of the optimal clustering number, based on connectivity analysis. In addition, this clustering technique avoid the bias of data, since it is a heuristic process not based on previous probability distributions of the dataset, which does not require that groups have a uniform distribution of patterns, and does not restrict the clustering discrimination areas to spherical shapes.

3) EXTERNAL VALIDITY

External validity is concerned to the generality of the results. To address this validity, we conducted two studies instead of only one to analyze the quality and kind of Java MVC applications from different points of view so as to extract more generic conclusions. With regard to the applications, the current study is based on 87 Java MVC applications with a wide variety of features, but they are limited to the sizes XS, S and M, despite the fact that most database types SQL, non-SQL and cloud databases were addressed. Nevertheless, the study is very extensive compared to other studies that have applied unsupervised clustering methods to software engineering [63], [69]. Therefore, the findings can be generalized, at least for Java MVC applications with sizes from XS to M.

VII. CONCLUSION

This paper presents the result of an exploratory study to determine the similarities and differences of Java MVC applications in terms of quality while avoiding the software architect bias. This work takes a step forward in the software architecture analysis by applying a novel SOM unsupervised clustering method. The exploratory study has been carried out using an extensive dataset of 87 applications with 26 descriptive features. In order to analyze them from different perspectives and avoid bias, two different studies were conducted.
and several findings and conclusions have been provided to software architects.

In addition to this main result, this work also provides a classification study of MVC Java applications and the SPEM formalized process for conducting exploratory studies that aim to classify applications using the SOM unsupervised neural network model. This process prescribes three phases: Feature_Engineering, SOM_Kohonen_Clustering and Clustering_Analysis. In addition, this process is extended providing a detailed data collection process to avoid bias, which is also formalized with SPEM.

The findings of this work represent the first stem of a wider branch of future studies. One of these is to increase the training dataset with more applications addressing sizes L and XL. Another is to study the correlation of energy consumption and the maintainability and complexity of these applications. Also, this work provides evidence that the identified features are suitable for the labelling in supervised learning methods. Finally, we will work on determining effective recommendation mechanisms between similar architectural applications that belong to the same cluster in order to improve their maintainability, code complexity, or energy consumption.

REFERENCES

[1] M. M. Lehman, “Laws of software evolution revisited,” in Proc. 5th Eur. Workshop Softw. Process Technol. (EWSPT), vol. 96. London, U.K.: Springer-Verlag, 1996, pp. 108–124.

[2] M. M. Lehman, “On understanding laws, evolution, and conservation in the large-program life cycle,” J. Syst. Softw., vol. 1, no. 3, pp. 213–221, Jan. 1979.

[3] Agile Manifesto. Accessed: 2011. [Online]. Available: http://agilemanifesto.org/

[4] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, “DevOps; IEEE Softw., vol. 33, no. 3, pp. 94–100, Apr. 2016.

[5] A. Dyck, R. Penners, and H. Lichter. “Towards definitions for release engineering and DevOps,” in Proc. IEEE/JACM 3rd Int. Workshop Release Eng., May 2015, p. 3.

[6] M. Soliman, M. Riebisch, and U. Zdun, “Enriching architecture knowledge with technology design decisions,” in Proc. 12th Work. IEEE/IFIP Conf. Softw. Archit., May 2015, pp. 135–144.

[7] J. Bosch, “Software architecture: The next step,” in Software Architecture (Lecture Notes in Computer Science), vol. 3047. Berlin, Germany: Springer, 2004, pp. 194–199.

[8] S. Moaven, J. Habibi, H. Ahmad, and A. Kamandi, “A decision support system for software architecture-style selection,” in Proc. 6th Int. Conf. Softw. Eng. Res., Manage. Appl., Prague, Czech Republic, 2008, pp. 213–220.

[9] Q. Khan, U. Akram, W. H. Butt, and S. Rehman, “Implementation and evaluation of optimized algorithm for software architectures analysis through unsupervised learning (clustering),” in Proc. 17th Int. Conf. Sci. Techn. Autom. Control Comput. Eng. (STA), Sousse, Tunisia, Dec. 2016, pp. 266–276.

[10] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, Software Architecture: Foundations, Theory, and Practice. Hoboken, NJ, USA: Wiley, 2009.

[11] T. Kohonen, Self-Organizing Maps. Berlin, Germany: Springer, 2001.

[12] S. Delgado, C. Higuera, J. Calle-Espinosa, F. Moran, and F. Montero, “A SOM prototype-based cluster analysis methodology,” Expert Syst. Appl., vol. 88, pp. 14–28, Dec. 2017.

[13] S. Delgado, M. Moreno, L. F. Vázquez, J. Á. Martín, and C. Briones, “Morphology clustering software for AFM images, based on particle isolation and artificial neural networks,” IEEE Access, vol. 7, pp. 163034–163032. 2019.

[14] G. A. Campbell, “Cognitive complexity: An overview and evaluation,” in Proc. Int. Conf. Tech. Debt (TechDebt), Gothenburg, Sweden, Feb. 2018, pp. 57–58.

[15] G. A. Campbell, “Cognitive complexity—A new way of measuring understandability,” SonarSource, Geneva, Switzerland, Tech. Rep. Version 1.4, Sep. 2018. [Online]. Available: https://www.sonarsource.com/docs/CognitiveComplexity.pdf

[16] T. J. McCabe, “A complexity measure,” IEEE Trans. Softw. Eng., vol. SE-2, no. 4, pp. 308–320, Dec. 1976.

[17] S. R. Chidamber and C. F. Kemerer, “A metrics suite for object oriented design,” IEEE Trans. Softw. Eng., vol. 20, no. 6, pp. 476–493, Jun. 1994.

[18] R. Martin, “OO design quality metrics," Anal. Dependencies, vol. 12, pp. 151–170, Oct. 1994.

[19] Fall, STAN. Structure Analysis for Java. Version 2. Accessed: May 2020. [Online]. Available: http://stan4j.com/papers/stan-whitepaper.pdf

[20] SonarQube. Metrics Definitions. Accessed: 2016. [Online]. Available: https://docs.sonarqube.org/latest/user-guide/metric-definitions/

[21] M. A. Rabar, D. Tingsöy, P. Lago, and H. V. Vliet, Software Architecture Knowledge Management. Berlin, Germany: Springer-Verlag, 2009.

[22] L. de Silva and D. Balasubramaniam, “Controlling software architecture erosion: A survey,” J. Syst. Softw., vol. 85, no. 1, pp. 132–151, Jan. 2012.

[23] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. A. Stafford, Documenting Software Architectures: Views and Beyond, 2nd ed. Reading, MA, USA: Addison-Wesley, 2010.

[24] A. Jansen and J. Bosch, “Software architecture as a set of architectural design decisions,” in Proc. 5th Work. IEEE/IFIP Conf. Softw. Archit. (WICSA), Nov. 2005, pp. 109–120.

[25] A. Tang, P. Avgieriou, A. Jansen, R. Capilla, and M. A. Babar, “A comparative study of architecture knowledge management tools,” J. Syst. Softw., vol. 83, no. 3, pp. 352–370, Mar. 2010.

[26] A. Tang, M. A. Babar, I. Gorton, and J. Han, “A survey of architecture design rationale,” J. Syst. Softw., vol. 79, no. 12, pp. 1792–1804, Dec. 2006.

[27] L. Lee and P. Kruchten, “Visualizing software architectural design decisions,” in Software Architecture (Lecture Notes in Computer Science), R. Morrison, D. Balasubramaniam, and K. Falkner, Eds. Berlin, Germany: Springer, 2008, pp. 359–362.

[28] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice, 2nd ed. Reading, MA, USA: Addison-Wesley, 2003.

[29] P. Kruchten, P. Lago, and H. van Vliet, “Building up and reasoning about architectural knowledge,” in Proc. 2nd Int. Conf. Qual. Softw. Archit. (QoSA), in Lecture Notes in Computer Science, vol. 4214, C. Hofmeister, I. Mrkovic, and R. Reussner, Eds. Västerås, Sweden, Jun. 2006, pp. 43–58.

[30] H. van Vliet, “Software architecture knowledge management,” in Proc. 19th Austral. Softw. Eng. Conf. (ASWEC), Mar. 2008, pp. 24–31.

[31] R. Capilla, A. Jansen, A. Tang, P. Avgieriou, and M. A. Babar, “10 years of software architecture knowledge management: Practice and future,” J. Syst. Softw., vol. 116, pp. 191–205, Jun. 2016.

[32] D. Falesi, G. Cantone, R. Kazman, and P. Kruchten, “Decision-making techniques for software architecture design: A comparative survey,” ACM Comput. Surveys, vol. 43, no. 4, pp. 1–28, Oct. 2011.

[33] J. Lytra, C. Carrillo, R. Capilla, and U. Zdun, “Quality attributes use in architecture design decision methods: Research and practice,” Comput.-Ing., vol. 102, no. 2, pp. 551–572, Feb. 2020.

[34] A. Manjunath, M. Bhat, K. Shumaia, A. Biessdorf, and F. Matthes, “Decision making and cognitive biases in designing software architectures,” in Proc. IEEE Int. Conf. Softw. Archit. Companion (ICSA-C), Seattle, WA, USA, Apr. 2018, pp. 52–55.

[35] A. Tang, M. Razavian, B. Paech, and T.-M. Hesse, “Human aspects in software architecture decision making: A literature review,” in Proc. IEEE Int. Conf. Softw. Archit. (ICSA), Gothenburg, Sweden, Apr. 2017, pp. 107–116.

[36] L. Lee and P. Kruchten, “A tool to visualize architectural design decisions,” in Proc. 4th Int. Conf. Qual. Softw.-Archit., Models Archit. (QoSA), vol. 8. Karlsruhe, Germany: Springer-Verlag, 2008, pp. 43–54.

[37] R. Capilla, F. Nava, S. Pérez, and J. C. Dueñas, “A Web-based tool for managing architectural design decisions,” in Proc. 1st Workshop Sharing Reusing Architectural Knowledge, 2006, pp. 1–8.

[38] M. Szlenk, A. Zalewski, and S. Kijas, “Modelling architectural decisions under changing requirements,” in Proc. Joint Work. IEEE/IFIP Conf. Softw. Archit. Eur. Conf. Softw. Archit., Helsinki, Finland, Aug. 2012, pp. 211–214.

[39] J.-Y. Chi, L. Sun, X.-G. Chen, and J.-L. Zhang, “Architecture and design of distributed decision support system based on agent grid,” in Proc. Int. Conf. Mach. Learn. Cybern., Guangzhou, China, 2005, pp. 321–327.
O. Maqbool and H. Baby, “Hierarchical clustering for software architecture recovery,” IEEE Trans. Softw. Eng., vol. 33, no. 11, pp. 759–780, Nov. 2007.

B. S. Mitchell and S. Mancoridis, “On the evaluation of the bunch search-based software modularization algorithm,” Soft Comput., vol. 12, no. 1, pp. 77–93, Aug. 2007.

C. Srinivas and C. V. G. Rao, “A feature vector based approach for software component clustering and reuse using K-means,” in Proc. Int. Conf. Eng. (MIS ICEMS), Istanbul, Turkey, 2015, pp. 1–5.

A. Corazza, S. Di Martino, and G. Scanniello, “A probabilistic based approach towards software system clustering,” in Proc. 14th Eur. Conf. Softw. Maintenance Reeng., Mar. 2010, pp. 88–97.

Y. Cai, H. Wang, S. Wong, and L. Wang, “Leveraging design rules to improve software architecture discovery,” in Proc. 9th ACM SigSoft Conf. Qual. Softw. Archit. (QSAC), Vancouver, BC, Canada, 2013, pp. 133–142.

V. Andreetto and S. Becker, “Combining clustering and pattern detection for the reengineering of component-based software systems,” in Proc. Joint ACM SIGSOFT Conf. QoSAC ACM SIGSOFT Symp. ISARCS Qual. Softw. Archit. QoSA Architecting Crit. Syst. ISARCS QoSA-ISARCS, Boulder, CO, USA, 2011, pp. 23–32.

C. Xiao and V. Tzerpos, “Software clustering based on dynamic dependencies,” in Proc. 9th Eur. Conf. Softw. Maintenance Reeng., Washington, DC, USA, 2005, pp. 124–133.

F. Beck and S. Diehl, “On the impact of software evolution on software clustering,” Empirical Softw. Eng., vol. 18, no. 5, pp. 970–1004, Oct. 2013.

A. Christl, R. Koschke, and M.-A. Storey, “Equipping the reflexion method with automated clustering,” in Proc. 12th Work. Conf. Reverse Eng. (WCRE), 2005, pp. 10–20.

G. Bavota, F. Carnevale, A. Lucia, M. Penta, and R. Oliveto, “Putting the developer in-the-loop: An interactive GA for software re-modularization,” in Proc. 4th Int. Symp. Search Based Softw. Eng. (SBSE), 2012, pp. 75–89.

I. Stavropoulou, M. Grigoriou, and K. Kontogiannis, “Case study on which relations to use for clustering-based software architecture recovery,” Empirical Softw. Eng., vol. 22, no. 4, pp. 1717–1762, Aug. 2017.

S. M. Nain, K. Damevski, and M. S. Hossain, “Reconstructing and evolving software architectures using a coordinated clustering framework,” Automated Softw. Eng., vol. 24, no. 3, pp. 543–572, Sep. 2017.

R. Raman and M. Drouza, “Learning framework for maturing architecture design decisions for evolving complex SoS,” in Proc. 15th Ann. Conf. Softw. Syst. Eng. (SoSE), Jun. 2018, pp. 350–357.

S. Eskl and F. Buzhaca, “An automatic extraction approach: Transition to microservices architecture from monolithic application,” in Proc. 19th Conf. Agile Softw. Develop. Companion, Porto, Portugal, May 2018.

M. Bhat, K. Shumaiaev, A. Biesdorf, U. Hohenstein, and F. Matthes, “Automatic extraction of design decisions from issue management systems: A Machine learning based approach,” in Software Architecture ( Lecture Notes in Computer Science), vol. 10475, A. Lopes and R. de Lemos, Eds. Cham, Switzerland: Springer, 2017.

M. Bhat, C. Timmes, K. Shumaiaev, A. Biesdorf, U. Hohenstein, and F. Matthes, “ADx: A tool for automatic curation of design decision knowledge for architectural decision recommendations,” in Proc. IEEE Int. Conf. Softw. Archit. Companion (ICSA-C), Hamburg, Germany, Mar. 2019, pp. 158–161.

J. Michanan, R. Dewri, and M. J. Rutherford, “GreenC5: An adaptive, energy-aware collection for green software development,” in Proc. 11th Int. Sci. Tech. Conf. Comput. Sci. Inf. Technol. (CSIT), Lviv, Ukraine, Sep. 2016, pp. 4–8.

T. M. H. Reenskaug, “The original MVC reports,” Dept. Inform., Univ. Oslo, Oslo, Norway, Tech. Rep., Feb. 2007. [Online]. Available: http://heim.ifi.uio.no/~trygver/2007/MVC_Originals.pdf

S. Burbage, “Applications programming in smalltalk-80 (TM): How to use model-view-controller (MVC),” Smalltalk, vol. 5 pp. 1–11, 1992.

Martin Fowler, GUI Architectures: Model View Controller. Accessed: 2006. [Online]. Available: https://martinfowler.com/eaaDev/ue/Archs.html#ModelViewController

A. Naderlinger and J. Tempel, “A framework for command processing in Java/Swing applications based on the MVC pattern,” in Proc. 6th Int. Symp. Pract. Prac. Program. Java (PPPJ), 2008, pp. 35–42.
[84] A. Syromiatnikov and D. Weyns, “A journey through the land of model-view-design patterns,” in Proc. IEEE/IFIP Conf. Softw. Archit., Apr. 2014, pp. 21–30.

[85] R. C. Martin, “The dependency inversion principle,” C++ Rep., vol. 8, no. 6, pp. 1–66, 1996.

[86] J. H. Sarker and K. Apu, “MVC architecture driven design and implementation of java framework for developing desktop application,” Int. J. Hybrid Inf. Technol., vol. 7, no. 5, pp. 317–322, Sep. 2014.

[87] A. Karagkasidis, “Developing GUI applications: Architectural patterns revisited,” in Proc. Conf. EuroPLoP, 13th Annu. Eur. Conf. Pattern Lang. Program., 2008, pp. 1–31.

[88] E. Gamma, R. Helm, and R. J. Y. J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, London, U.K.: Pearson, 2015.

[89] T. Lou, “A comparison of Android native app architecture–MVC, MVP and MVVM.” Res. Tue.Ni, Eindhoven Univ. Technol., Eindhoven, The Netherlands, Tech. Rep., 2016.

[90] M. Hameed, M. Abrar, A. Siddiq, and T. Javeed, “MVC software design pattern in Web application development,” J. Sci. Eng. Res., vol. 5, no. 5, pp. 17–20, 2005.

[91] K. Sokolova, M. Lemercier, L. Garcia, and L. C. S. Luc, “Towards high quality mobile applications: Android passive MVC architecture,” Int. J. Adv. Softw., vol. 7, no. 1, pp. 123–138, 2014.

[92] D. Plakalović and D. Simic, “Applying MVC and PAC patterns in mobile applications,” J. Comput., vol. 2, no. 1, pp. 65–72, 2010.

[93] E. Tom, A. K. Aurum, and R. Vidgen, “An exploration of technical debt,” J. Syst. Softw., vol. 86, no. 6, pp. 1498–1516, 2013.

[94] L. Peters, “Technical debt: The ultimate antipattern—The biggest costs may be hidden, widespread, and long term,” in Proc. Int. Workshop MTD: Techn. Debt (MTD), 2014, pp. 8–10.

[95] P. Velasco-Elizondo, L. Castañeda-Calvillo, A. García-Fernandez, and S. Vazquez-Reyes, “Towards detecting MVC architectural smells,” in Trends and Applications in Software Engineering, Cham, Switzerland: Springer, 2018, pp. 251–260.

[96] M. Aniche, G. Bavota, C. Treude, A. Van Deursen, and M. A. Gerosa, “A validated set of smells in model-view-controller architectures,” in Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME), Raleigh, NC, USA, Oct. 2016, pp. 233–243, doi: 10.1109/ICSME.2016.12.

[97] M. Fowler, Refactoring: Improving the Design of Existing Code, Reading, MA, USA: Addison-Wesley, 2008.

[98] R. Perez-Castillo and M. Piattini, “Analyzing the harmful effect of god class refactoring on power consumption,” IEEE Softw., vol. 31, no. 3, pp. 48–54, May 2014, doi: 10.1109/MS.2014.23.

[99] A. Fernandez, “Detection strategies of smells in Web software development,” in Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME), Bremen, Germany, Sep. 2015, pp. 598–601, doi: 10.1109/ICSME.2015.7332522.

[100] D. M. Le, C. Carrillo, R. Capilla, and N. Medvidovic, “Relating architectural decay and sustainability of software systems,” in Proc. 13th Work. IEEE/IFIP Conf. Softw. Archit. (WICSA), Venice, Italy, Apr. 2016, pp. 178–181, doi: 10.1109/WICSA.2016.15.

[101] D. Dobraean and L. Diosan, “Model view controller in iOS mobile applications development,” in Proc. 31st Int. Conf. Softw. Eng. Knowl. Eng., Jul. 2019, pp. 547–716.

[102] P. Breeton, B. Kitchenham, D. Badgen, and Z. H. Li, “Using a protocol template for case study planning,” in Proc. 12th Int. Conf. Eval. Assessment Softw. Eng., Jun. 2008, pp. 1–8.

[103] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Weisslén, Experimentation in Software Engineering, Berlin, Germany: Springer-Verlag, 2012.

[104] P. Runeson, M. Höst, A. Rainer, and B. Regnell, Case Study Research in Software Engineering: Guidelines and Examples, Hoboken, NJ, USA: Wiley, 2012.

[105] S. Keefe, “Guidelines for performing systematic literature reviews in software engineering,” Ver. 2.3 EBSE, Seoul, South Korea, Tech. Rep., 2007.

[106] K. Petersen, S. Vakkalanka, and L. Kuzniarz, “Guidelines for conducting systematic mapping studies in software engineering: An update,” Inf. Softw. Technol., vol. 64, pp. 1–18, Aug. 2015.

[107] Object Management Group, Software & Systems Process Engineering Metamodel (SPEM). [Online]. Available: http://www.omg.org/spec/SPEM/

[108] S. Kaski and K. Lagus, “Comparing self-organizing maps,” in Proc. Int. Conf. Artif. Neural Netw., 1996, pp. 809–814.

[109] T. Villmann, R. Der, M. Herrmann, and T. M. Martinetz, “Topology preservation in self-organizing feature maps: Exact definition and measurement,” IEEE Trans. Neural Netw., vol. 8, no. 2, pp. 256–266, Mar. 1997.

[110] D. L. Davies and D. W. Bouldin, “A cluster separation measure,” IEEE Trans. Pattern Anal. Mach. Intell., vol. PAMI-1, no. 2, pp. 224–227, Apr. 1979.