Characterizing the architectural erosion metrics: A systematic mapping study

AHMED BAABAD1,2, HAZURA BINTI ZULZALIL1, SA’ADAH HASSAN1, AND SALMI BINTI BAHAROM1

1 Department of Software Engineering and Information System, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Serdang 43400, Malaysia
2 Department of Management Information Systems, Administrative Sciences, Hadhramout University, Mukalla, Yemen.

Corresponding author: Hazura Binti Zulzalil (hazura@upm.edu.my)

This work was supported in part by the Universiti Putra Malaysia (UPM).

ABSTRACT Software architecture is crucial in determining success or failure in a variety of software development and design fields. Typically, as a system evolves, software architecture deteriorates. This phenomenon is known as architectural erosion. Several studies have addressed architectural erosion based on different solutions. As a result, the metrics technique is the most prevalent solution for architectural erosion. Nevertheless, a comprehensive description of architectural erosion metrics remains unorganized and scattered. This work aims to conduct a systematic mapping to describe and analyze the architectural erosion metrics to provide an overview of erosion metrics and their current trends. Furthermore, no systematic attempts have been made on architectural erosion metrics. The final samples of this study were specified as a total of 43 included papers. Nearly 100 architectural erosion metrics were found. We proposed nine classifications to address architectural erosion challenges, based on adopted approaches in primary studies. The metrics of architectural erosion provide strong evidence for identifying decay and a rapid enabler factor for the adoption of numerous metrics mechanisms to address architectural erosion. The classification of metrics, which is the first of its kind, benefits researchers and practitioners. However, it can be concluded that various aspects are still ambiguous and require further research on architectural erosion measures.

INDEX TERMS Architectural erosion, architecture erosion, architecture degradation, architectural degradation, metrics, measures, mapping study

I. INTRODUCTION

Since late 1989, software architecture has appeared as the initial conception of the large-widely structures of software systems. It plays a prominent role in many aspects of software development: analysis, reuse, understanding, evolution, construction, and management. Practitioners have realized that having the correct architecture is crucial for system development and design [1]. The decisions and principles of the system to be developed are considered part of the software architecture [2].

Software architecture is degraded over time as a variety of factors: code complexity, adding new features, time pressure, fixing bugs, design decisions, accumulating architectural debt, a dependency of the unintended cyclic among components, and inconsistent requirements, as well as technical requirements for changes (i.e., programming languages, hardware, new platforms, and operating system), all of these architectural problems may appear quietly and considerably unobserved and undiscovered unto they grow in the domain and become hard to maintain [3, 4].

The potential risks that refer to problems in architecture evolution and quality of systems [5, 6] must be identified and detected based on an architectural evaluation to address and predict architectural erosion. Numerous solutions have been proposed in [7-9] to combat architectural erosion. Consequently, the metrics strategy is the most common and successful among the available solutions [7, 10]. At various stages of software development, software metrics are used to capture various software characteristics to improve and monitor different products and processes in the software engineering discipline[11]. The basic logic originates from this concept " you cannot control what you cannot measure " [12].
Despite the vast number of measures and their distribution across different topics in software engineering, there is a need to understand the metrics of architectural erosion and analyze those metrics that are the most widely used solutions to determine and comprehend architectural erosion and maintain architectural sustainability due to the lack of any previous systematic study. This study will focus on metrics related to architectural erosion. The study aims to explore the nature of existing metrics by classifying them regarding software architecture erosion. The architectural erosion metrics provide substantial help for a faster and less costly architectural inspection [7, 8]. Since the presence of risks, defects, and problems that arise in the architecture over time with evolution, metrics will adopt this phenomenon to assess the extent of erosion and predict it before it occurs.

Several studies have presented metrics as effective solutions to address architectural erosion [6-9]. Accordingly, these efforts have yielded many results that need more comprehensive reviews and complete details. Furthermore, they are subject to some criteria to obtain a broad knowledge that leads researchers and practitioners to establish ideas and future directions in this field. Nonetheless, as stated previously, the concept of architectural erosion metrics is still disorganized and dispersed. Furthermore, based on what we know, no systematic attempts have been made to classify and describe the existing studies to provide practitioners and researchers with more insightful evidence and a deeper understanding. Therefore, this paper presents a systematic mapping analysis on characterizing architectural erosion metrics to determine the existing state of the art in using erosion metrics and how they are effective in empirical studies.

The key contribution of this paper presents three folds different to the domain: Firstly, we identified 43 included studies to characterize metrics of the architectural erosion, which no other secondary study found out and can be used as an inception indicator to broaden knowledge on the topic. Secondly, we conducted a comprehensive explanation and in-depth comprehension to gain knowledge about i) classification of the architectural erosion approaches and identification of the metrics used in each category, (ii) mapping the metrics related to software quality, (iii) validation of architectural erosion metrics and the extent of its level and case study systems, (iv) used tools supporting for calculation of degradation metrics, (v) comparative analysis among the metrics to identify the extent of the effectiveness of the measures, and (vi) applicability of metrics. Thirdly, we identified the current study trends in architectural erosion metrics to support further investigation and research in this domain.

The following is how the rest of the paper is organized: the background of issues of the relevant architecturally is depicted in section 2. The related work is stated in section 3. The systematic mapping methodology protocol is planned in section 4. The results are shown in section 5. The discussion is explained in section 6. The potential threats to the study's validity are discussed in section 7. The implications of research and practice are demonstrated in section 8. The conclusion is drawn in section 9.

II. BACKGROUND

In order to introduce a correct definition of architectural erosion metrics, this section presents a brief overview concerning software architecture, architectural erosion, and software metrics. The definition is demonstrated in the subsections.

A. SOFTWARE ARCHITECTURE

Software architecture (SA) has been getting attention increasingly since the past decade of the last century. SA plays a pivotal and essential role within the software engineering environment, particularly in software development aspects: understanding, analysis, construction, evolution, reuse, and management; thereby, it's a crucial and high-priority factor to identify success or failure of system development and design [1, 13, 14].

The SA of the system can be described as the system's structures, which include software components, observable properties of components, and the interactions between them. [15]. SA is concerned with the high-level structure and system attributes [16, 17]. It involves and interacts with software families dealing with component-based reuse, limited classes of components, domain-particular design, and software analysis [14]. SA can be represented across two perspectives: prescriptive architecture and descriptive architecture. The descriptive software architecture is concerned with how the system has been designed as-implemented architecture. In contrast, prescriptive software architecture is concerned with design decisions taken prior to system construction as-intended architecture. Hence, its prominence and representation have to be led to the initial understanding of the structure of any software system and analyzing crucial early design decisions [18].

B. ARCHITECTURAL EROSION

Usually, as a system evolves, the software architecture deteriorates. [3, 4]. This problem is not born recently, but it's long-standing software engineering. This phenomenon called architectural erosion [19-22], architectural decay [23], architectural degeneration [21, 24], or architectural degradation [16, 25, 26].

Architectural erosion can be defined as a continuous divergence between prescriptive and descriptive software architecture as intended and implemented. [16, 25]. It occurs when the implemented software architecture, representing the system's actual functions, differs from the planned architecture, representing the system's original design. Several factors may contribute to architectural
erosion, such as the architectural change of a system over time [24], developer mistakes, bad practices [27], disregard of fundamental architectural rules of a system due to the modification. Consequently, continuing architectural erosion could shorten the system lifetime or re-engineering from scratch [27, 28].

Architectural erosion has a significant negative effect on software quality and software architecture. It can result in considerable problems such as increasing software development costs [29, 30], minimizing software performance [31, 32], and declining software quality properties [33] like maintainability, adaptability, or reusability because of an erosion factor contributing to software aging.

B. SOFTWARE METRICS
Measurement is considered, as in all other engineering domains, is pivotal in software engineering to identify, assess, predict, and monitor software entities such as resources, products, and processes for evaluating, controlling, improving, enhancing, and monitoring software quality, productivity, estimation, accuracy, and reliability [11, 34-37]. Inevitably, software measurement is prevalent in every process or product thereby, and it isn’t easy to control something that cannot be measured.

According to the ISO/IEC 15939:2017 [38], the measurement process can be defined as a “primary tool for managing, organizing, performing, and evaluating measure within an overall system, enterprise, or organizational measurement structure. Hence, software metrics are used to appropriately measure different elements of the life cycle of software development [39] since it’s an essential task to a process of software measurement and quality attributes. Goodman [40] defined software metrics as “the continued application of measurement-based techniques to the software development product and process to provide meaningful and timely management information, as well as the use of those techniques to improve and enhance the process and its products.”

A building of product or process for a system must be subject to quality criteria identified by software metrics. It explains that software metrics are the main factor for analyzing the evolution or degradation of the system in general.

III. RELATED WORK
It is significant to present an overview of prior studies on characterizing the metrics based on different forms of software architecture erosion.

Baabd, Zulzalil et al. [7] performed a systematic literature study to thoroughly understand the architectural decay within open-source projects by analyzing the possible reasons, indicators of decay symptoms, proposed solutions, and an extent of solutions effectiveness. According to their research, metrics-based strategies are the most commonly used solutions. Abdellatif, Sultan et al. [41] performed a systematic mapping review (SMR) to provide an overview of component-based software system (CBSS) metrics. In addition, they identified proper metrics to measure required attributes, focusing on elements and approaches utilized to assess the quality of CBSS from the perspective of the component consumer. Staron and Meding [42] surveyed a set of metrics used for information needs that represent architecture metrics, technical debt/risk, and design stability. They found 54 metrics in the literature distributed a generic measurement portfolio regarding the continuous prevalence of the software design properties and the architecture quality. Tahir, Rasool et al. [43] conducted a systematic literature review on current metrics that center on the software architecture early in the design process and over the software’s lifetime for assessing the quality. They provided architecture metrics to identify its relationship with some of the quality attributes and the degree to which these measures represent the quality of architecture. Stevanetic and Zdun [45] conducted a systematic mapping study on the relevant measures to the understandability of architectural structure concepts. These metrics represent the high-level architectural structures (i.e., measures that work above the level classes) concerning their relationship to the system implementation. They classified the metrics in terms of definitions, mapping quality attributes, measured artifacts, level of validation, usability, applicability, comparative analysis, tool support. This is the only study that has a significant similarity to our study in terms of the characteristics of the level of maturity. Still, its fundamental difference lies in that this study classifies measures in terms of understandability of architectural structure concepts, while our study is concerned with characterizing architectural erosion metrics. Mamedou Alenezi [46] surveyed the software architecture quality attributes, particularly stability and understandability of software architecture because several metrics affect stability and understandability. This work paves the way for researchers and practitioners to provide better ways of investigating and measuring the software architecture quality characteristic. Nuñez-Varela, Pérez-Gonzalez et al. [47] performed a systematic mapping study on source code metrics, considered a fundamental component in the software measurement process. They provided a comprehensive overview of an existing state of source code measures and their current trendy track. They also focused on programming paradigms, types of systems, programming languages, and benchmarks systems currently measured by
source code metrics. This study presents proof that source code metrics has a considerable body of research for different computer science areas, empirical analysis studies to be investigated and published for persistent development in source code metrics research. Koziolek [48] conducted a systematic literature review on measuring the sustainability of software architecture. He evaluated sustainability through evolution using scenarios and metrics and early design using scenarios. He carefully investigated the appropriateness of current methods for sustainability analysis and collected a list of more than 40 architecture-level metrics as reported by numerous design principles.

According to the previous reviews, none performed systematic mapping analysis or any other type of review research on characterizing architectural erosion metrics. However, understanding the nature of the existing metrics assessment about software architecture is necessary to address architectural degradation. As a result, this study provides a complete characterization to provide a holistic overview of existing studies in analyzing and accomplishing software metrics at the architectural level based on specific research issues that haven't been addressed in prior published reviews. Table 1 demonstrates a summarized comparison between our study and the previous studies concerning the focus, period-span, number of primary studies, review method, and domain covered in the study.

### Table 1: Review Summary of Prior Studies Literature

| Reference                  | Focus                                                                 | Period-span      | Primary paper | Review method | Domain                              |
|----------------------------|------------------------------------------------------------------------|------------------|---------------|---------------|--------------------------------------|
| Baabad, Zulzali et al [7]  | Understanding the reasons, indicators, symptoms, solutions, and effectiveness of architecture decay. | Up to March 2020 | 74            | SLR           | Architectural decay and OSS          |
| Abdellatief, Sultan et al [41] | Analysis of component-based metrics, focusing on approaches and elements to assess the quality of CBSS. | Up to 2010       | 31            | SMR           | Component-based software system (CBSS) |
| Staron and Meding [42]     | Investigation of metrics used for information needs that represent architecture metrics, technical debt/risk, and design stability. | 2017             | -             | Survey        | Architecture measures               |
| Tahir, Rasool et al. [43]  | Highlight software measurement programs (MPs) to identify current tools, models, strategies and provide information for implementing MPs. | Up to 2016       | 36            | SLR           | Software measurement programs (MPs)  |
| Coulin, Detante et al [44] | Analysis of existing architecture metrics for evaluating the quality | Up to 2019       | 56            | SLR           | Architecture metrics                |
| Stevanetic and Zdun [45]   | To measure the understandability of architectural structure concepts of high-level architecture. | 1990- June 2013  | 25            | SMR           | Understandability of architectural structure |
| Mamdouh Alenezi [46]       | To measure and investigate the software architecture quality attributes in terms of stability and understandability of software architecture. | 2016             | -             | Review        | Software Architecture Quality Measurement Stability and Understandability |
| Nuñez-Varela, Pérez-Gonzalez et al [47] | To collect source code metrics and analyze the current state and their current trends. | 2010- 2015      | 226           | SMR           | Source code metrics                 |
| Koziolek [48]              | To measure the sustainability of a software architecture using scenarios and metrics | Up to 2011       | -             | SLR           | Sustainability of software architecture |

### IV. Systematic Mapping Methodology

A systematic mapping study is a secondary method that provides a comprehensive overview concerning identifying, categorizing, and outlining all available research results of a specific research area [49-51]. Systematic mapping studies focus on an overview of the topic through broad research questions. In contrast, systematic literature studies (SLR) focus on a specific topic through the deep analysis of narrow research questions. Petersen et al.[50] presented a comparison between systematic and mapping studies to describe the goal, process, breadth, and depth to choose the
appropriate study for research. This study aims to provide an overview of a research field by categorizing measures, mapping quality aspects to those categories, and validating metrics through a comprehensive analysis. Therefore, we perform a comprehensive systematic mapping study to characterize the current status of architectural erosion metrics. We follow recent guidelines of systematic maps by Petersen et al. [51]. Fig 1 depicts a flowchart of the mapping process followed by us, confirmed through guidelines and instructions by [50-52].

**FIGURE 1.** The systematic mapping protocol process

A. **PLANNING PROCESS**
The planning process includes sub-processes such as identifying needs, formulating research question(s), and reviewing protocol.

1) **IDENTIFICATION OF NEEDS**
Measures are an essential field in software engineering. There are several classifications of software metrics according to specific measurement needs, such as architectural erosion metrics concerned with detecting, repairing, or predicting problems and defects that affect software architecture. Therefore, many research papers have been published to cover the definitions, concepts, and approaches that reduce the problem of degradation within the software architecture based on the concepts of metrics. Consequently, it is necessary to obtain a comprehensive description of architectural erosion metrics by gathering evidence from current research identified and discussed in this study.

2) **RESEARCH QUESTIONS**
It is necessary to define research questions for the systematic mapping study since it is the basic idea to identify the scope and concepts of knowledge. One of the most critical steps in the mapping studies protocol process is determining and formulating the research question. The research questions and their motivation are shown in Table 2.

3) **REVIEW PROTOCOL**
To perform systematic mapping studies, Petersen et al. [51] and Kitchenham and Charters [52] identified a review protocol method to choose without randomly and individually based on the researcher’s intuition and anticipation, which lead to a bias in research is likely to occur.

The following processes are included in the review protocol: (i) generating research questions, (ii) developing a search plan, (iii) selecting study criteria and procedures, (iv) evaluating quality criteria for collected studies, (v) extracting relevant data, and (vi) synthesizing and analyzing the extracted data. The phases of the review protocol for this study are depicted in Fig. 2.
TABLE 2. RESEARCH QUESTIONS

| ID  | Research question                                                                 | Motivation                                                                                  |
|-----|-----------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|
| RQ 1| What classifications and metrics have been established for assessing architectural erosion? | To find metrics and categories of its approaches based on architectural erosion that can be used to solve architectural challenges. |
| RQ 2| Which metrics have been mapped to each quality attribute?                          | To identify metrics correlated to quality attributes (i.e., metrics evaluate a specific quality whether one or more, such as maintainability, reusability concerning architectural erosion). |
| RQ 3| What are the metrics validation criteria in the context of architectural erosion?   | To identify the criteria of validating erosion measures, it is necessary to recognize the measure to be assessed. |
| RQ 3.1| What validation are approaches used for architectural erosion metrics?             | By describing the proposed metrics, it is essential to ensure how approaches are validated through metrics. |
| RQ 3.2| What validation level and case study are applied to architectural erosion metrics? | It is essential to identify which validation level and systems context were conducted by validating architecture decay metrics. |
| RQ 3.3| How could the extent of metrics effectiveness among several studies be identified based on comparative analysis? | To clarify comparative analysis, if it exists, identify an extent of metrics effectiveness among several studies. |
| RQ 4| Are there tools that support automatically calculating architectural erosion metrics? | Find what tools are supporting metrics. For example, are researchers proposing tools to calculate metrics? or is research dependent on identified metrics without automatic tool support? |
| RQ 5| What is the representative approach for the metric context used?                   | Describe the representation model of the used metric context within the decay phenomenon. |
| RQ 6| To what extent are the metrics applied within the environment of architectural erosion? | Applied metrics can contribute to Identifying the extent of ease and difficulty relating to applicability. |

B. CONDUCTING PROCESS

Once the planning process has been determined, the conducting process begins through sub-processes which include generating search strategy, studies selection relevant to the defined questions to be primary studies at the end, and the quality assessment.

1) SEARCH STRATEGY

Generating a research strategy is crucial since it provides satisfactory results regarding coverage of studies, provided that a strategy generation is accurate and comprehensive. We also performed a manual search strategy to ensure that no articles correlated to the research question were missing, as indicated by [53, 54]. A search mechanism includes search string and searching resources venues. Fig. 2 summarizes the review protocol, including the mechanism of search strategy, which consists of manual and automatic searches.

A: SEARCH STRING

To construct a search string that is relevant to the subject, we should follow the guidelines and instructions given by Petersen et al. [51] and Kitchenham and Charters [52], which involve: a) obtaining main terms by deriving them from the research questions; b) searching for abbreviations, spellings, alternate words, and synonyms for any of the major terms; c) Investigating for the aforementioned prior steps by matching keywords in any relevant research study; d) constructing search strings with the Boolean operators "OR" and "AND". The "OR" operator is used to connect synonyms, alternative words, and abbreviations. While "AND" is used to connect the main terms. e) merging the main terms to construct the ultimate search term.

The researchers specified the question structure based on the experimental design, outcome, intervention, and population to improve the design of the key terms, as declared below:

- Population: Software architecture.
- Intervention: checking of architectural erosion metrics.
- Outcomes: improved reliability of metrics in detection, repair, or prediction of the architecture decay
- Experimental Design: Empirical studies, experimental studies, and case studies.

Once the search string terms have been gathered and made sure regarding some tests that were performed to identify the validity of terms of search string on the chosen libraries, the following comprehensive search terms were selected in this study:

["Architectural drift" OR "Architectural problem" OR "Architectural Smell" OR "Architectural degradation" OR "Architectural erosion" OR "Architectural inconsistency" OR "Architectural decay" OR "Architectural anomaly" OR "Architectural violation" OR "Architectural debt" OR "Architectural change" OR "Architecture drift" OR "Architecture degradation" OR "Architecture erosion" OR "Architecture decay") AND ("measure" OR "metric" OR "measurement" OR "evaluation" OR "quantitative" OR "assessment")].

The prior search terms were approved based on our preliminary reading of a few scientific articles, particularly those recently published, as well as our familiarity with standard search terms.

Except for the web of science and SpringerLink libraries, the search string term was checked in the specific digital libraries by the keywords, abstract, and title per a study, whereas the full text tested the search string term because the advanced search by keywords, title, and abstract is not easy and straightforward to use.
Generating Research Questions

**Search Strategy:**

("Architectural erosion" OR "Architectural degradation" OR "Architectural Smell" OR "Architectural anomaly" OR "Architectural drift" OR "Architectural decay" OR "Architectural inconsistency" OR "Architectural problem" OR "Architectural violation" OR "Architectural change" OR "Architectural debt" OR "Architecture degradation" OR "Architecture decay" OR "Architecture erosion" OR "Architecture drift") AND ("measure" OR "metric" OR "measurement" OR "evaluation" OR "quantitative" OR "assessment")

---

- Wiley: 65
- Scopus: 603
- IEEE Explore: 70
- WOS: 62
- ScienceDirect: 149
- ACM: 514
- SpringerLink: 94

**Required articles after applying inclusion and exclusion criteria**

The scan: 1463 - 1033 = 430

- Screen out Duplication: 430 - 235 = 195
- Title and Abstract scan: 195 - 95 = 100
- Full Text Reading: 100 - 60 = 40
- Final Included Set: 40 + 0 = 40

Search Alert after saving a string strategy within each search engine & applying the above criteria = 0

The primary studies total = 43

- Quality assessment, Full (1), Partial (0.5), No (0):
  - 45 - 2 = 43

Data Extraction

Data Synthesis

**FIGURE 2. Review protocol stages**
B: RESEARCH RESOURCES
Choosing the searching resources venues plays a crucial role in the result determination effectively of the mapping study. Therefore, the researchers must identify the locations of the research resources that are entirely compatible with the topics of their research, whether the resources are specialized towards specific research or the resources that are compatible with all research’s specializations due to the nature of the design of the comprehensive research resource mechanism. In this study, seven databases were used in the search scope. These digital database libraries are the most common and efficient to perform systematic studies in the software engineering context.

The first step is to conduct an automatic search across seven databases, as shown in Table 3. The second step is to conduct a manual search using the backward-forward or snowballing search approach to find related studies among the primary studies that have been chosen. [55]. Finally, the Google Scholar engine was also employed to define related studies citations based on the selected primary studies.

| Name                        | URL                                       |
|-----------------------------|-------------------------------------------|
| IEEE Xplore                 | http://ieeexplore.ieee.org                |
| Springer Link               | http://link.springer.com                 |
| Science Direct              | http://www.sciencedirect.com             |
| Scopus                      | https://www.scopus.com                   |
| ACM Digital Library         | http://dl.acm.org                        |
| Web of Science              | http://www.webofknowledge.com            |
| Wiley Online Library        | https://onlinelibrary.wiley.com/         |

2) STUDIES SELECTION CRITERIA
The selection criteria target all related articles in our systematic mapping study, including exclusion/inclusion criteria and procedure of primary study selection.

A: INCLUSION AND EXCLUSION CRITERIA
The inclusion and exclusion criteria aim to ensure that the selected studies are more related to the defined research questions. In addition, this criterion was determined to find unique papers relevant to the study. Many research papers were discovered during the search process (such as journals, conferences, chapters of books, workshops, symposiums, and other research papers). In this study, the inclusion criteria select the papers published until the end of 2020. Articles in the editorials, research proposals, summaries of tutorials, controversial corners of journals, poster sessions, and panels were excluded. The published articles as short papers were also excluded. The research papers written in English were included, while those not written in English were excluded. In addition, research articles related to architectural decay metrics, whether the answer has a direct or indirect correlation with the research question, were included. Articles that were irrelevant to answering the research questions, on the other hand, were excluded. Table 4 summarizes the papers’ inclusion and exclusion criteria.

B: PROCEDURE OF PRIMARY STUDY SELECTION
The first search process extracted an elementary list of articles, including 1463 (as demonstrated in Fig. 2). Then, the first author performed the primary studies selection, and the other authors investigated the selection process to identify the extent of consistency. In this case, several steps were performed to determine the inclusion of the articles that related to the topic and exclusion the articles that are irrelevant to the topic through following the guidelines by Kitchenham and Charters [52], Petersen et al. [51], and developing search strategies including some recommendations to find out the related studies [55, 56].

Concerning conduct processes, in the first process, the exclusion and inclusion criteria were applied to get the conclusive studies, which refined 430 relevant articles (as shown in Table 4). In the second process, screening duplicated studies was eliminated using the Endnote reference manager, which resulted in 195 articles. The abstract and title were read in the third process to determine whether or not the article was relevant to the defined research question, yielding 100 articles. In the fourth process, reading the full text to assess the article for making the final decision to be included or excluded, refined 40 articles. There might be duplicate articles in terms of content; thereby, the journal article has to be included instead of the conference paper as long as it is up-to-date. In the fifth process, a snowballing search strategy was used to track references and citations of approved studies to ensure no relevant study was missing, which found five articles in the first repetition. In the second repetition, five papers were investigated, and no further studies were found [54]. The search alert was applied for all the search resources presented (as shown in Fig 2) to know the relevant papers published after the first date research, ensuring no further articles were found. In the final process, the quality assessment of the articles was applied to evaluate the content of the article concerning quality, and three studies were excluded because they do not meet the criteria for quality questions evaluation shown in Table 6. Consequently, the total of the included studies was specified in this study, encompassing 43 primary studies after applying the exclusion and inclusion criteria (as illustrated in Fig. 2). Table 5 presents the details of the primary studies of our research.

3) STUDY QUALITY ASSESSMENT
As mentioned in the general criteria for inclusion or exclusion [51, 52], it is essential to reconsider the primary studies regarding study quality assessment. An evaluation of the study quality was applied to investigate whether quality differences illustrate differences in the accuracy of study findings. The checklist method was formulated to evaluate the primary study’s quality based on the defined research questions and extracted data. The purpose of the study quality assessment is to decide whether or not to include a study in order to ensure the quality of the study. The first author conducted the study quality assessment, and the other authors investigated the included studies for quality assessment. Additionally, a dialogue was also held between the authors to discuss and agree on the point of disagreement.

The study quality assessment was partitioned into three levels: High, Medium, and Low. The scores per question were specified in three portions. The first portion, number 1, denotes a quality criteria achievement perfectly and completely. The second portion, number 0, denotes not performing anything from the declared quality criteria in the second portion. The last portion, number 0.5, denotes partial fulfillment of the criteria. Table 6 shows the quality assessment criteria.
TABLE 4. INCLUSION AND EXCLUSION CRITERIA

| No | Inclusion                                                                 | No | Exclusion                                                                 |
|----|----------------------------------------------------------------------------|----|--------------------------------------------------------------------------|
| 1  | Articles provide metrics discussion in the context of architectural erosion issues. | 1  | Articles are written in languages other than English.                    |
| 2  | Articles must have a significant relationship with metrics topics in order to address the research questions. | 2  | Articles with incomplete and insufficient information about how the metrics estimates were derived. |
| 3  | Articles must be written in English.                                        | 3  | Articles in the editorials, research proposals, summaries of tutorials, controversial corners of journals, poster sessions, and panels. |
| 4  | Articles must be published by the end of 2020.                              | 4  | Articles are written in languages other than English.                    |
|    |                                                                            | 5  | Articles that do not cover metrics in the context of architectural erosion issues |
|    |                                                                            | 6  | Articles that have a duplication in several search engines.               |

TABLE 5. DETAILS OF PRIMARY STUDIES

| Ref. No | Author               | Title                                                                 | Year | Publication Source |
|---------|----------------------|----------------------------------------------------------------------|------|--------------------|
| S01     | Aversano et al [57]  | An Empirical Study on the Architecture Instability of Software Projects. | 2019 | Journal            |
| S02     | Lenhard et al. [26]  | Exploring the suitability of source code metrics for indicating architectural inconsistencies | 2019 | Journal            |
| S03     | Sejfia [58]          | A Pilot Study on Architecture and Vulnerabilities: Lessons Learned.    | 2019 | Conference         |
| S04     | Mo et al. [59]       | Architecture Anti-patterns: Automatically Detectable Violations of Design Principles. | 2019 | Journal            |
| S05     | Maisikeli [60]      | Measuring Architectural Stability and Instability in the Evolution of Software Systems. | 2019 | Conference         |
| S06     | Carvalho et al. [61] | Investigating the Relationship between Code Smell Agglomerations and Architectural Concerns. | 2018 | Conference         |
| S07     | Shahbazian et al. [62] | Toward Predicting Architectural Significance of Implementation Issues. | 2018 | Conference         |
| S08     | Behnamghader et al. [63] | A large-scale study of architectural evolution in open-source software systems. | 2017 | Journal            |
| S09     | Mohsin et al. [64]   | Evaluating Dependency based Package-level Metrics for Multi-objective Maintenance Tasks. | 2017 | Journal            |
| S10     | Henrique et al. [65] | DCL 2.0: modular and reusable specification of architectural constraints. | 2017 | Journal            |
| S11     | Fontana et al. [66]  | An Experience Report on Detecting and Repairing Software Architecture Erosion. | 2016 | Conference         |
| S12     | Mo et al. [67]       | Decoupling Level: A New Metric for Architectural Maintenance Complexity. | 2016 | Conference         |
| S13     | De Oliveira Barros et al. [68] | Learning from optimization: A case study with Apache Ant. | 2015 | Journal            |
| S14     | Guimarães et al. [69] | Architecture-Sensitive Heuristics for Prioritizing Critical Code Anomalies. | 2015 | Conference         |
| S15     | Fontana et al. [70]  | Towards assessing software architecture quality by exploiting code smell relations. | 2015 | Conference         |
| S16     | Zengyang et al. [71] | An Empirical Investigation of Modularity Metrics for Indicating Architectural Technical Debt. | 2014 | Conference         |
| S17     | Ferreira et al. [72] | Detecting Architecturally-Relevant Code Anomalies: A Case Study of Effectiveness and Effort. | 2014 | Conference         |
| S18     | Macia et al. [73]    | Enhancing the Detection of Code Anomalies with Architecture-Sensitive Strategies. | 2013 | Conference         |
| S19     | Guimaraes et al. [74] | Prioritizing Software Anomalies with Software Metrics and Architecture Blueprints. | 2013 | Conference         |
| S20     | Macia et al. [75]    | Are Automatically-Detected Code Anomalies Relevant to Architectural Modularity? An Exploratory Analysis of Evolving Systems. | 2012 | Conference         |
| S21     | Zade et al. [76]     | Characteristics of multiple-component defects and architectural hotspots a large system case study. | 2011 | Journal            |
| S22     | Steff and Russo [77] | Measuring Architectural Change for Defect Estimation and Localization. | 2011 | Conference         |
| S23     | Sangwan et al. [78]  | Use of a multidimensional approach to study the evolution of software complexity. | 2010 | Journal            |
| S24     | Andrea and Thomas [79] | Software Engineering in Practice: Design and Architectures of FLOSS Systems. | 2009 | Conference         |
| S25     | Singh et al. [80]    | Reducing Maintenance Efforts of Developers by Prioritizing Different Code Smells. | 2019 | Journal            |
Afterward, implementing quality criteria, scores of six criteria were gathered, and the scope of the three levels was identified for quality assessment. If the range is between the scores 5.0 – 6.0, this means a high level; and if the range is between scores 4.5 - 3.5, this means a medium level; and if the range is between the scores 2.5 – 3.0, this means a low level. Consequently, most of the scores were of a high level, while three studies were excluded because they did not fulfill the bounded criteria. In Fig. 3, the classification of studies is demonstrated after implementing the quality criteria for the stated three levels. The scores of the quality assessment criteria for primary studies were explained in Appendix A.

4) DATA EXTRACTION AND SYNTHESIS

Once the selected studies have been approved and evaluated to be applied to the mapping study, the data extraction is begun from the primary papers to write down all the relevant data required to address the defined research questions. To simplify the data extraction process, we followed guidelines in the study [51, 52] to use data design forms that contribute to refining the list of required
attributes for this study. Table 5 describes the selected studies' reference details (SID, Title, Author, Year, and Publication source). Finally, the data extraction of the required attributes list was applied to 43 primary studies with a summarized description of all the attributes listed in Table 7. The extracted data of the required attributes list was stored in Endnote reference manager and the MS Excel spreadsheets.

FIGURE 3. Distribution of study quality assessment levels

It is significant to identify whether the results obtained can be synthesized in line with each specific research question regarding data synthesis. The data synthesis can contain the quantitative data if the data converge on a particular indicator that has thoroughly influenced the outcome. Data synthesis can contain descriptive data (non-quantitative). In this study, data were extracted to include descriptive data (e.g., used metrics, proposed used approaches, mapping quality, the context of the metrics, tool support, and validation process) and quantitative data (e.g., the extent impact of metrics accuracy with respect to addressing issues of the architectural decay).

IV. RESULTS

In this section, the results obtained according to the research questions that have been answered by the selected primary studies shown in Table 2 will be presented.

A. RQ1) WHAT CLASSIFICATIONS AND METRICS HAVE BEEN ESTABLISHED FOR ASSESSING ARCHITECTURAL EROSION?

To address RQ1, we conducted an exploratory study of all used approaches of architectural erosion metrics. Based on our exploratory study, which is shown in Table 8, we classify the methods to architectural erosion analysis into nine categories established on the analysis of two criteria for the primary studies (i.e., the study objective and the approach adopted to address architectural erosion): Historical data revision, Architectural bad smells, Architectural bad smells, Architectural complexity, Architectural dependencies, Architectural change, Architectural technical debt, Architectural dependency coupling, Architectural cohesion, Architectural complexity, and Analysis of software architecture size. All these classifications are defined in Appendix C.

Our classification makes it possible to determine the metrics related to each architectural erosion type. This classification of metrics is defined using the prioritization of the proposed metric of an empirical study based on two identified vital criteria: 1) metric repeatability in used approach, which means a measure is considered to be repeatable by how many this metric in stated classification is performed 2) obtained positive value, which refers to what extent the value is being met positively. Fig 4 summarizes the number of metrics per classification.

Regarding the adopted of approaches classifications, architectural dependency coupling analysis represents 26.66%, of all included studies, architectural bad smells 17.33%, architectural change 12.00%, architectural complexity 9.33%, architecture modularization, architectural technical debt, and analysis of software architecture size 8.00%, while architectural cohesion 4.00%. These categories are clarified by a number of papers in Fig 5.
Figure 5 depicts metrics distribution in terms of classifying architectural erosion we considered. It shows that architectural dependency coupling analysis is measured by 24.17% of all well-chosen metrics from studies, making it most widely used for developing a mechanism of metrics approaches to address issues of architectural erosion. In addition, the architectural change is measured by 13.18%; architectural bad smells are measured by 12.08%; historical data revision and architecture modularization are measured 10.98%; analysis of software architecture size is measured by 9.89%; architectural complexity is measured by 7.69%, while architectural technical debt and architectural cohesion are measured by 4.49%.

B. RQ2) WHICH METRICS HAVE BEEN MAPPED TO EACH QUALITY ATTRIBUTE?
The results indicate that most of the metrics directly relate to the mapping of quality attributes to address architectural erosion by investigating the quality attributes of the architecture. Therefore, we assess ranking

### TABLE 7. DATA EXTRACTED ATTRIBUTES LIST OF THE PRIMARY STUDIES

| Attribute     | Description                                      |
|---------------|--------------------------------------------------|
| Study ID      | Unique identifier per study.                     |
| Year          | To specify the year in which the paper is published. |
| Study Type    | Identify the publication source such as journal, conference. |
| Study Title   | Summarize the topic that takes precedence over other topics. |
| Study Focus   | Outline the main goal concerning motivation and use approaches to justify the extent of approaches effectiveness. |
| Measures of architectural decay | To clarify the name and description of metrics, mapping quality by which the metrics related to, granularity level of the metrics, the context of the metrics, usability, applicability, and comparative analysis among metrics itself. |
| Validation process | Describe the validation process regarding the state metrics to identify the extent of effectiveness to address architectural decay. |
| Obtained results | Clarifying the key findings to identify positive or negative impact by which metrics were performed. |
| Tool Support  | Are the used metrics supported by the tool or not, and which is the tool’s name? |

### TABLE 8. ADOPTED METRICS AND APPROACHES CLASSIFICATIONS

| Study ID | Adopted classifications | Abbreviation of the defined metrics |
|----------|--------------------------|------------------------------------|
| [S02], [S04], [S25], [S36], [S38]. | Historical data revision. | NC, NCL, Severity, CF, TF, BCF, PCF, BC, CCH, NCMS. |
| [S05], [S14], [S15], [S17], [S19], [S20], [S30], [S31], [S32], [S34], [S37], [S41], [S42]. | Architectural bad smells | PCD, PCC, Fan-in, Fan-out, NCC, CDC, CDO, CDLOC, NAC, COL, CC. |
| [S03], [S06], [S09], [S12], [S16], [S26]. | Architecture modularization | NOCL, IPMD, IPMC, IPMCD, IPMC, Mnaewm, Mgdkg, Mrcc, Mbunch, MQ. |
| [S01], [S03], [S05], [S07], [S08], [S13], [S22], [S23], [S42]. | Architectural change | CMC, IMC, PDI, PCI, RBMS, ASM, CVG, MoJo, CD, Abstractness(A), Instability(I), SD. |
| [S02], [S16], [S26], [S29], [S34], [S40]. | Architectural technical debt | DL, PC, MI, SIG, SQALE, IIPU, IIE, PCI, IPUD, IPED, IPGF, ANMCC. |
| [S02], [S03], [S09], [S11], [S13], [S18], [S19], [S23], [S24], [S27], [S28], [S30], [S31], [S32], [S34], [S35], [S39], [S40], [S42], [S43]. | Architectural dependency coupling | CMD, OMD, TCMD, TOMD, IMD, EMD, MDS, DFSM, TWD, AEL, CBO, SOC, EWSOC, LWSSOC, RFC, ROC, Ca, Ce, MSC, DAC, NOD (Cost Function), ATFD. |
| [S02], [S13], [S33]. | Architectural cohesion | PCQ, LCOMP, LCOM, TCC, RCI. |
| [S02], [S05], [S11], [S21], [S23], [S25], [S34]. | Architectural complexity | SCC, CC, MCC, WMC, XS, DS, DP. |
| [S02], [S05], [S10], [S13], [S35], [S37]. | Analysis of software architecture size | LOC, TNC, NCON, NELEM, NCML, CLD, NOS, Public API, Class elegance. |
TABLE 9. MAPPING METRICS THE ARCHITECTURALLY RELEVANT ATTRIBUTES

| Study ID | Quality attribute | Metrics abbreviation |
|----------|-------------------|-----------------------|
| [S01], [S03], [S05], [S10], [S21], [S23], [S27], [S28], [S30], [S31], [S34], [S38], [S39], [S40]. | Analyzability | PDIpk, PClpk, PClc, PClcl, PKDI, PKCl, Instability, RCI ROC, Ca, Ce, Abstractness, MDS, LOC, FAN-IN, FAN-OUT, CF, TF, BCF, PCF, SOC, EWSOC, LWSOC, XS. |
| [S01], [S12], [S13]. | Reusability | PDIpk, PClpk, PClc, PClcl, PKDI, PKCl, CBO, Ca, Ce, Class elegance, NOD (Cost Function), CD |
| [S03], [S09], [S12], [S13], [S16], [S22], [S26], [S33], [S36], [S37]. | Modularity | MQ, WMC, LOC, LCOM, CBO, NCC, CDC, CDO, RFC, FAN-IN, FAN-OUT, CF, TF, BCF, PCF, NAC, COL, COC, IPMD, IPMC, IPMCD, PCD, CDLOC, IPMCC, Mnewm, Mg&g, Mrcc, Mbunch, MQ, DL, PC, PCQ, LCOMP, IIPU, IIPC, IIPUD, IIPED, IPGF, ANMCC. |
| [S04], [S18], [S22], [S34]. | Modifiability | BF, BC, CF, CCH, FAN-IN, FAN-OUT, CBO, RFC, LOC, XS, MCC, IPMCD, PCC. |
| [S03], [S05], [S07], [S08], [S10], [S14], [S15], [S25], [S29], [S34], [S42], [S43]. | Maintainability | NOCL, WMC, a2a, CVG, CDC, CDO, LOC, TCC, ATFD, CC, Severity, Instability, FAN-IN, FAN-OUT, Ce, Ca, LCOM, CBO, MCC, TNC, MoJo, CD, MI, SIG, SQALE, DC, DP |
| [S05], [S06]. | Usability | FAN-IN, FAN-OUT, NOD (Cost Function). |
| [S25], [S35], [S36]. | Understandability | LOC, NCONN, NELEM, FAN-IN, FAN-OUT, CF, TF, BCF, PCF, CC, TCMD, CMD, OMD, TMOD, IMD, EMD, SCC, LOC, NCONN, NELEM, FAN-IN, FAN-OUT, CF, TF, BCF, PCF, CC, TCMD, CMD, OMD, TMOD, IMD, EMD, SCC, LOC, NCONN, NELEM, FAN-IN, FAN-OUT, CF, TF, BCF, PCF, CC, TCMD, CMD, OMD, TMOD, IMD, EMD, SCC, LOC, NCONN, NELEM, FAN-IN, FAN-OUT, CF, TF, BCF, PCF, CC, TCMD, CMD, OMD, TMOD, IMD, EMD, SCC, LOC, NCONN, NELEM, FAN-IN, FAN-OUT, CF, TF, BCF, PCF, CC, TCMD, CMD, OMD, TMOD, IMD, EMD, SCC, LOC, NCONN, NELEM, FAN-IN, FAN-OUT, CF, TF, BCF, PCF, CC, TCMD, CMD, OMD, TMOD, IMD, EMD, SCC, LOC, NCONN, NELEM, FAN-IN, FAN-OUT, CF, TF, BCF, PCF, CC, TCMD, CMD, OMD, TMOD, IMD, EMD, SCC, LOC, NCONN, NELEM, FAN-IN, FAN-OUT, CF, TF, BCF, PCF, CC, TCMD, CMD, OMD, TMOD, IMD, EMD, SCC, LOC, NCONN, NELEM, FAN-IN, FAN-OUT, CF, TF, BCF, PCF, CC, TCMD, CMD, OMD, TMOD, IMD, EMD, SCC, LOC, NCONN, NELEM, FAN-IN, FAN-OUT, CF, TF, BCF, PCF, CC, TCMD, CMD, OMD, TMOD, IMD, EMD, SCC, LOC, NCONN, NELEM, FAN-IN, FAN-OUT, CF, TF, BCF, PCF, CC, TCMD, CMD, OMD, TMOD, IMD, EMD, SCC, LOC, NCONN, NELEM, FAN-IN, FAN-OUT, CF, TF, BCF, PCF, CC, TCMD, CMD, OMD, TMOD, IMD, EMD, SCC, LOC, NCONN, NELEM, FAN-IN, FAN-OUT, CF, TF, BCF, PCF, CC, TCMD, CMD, OMD, TMOD, IMD, EMD, SCC, LOC, NCONN, NELEM, FAN-IN, FAN-OUT, CF, TF, BCF, PCF, CC, TCMD, CMD, OMD, TMOD, IMD, EMD, SCC, LOC, NCONN, NELEM, FAN-IN, FAN-OUT, CF, TF, BCF, PCF, CC, TCMD, CMD, OMD, TMOD, IMD, EMD, SCC, LOC, NCONN, NELEM, FAN-IN, FAN-OUT, CF, TF, BCF, PCF, CC, TCMD, CMD, OMD, TMOD, IMD, EMD, SCC. |
| [S02] | Functional Suitability, Reliability | LOC, NOS, LCOM, FAN-OUT, FAN-IN, NOC, RFC, WMC, Public API, CLD, MPC, DAC, SOC, MCC. |
| [S05], [S11], [S17], [S42]. | Performance efficiency | FAN-IN, FAN-OUT, LOC, NOC, WMC, LCOM, TCC, FAN-OUT, ATFD, RFC, CBO, CC, NOCL, MCC. |

TABLE 10. METRICS WITH THE HIGHEST NUMBER OF OCCURRENCES

| Metrics abbreviation | Total occurrences |
|----------------------|-------------------|
| PKDI, PKCl, PCIPk, PClpk, PClc, PClcl, PD, MDS, LWSOC, NOD, Class elegance, IPMD, IPMC, CCH, MCC, Severity, NCONN, Public API, DAC, Two-way dependencies (TWD), ROC, ST, EWSOC, IPMCD, BC, TCMD, NODI, NOS, CLD, MPC, PCC, NCML, DFSM, AEL, NAC, COL, NC, NCL, NCMs, PCD, COC, IPMCC, Mnewm, Mg&g, Mrcc, Mbunch, MQ, CMC, INCl, RBMS, ASN, MoJo, CD, SD, SIG, SQALE, CMD, OMD, TMOD, IMD, EMD, PCQ, LCOMP, RCI, SCC, DC, DP, NELEM, IIPU, IIPC, IPDI, IIPUD, IPF, IPGF, ANMCC. | 1 |
| TF, BCF, CDLOC, BF, TCC, MCC, MQ, NCC, ATFD, CC, TNC, SOC, DL, PC, MI | 2 |
| CF, a2a, CVG, CDC, CDO, NOCL, PCF, Instability, XS, Abstractness, RFC WMC. | 3 |
| NOC, LCOM, Ca, Ce. | 5 |
| CBO. | 6 |
| FAN-IN. | 7 |
| LOC. | 9 |
| FAN-OUT. | 10 |
| VOLUME XX, 2017 | 11 |

criteria of metrics mapping based on two identified key criteria [98]: 1) measure's significance in terms of reliability, which refers to the study's primary goal in terms of identified architectural quality attributes. 2) repeatability, which refers to what extent the metric is being used or applied to the quality properties to address the erosion.

The mapping process for the metrics used to identify quality characteristics is based on the general goal of architectural erosion that quantitatively estimates more profound quality attribute problems.
The results also indicate that analyzability, modularity, modifiability, and maintainability are the most investigated measures to address the degradation. Additionally, some quality characteristics have less effect than the above-stated characteristics, such as usability, understandability, reusability, reliability, and performance efficiency. The metrics abbreviations are defined in Appendix B. Table 9 summarizes mapping metrics to the architecturally relevant quality attributes.

The results also show the metrics with the highest number of occurrences used for architectural quality qualities in resisting erosion and identifying essential measures and their function for addressing the issue of architectural decay. The results also indicate that metrics such as FAN-OUT, FAN-IN, the coupling between objects (CBO), afferent coupling (Ca), and efferent coupling (Ce), size metrics such as line of code (LOC), number of clusters (NOCL), number of classes (NOC), and weighted methods per class (WMC), and Lack of Cohesion in Methods (LCOM) are most commonly used metrics among the various attributes of architectural quality. Table 10 shows the metrics occurrences among the different quality attributes.

C. **RQ3.1 WHAT ARE METRICS VALIDATION CRITERIA IN THE CONTEXT OF ARCHITECTURAL EROSION?**

Assessing the validation of metrics in software engineering is vital for understanding, controlling software development practices, and helping build quality into software in the erosion metrics validations. As a result, evaluating architectural erosion metrics entails finding defects in a system and determining whether it will be eroded or evolved based on various criteria.

**A: RQ3.1 WHAT VALIDATION APPROACHES ARE USED FOR ARCHITECTURAL EROSION?**

Each metric must be validated in the software measurement. There are two approaches to software metrics validations: theoretical and empirical validations [99, 100]. The theoretical validation refers to details on the mathematical and statistical operations that may be done with the measure, which is significant when dealing with it. While empirical validation refers to the extent to which a test's, models, or other construct's accuracy can be shown through systematic and experimental observation (i.e., the collection of supporting research evidence) instead of theory alone. Figure 6 depicts the two approaches of software metrics validations.

We classified the studies into four categories concerning the validation approaches for these architectural erosion metrics: theoretical validation, empirical validation, complete validation (i.e., both theoretical and empirical), and no validation (i.e., neither empirical nor theoretical). The last two classifications were established based on primary study analysis and derived from similar studies to assess the validity of metrics [45, 48].

The results show that all of these included studies involve only the complete validation or empirical validation. In contrast, the empirical validation represents 79% of the studies that followed these metrics for this validation. In comparison, the complete validation represents 21%. The theoretical and no validation have not been stated in these included studies independently. Furthermore, empirical validation in some studies [S02, S03, S16, S18] shows that some metrics, such as the Sqale index, Sqale debt ratio, IIPU, IIP, IIPUD, IIPED, and AEL have no positive impact on contributing to address architectural erosion. Table 11 demonstrates validation approach types.

**TABLE 11. VALIDATION APPROACH TYPES**

| Study ID | Validation approach |
|----------|---------------------|
| [S01], [S02], [S03], [S04], [S05], [S07], [S08], [S11], [S12], [S13], [S15], [S16], [S18], [S19], [S21], [S22], [S23], [S24], [S25], [S26], [S28], [S30], [S32], [S33], [S34], [S35], [S36], [S37], [S38], [S39], [S40], [S42], [S42], [S43], [S06], [S09], [S10], [S14], [S17], [S20], [S27], [S29], [S31] | Empirical validation |
| [S01], [S02], [S03], [S04], [S05], [S07], [S08], [S11], [S12], [S13], [S15], [S16], [S18], [S19], [S21], [S22], [S23], [S24], [S25], [S26], [S28], [S30], [S32], [S33], [S34], [S35], [S36], [S37], [S38], [S39], [S40], [S42], [S42], [S43], [S06], [S09], [S10], [S14], [S17], [S20], [S27], [S29], [S31] | Complete validation |

![Software Metrics Validations](image)

**Figure 6.** depicts the two approaches of software metrics validations

**B: RQ3.2 WHAT THE VALIDATION LEVEL AND CASE STUDY ARE APPLIED TO ARCHITECTURAL EROSION METRICS?**

Concerning the validation level that indicates how well the metrics have been empirically validated, we classified the studies into four levels: small experiment shows the metrics that have been empirically validated using one system or a few small to medium-sized systems, large experiment the metrics that have been empirically validated using many small to medium-size systems or one / few large systems, independently validated metrics indicate the metrics proposed by some researchers have been successfully validated by other authors team, and their applicability to the specific context has been confirmed, and unknown experiment indicates the metrics that have been empirically validated using unspecified detail of target systems. These classifications were inspired based on supported evidence research as shown in [45].

The results show that the large experiment is the most commonly used to apply metrics. In contrast, the large experiment represents 58%, the small experiment represents.
30%, independently metrics validated represents 7%, and the unknown experiment represents 5%. Table 12 shows the validation level for included studies.

![Common case study of systems among limited studies](image)

**FIGURE 7.** Common case study of systems among limited studies

**TABLE 12. VALIDATION LEVEL**

| Study ID | Validation level |
|----------|------------------|
| [S01], [S16], [S20], [S21], [S26], [S27], [S32], [S34], [S36], [S37], [S39], [S40], [S43], [S02], [S03], [S04], [S05], [S06], [S07], [S08], [S09], [S11], [S13], [S14], [S15], [S17], [S19], [S23], [S24], [S25], [S28], [S29], [S30], [S31], [S35], [S38], [S41], [S42], [S10], [S18], [S22]. | Small experiment |
| [S12], [S33]. | Independently validated |
| [S05], [S14], [S20], [S29]. | Large experiment |
| [S17], [S30]. | Unknown experiment |

**TABLE 13. CASE SYSTEMS TYPE**

| Study ID | Case systems type |
|----------|------------------|
| [S01], [S02], [S03], [S04], [S05], [S06], [S07], [S08], [S09], [S10], [S12], [S14], [S15], [S17], [S18], [S20], [S21], [S23], [S24], [S25], [S26], [S29], [S31], [S34], [S36], [S37], [S39], [S05], [S11], [S14], [S19], [S20], [S27], [S28], [S35], [S38], [S40], [S13], [S16], [S22], [S29], [S30], [S32], [S33], [S41], [S42], [S43]. | Open-source system case study |
| [S17], [S30]. | Industrial system case study |
| [S17], [S30]. | Unknown system case study |

The findings also indicate three categories’ systems for the case study employed in the research: open-source, industrial, and unknown systems. The case study of open-source systems is the most frequently applied to the proposed architectural erosion metrics. Thus, open-source systems represent 58%, industrial systems represent 21%, and unknown systems represent 21%. Table 13 shows the case study systems that have been validated in the studies. Furthermore, some studies may apply a mixture of the case study of the systems such as what appears in the studies [S05], [S14], [S20], and [S29], whether they are open-source, industrial, or unknown systems. As seen in Fig 7, it represents the number of systems used in each study.

D: **RQ3.3 HOW COULD THE EXTENT OF METRICS EFFECTIVENESS AMONG SEVERAL STUDIES BE IDENTIFIED BASED ON COMPARATIVE ANALYSIS?**

To identify the comparative analysis regarding the effectiveness of the metrics, whether, within the same study or when compared to another study, The findings show a wide range of efficiency when it comes to analyzing the metrics for studying a given topic of architectural erosion. The study [S02] compares several metrics to describe the classes contributing to architectural decay. Clarifying a predominance a2a metric on a critical deficiency of the widely used MoJo FM metric was the focus of the study [S08]. In the study [S09], a comparative analysis is conducted with Abdeen’s metrics (AbdeenMod) against Martin’s package level metrics through Logistic Regression (LR) model.

The reliability of the decoupling level measure (DL) in comparison to the propagation cost (PC) and independence level (IL) metrics was investigated in the study [12]. In the study [S17], they are comparing the inspection-based strategy with an ad hoc metrics-based strategy overall. Finally, the study [S18] compares traditional detection metrics with architecture-sensitive metric strategies.

The underlying principle is included in all measurement approaches and methodologies that may be used in the various analysis, whether in one study or across multiple investigations. As a result, it’s critical for researchers to select software metrics for deterioration architecture that most closely fits and provides detection capabilities for the study, particularly In the early stages of development.

D: **RQ4 ARE THERE TOOLS THAT SUPPORT AUTOMATICALLY CALCULATING ARCHITECTURAL EROSION METRICS?**

Some research directly disclosed the tools used to automate calculating metrics. These tools provide a set of metrics for engineers and researchers to compute an accurate value that can detect the architecture and code problems.

The results also indicate several different tools regarding the number of metrics involved and the number of their use frequency within the studies based on the importance of the particular tool and the approaches designed to address an issue.

The Understand tool represents the most considerable used portion among other tools within the included studies. In addition, CJKM, SonarQube, PMD, FindBugs, and Together tools significantly impact researchers due to the effectiveness of the metrics in which implemented. Table 14 illustrates the tools used for calculating architectural metrics.
erotion metrics.

TABLE 14. TOOLS USED FOR CALCULATING ARCHITECTURAL EROSION METRICS

| Study ID            | Tool used                        |
|---------------------|----------------------------------|
| [S02], [S22]        | CKJM                             |
| [S02], [S34]        | SonarQube                        |
| [S02], [S09]        | PMD, FindBugs                    |
| [S02], [S03], [S42]| Git, VizzAnalyzer, and SourceMonitor |
| [S04], [S09], [S14], [S20], [S36], [S42], [S11], [S12], [S13], [S14], [S16], [S17], [S18], [S20], [S18], [S20], [S24], [S29], [S41], [S33], [S34] |                                      |
|                     | ARCADE                           |
|                     | Understand                        |
|                     | DFMC4J                           |
|                     | Titan                             |
|                     | PF-CDA static analysis           |
|                     | NDepend                           |
|                     | ModularityCalculator             |
|                     | SCOOP                             |
|                     | Hist-Inspect, inCode             |
|                     | Sonar, MuLAtO                    |
|                     | Together                          |
|                     | Doxygen source code documentation generator |
|                     | Arcan                             |
|                     | Package Cohesion Evaluator       |
|                     | inFusion (InF) and Structure101  |

E: RQ5) WHAT IS THE REPRESENTATIVE APPROACH FOR THE METRIC CONTEXT USED?

Concerning the outline context and methodology of the metrics so as to identify the used representation model during data analysis, we tend to classified all the studies into three approaches of the context of the metric supported the identical classification in [45]: i) internal structure-based metrics, which refer to metrics based on the internal structure and their relationships of the higher-level artifacts (components, modules, packages, etc.), ii) specific model-based metrics which refer to metrics that are defined using a specific representation or model of the system’s structure, iii) and graph-based metrics which refer to metrics for a certain system that is based on a graph model as a set of nodes and edges.

The results show that the internal structure-based metrics are most widely used in the context of the metric according to included studies, while specific model-based metrics play a prominent role in the context of the metric for identifying representation of a specific model within the system structure. Graph-based metrics have no significant impact on architectural erosion despite the importance of the context of these metrics used in other software architecture concepts. The approaches to the types of the specified metric are shown in Table 15.

F: RQ6) TO WHAT EXTENT ARE THE METRICS APPLIED IN THE CONTEXT OF ARCHITECTURAL EROSION?

Regarding the essential parameters to identify the extent of applicability of these metrics within the context of architectural erosion considering what proportion usage or future prospects provided by the authors, we classified these parameters into three criteria: i) future prospects and usage, which means only a small amount of knowledge about future prospects of metrics applicability, ii) requiring further improvements which means to provide a research analysis that can be applied in future studies, but it still needs to be improved or additional evaluations for suitable assessment criteria, and iii) successful applicability which means the measures can be used successfully in real-world applications.

The results indicate that 90% of studies represent the applicability of the metrics in real projects successfully, 5% of the studies contribute to useful analysis that can be applied for future directions of the research but still need further improvements and experiments or more logical assessment criteria to be applied in real-world applications, and 5% of studies concern about obtaining some information to clarify the future prospects and usage of the applicability of the metrics. Fig 8 demonstrates the three essential parameters of metrics applicability.

![FIGURE 8. The extent of the applicability of the metrics within the context of architectural erosion](image-url)
TABLE 15. THE APPROACH OF THE METRICS TYPE

| Metric | The approach of metrics type |
|--------|-----------------------------|
| PDlpk, PClpk, PDlcl, PClcl, PKDI, NC, NCL, Severity, CF, TF, BCF, PCF, BC, CC, NCms, PCD, PCC, Fan-in, Fan-out, NOCL, IPMD, IPMC, IPMCD, IPMCC, CMC, IMC, PDI, PCI, ASM, CD, Abstractness (A), Instability (I), DL, CMD, OMD, TCMD, TOMD, IMD, XMD, DFSM, TWD, AEL, CBO, SOC, EWSOC, LWSOC, RFC, ROC, Ca, Ce, MFC, DAC, NOD, ATFD, PCQ, LCOMP, LCOM, TCC, RCI, SCC, WMC, Class elegance, LOC, TNC, NCONN, NELEM, NCML, CLD, NOS, NCC, CDC, CDO, CDLOC, NAC, COL, COC, Mrcc, MQ, RBMS, cvg, a2a, MoJo, PC, MI, SIG, SQALE, MDS, CC, DC, DP, Public API, Mnewm, Mg&g, Mbunch, SD, MCC, XS. | Internal structure-based metrics |
| | Specific model-based metrics |
| | Graph-based metrics |

A: CLASSIFICATIONS AND ADOPTED METRICS (RQ1)

We observed classifications and defined metrics per category based on our exploratory study to address the issues of architectural erosion. This classification reveals the diversity of proposed architectural erosion solutions based on the adopted approaches by the authors in their studies. Furthermore, this classification provides researchers with analyzing methods that would open a wide field for them to address architectural erosion from several different perspectives, including the combination of these classifications or approved metrics to provide solutions that could have a higher impact or efficiency than current solutions. This classification also reveals for researchers and practitioners the ambiguity on studying the phenomenon of erosion directly concerning the methodology used to address this phenomenon. The classifications of the architectural dependency coupling analysis, architectural bad smells, and architectural change represent the familiar picture of most of the approaches and methods proposed by researchers to address the phenomenon of erosion than other classifications, due to the several metrics that are included in these classifications, as well as several studies that are based on the analysis of relationships and dependencies among architecture artifact of abstraction level (such, model, package, file).

Regarding the defined metrics per classification, the architectural dependency coupling analysis has several metrics compared with another classification. This classification of dependency coupling metrics is the most commonly used and popular among the other metrics. It can be stated that these measures, by their nature, are focused on linking and connecting components, files, modules, packages, classes, and methods with each other. This is made the researchers focus on dependency coupling metrics to calculate metric values between the architecture files. In addition, the classifications of architectural change and architectural bad smell comprise several metrics that may be similar to the study of erosion with the dependency coupling analysis. The remaining metrics of other classifications may address the architectural erosion in a few approaches proposed by the authors. Still, they will not achieve the same high level as the three categories stated.

B: MAPPING METRICS TO EACH QUALITY ATTRIBUTE (RQ2)

We observed that the quality attributes representing maintainability, analyzability, modularity, reusability, and modifiability are the most addressed by researchers in their study by linking metrics to them to address architectural erosion. It reveals that these attributes are all fall under the maintainability attribute, which represents the degree of effectiveness and efficiency with which a system can be altered to evolve, correct, or conform to changes in the environment and requirements. Although some studies [S03], [S05], [S07], [S09], [S10], [S14], [S15], [S25], [S29], [S32] have linked the metrics mapping to maintainability attribute since the researchers’ correlated metrics of the maintenance efforts with the relationship of architectural erosion in general without broaching the sub-feature specifically.

The architectural structure is measured by knowing its deterioration or evolution through its maintainability attributes and sub-features. We also revealed that other quality attributes impact the extent of the erosion of the architecture, such as understandability, performance efficiency, and reliability. Still, this effect is not considered when compared with the maintainability attribute and its sub-features.

We also observed that metrics with the highest number of occurrences between the various quality characteristics are dependency coupling and size metrics. This explains that the dependency coupling metrics play a significant role in identifying the architectural erosion based on the description of the stated attributes, and it also indicates an ability of analysis depth of these metrics for addressing the degradation and an extent of providing sufficient results, especially FAN-OUT, FAN-IN, afferent coupling, and efferent coupling metrics. We also revealed that the role of size metrics has a significant effect on identifying architectural erosion compared with the dependency coupling metrics. From this, we conclude that it is possible that if these metrics are integrated with more than one metric rather than investigating them in isolation through a
broader and deeper analytical way, it will provide critical importance results to address architectural erosion. Therefore, this is a new trend in the future of our study, whereas these metrics will be evaluated after the appropriate integration among them to form the appropriate model for determining the eroded or evolved architecture based on the measures.

C: VALIDATION OF THE METRICS (RQ3.1)
We observed that the validation approach involves complete validation and empirical validation. In contrast, the other two validation approaches, theoretical and no validation, have not been stated in any study. This explains that the empirical and complete validation is closely related to the metrics mechanism's nature. It is impossible to conduct a theoretical validation for architectural erosion metrics without empirical validation. From this point, researchers must identify the appropriate validation mechanism for any topic to address the issue.

The empirical validation, which represented the majority of included studies, is the majority because the metrics cannot be empirically applied unless they can be theoretically applied. This is what combines the two types. Therefore, it does not need to be theoretically proven to validate the theoretical type, but it is possible to prove theoretically to clarify the content of metrics work.

Several researchers have focused their efforts on validating the extent to which metrics are efficient in detecting architectural problems that cause erosion. Some metrics that were found out in studies [S02], [S03], [S16], [S18] did not provide sufficient impact and results for addressing architectural erosion and disclosing architecturally relevant concerns, but that does not indicate they are insufficient or unsuitable for this concept. Thus, some metrics have been evaluated in more than one study, such as instability metrics in the study [S03] has no impact in terms of analyzing the correlation between architecture changes, decay, and the presence of vulnerabilities while in studies [S06] and [S25] provided generally significant results, especially when it is accurately identified in efferent coupling (Ce) and afferent coupling (Ca) metrics. We also observed that the architectural debt metrics, represented Sqale index and Sqale debt ratio metrics in the study [S02], are entirely inadequate in describing classes contributing to the architectural inconsistencies. However, they may have a significant impact when used on a concept of appropriate importance concerning various architectural problems.

This is an important open issue for researchers to conduct further studies and validate some metrics and apply them to many concepts subject to the architectural erosion concept to ensure the extent of metrics sufficiency, especially when applied to different systems in terms of domain architecture type and size.

D: VALIDATION LEVELS (RQ3.2)
We found out that the large validation experiment was the majority among other validation levels. This explains the reliability of the obtained metrics results since that validation depends on many systems, whether small or medium-sized or a few large systems. Also, some researchers used metrics to verify architectural erosion based on the metrics proposed by some researchers that other authors have successfully validated, and their applicability to the specific context has been confirmed. This also reveals that the validation level of these metrics clearly presents its work and extent to its applicability has been appropriately carried out, thereby reducing the extent of the metrics results biased to address architectural erosion.

We also observed that most of the case study of the system used is open-source systems because these systems are available and easily accessible. In addition, some studies validated the results reliability of the metrics using a combination of the case study of systems between open-source systems, industrial systems, and non-described systems used in previous studies, as shown in Fig 7. This explains that some researchers generalize the results using different domains, sizes, and architecture types.

Although most studies have validated these metrics based on extensive experiments, researchers have a considerable opportunity to conduct more research on integration from different system contexts such as academic and student systems that are not addressed in prior studies depending on different sizes of fields and architecture types.

E: COMPARATIVE ANALYSIS TO IDENTIFY THE EFFECTIVENESS OF THE METRICS (RQ3.4)
Considering the effectiveness of the metrics among each other based on the comparative analysis performed by many researchers, whether in their study or a comparison with other studies through different metrics, we observed that the comparative analysis would either be built on metrics strategies generally without indicating to specific metrics as in studies [S17], [S18] or metrics will have to be distinguished from the rest of the metrics in terms of efficiency, the supported specific tool for them and the particular issue in which the metrics can provide essential results as in studies [S02], [S08], [S09].

This explains that the metrics strategies generally may explore the comparison by highlighting the element and effort required to evaluate the architectural erosion regardless of metrics that may provide desired results. The concept is confined to the general strategy to which all metrics are subject. While comparison analysis may study one metric and another to determine the reliability and reasonable accuracy of predicting errors, improving performance, and responding to identify the extent of the architectural erosion.
in other studies without specifying the used tool, such as the instability metric.

We also observed that there are metrics or what is called the metrics strategy has been stated in general, and indication to the tool that includes these metrics within another study by specifying the reference as in the study [S16]. We also found out that the tool may be incorporated into the integrated development environment (IDE), which leads to ambiguity, especially for new relevant readers to the topic. Therefore, researchers may not realize this issue when writing their articles.

G: APPROACH OF THE METRICS CONTEXT (RQ5)
We observed that an approach classification of the metric context was divided into internal structure-based metrics, specific model-based metrics, and graph-based metrics. We have relied on the study [45] concerned with software metrics to measure the ability to understand the architectural structures to make the classification due to convergence of the two studies in terms of the distribution of metrics to capture the big picture of the system architecture.

We also noticed that the context of the metric represented by internal structure-based metrics was most commonly and frequently used in the studies than other approaches of the context of the metric. This reveals the internal structure-based metrics based on higher-level artifacts (i.e., components, files, systems, projects, modules, packages) and their relationship. Thus, the answer to (RQ1) regarding the metrics mechanism approach indicated that most of the reviewed papers were based on analyzing relationship correlation between issues relevant architecturally or inconsistencies based on metric values.

Concerning graph-based metrics, the software artifacts and their interaction represent a set of nodes and edges.

Although the inherent simplicity of graph-based metrics and their use widely in software engineering, it appears a scarcity within metrics of architectural erosion. This may explain its inadequacy significantly, or many researchers have not studied them for identifying architectural deterioration.

H: APPLICABILITY OF METRICS (RQ6)
We observed that the applicability of metrics is divided into three significant parameters. The results revealed that the applicability of the metrics within real-world applications represents the majority of studies through a considerable percent exceeding 90% compared to the rest.

This explains that the applicability of these metrics to real systems has provided a strong indication and motivation for many researchers to adopt addressing architectural erosion through the metrics strategy of various levels and types. Therefore, it could be that the concept of metrics strategy to present as a solution that has a significant impact when compared with other solutions as was demonstrated in our prior study [7], because of the widespread use and prevalent applicability of metrics successfully, whether a solution of the metrics strategy is being applied as isolation or combination with other mechanisms to provide a solution integration of importance or reduce the defects for some metrics.

As for good analysis that requires further improvements and experiments, we noticed a small percentage within the studies. Instead, it represents only two out of 43 studies. Nevertheless, sound analysis and a context of importance were presented to pave the applicability of these metrics’ types within the real projects by conducting further improvements and experiments to identify positive or negative effects.

VI. IMPLICATIONS FOR RESEARCH AND PRACTICE
Since our key objective was to characterize metrics of the architectural erosion based on the profound analysis in terms of investigating validation approaches and levels, mapping erosion metrics to each quality attribute, identifying the applicability metrics, comparative analysis conducted to determine the effectiveness of metrics, and specifying a tool-supported for calculating architectural erosion metrics thereby a mapping study provides implications for researchers and practitioners on characterizing the architectural metrics as follows:

The study presented the mechanisms of adopted metrics approaches that have been shown to address the issues of architectural erosion. In contrast, it shows that the relationship analysis between components of architecturally related issues is essential among researchers for adopting a mechanism of the metrics approaches used. Thus, further studies need to be conducted from several aspects, factors, and several different metrics to identify the correlation and the extent to which software architecture negatively or positively impact. Practitioners should recognize all the mechanisms of effective metrics approaches to avoid the problems leading to architectural erosion, particularly the beginning of the design and analysis of systems architecture.

Based on the approaches used by the authors in their primary studies, the classification reveals a variety of proposed architectural decay solutions. This classification provides researchers with analyzing methods that will allow them to handle architectural erosion from various perspectives, including the combination of these classifications or approved metrics to provide solutions that may have a more significant impact or efficiency than current solutions. Furthermore, the classification also reveals to researchers and practitioners the ambiguity in studying erosion directly in terms of the methodology used to address this phenomenon.

Mapping metrics to quality attributes to identify architectural erosion lies considerably in maintainability, analyzability, modularity, reusability, and modifiability, representing the degree of effectiveness and efficiency with which a system can be altered to evolve and correct or conform to changes in the environment and requirements. Also, the highest number of metrics occurrences between the various quality characteristics appeared dependency coupling and size metrics. This
reveals that there is an open opportunity for researchers to shed light on the quality characteristics that affect the architecture in terms of degradation or evolution, as well as the high-impact metrics and the change between metrics and the appropriate characteristics to study the relationship, impact, and change that may occur on the architectural software based on the composition of the metrics and quality attributes that are directly related to the determination of the erosion or evolution of the architecture.

The study revealed that the empirical and complete validation approaches are considerably applied to the metrics mechanism and validation level described by extensive validation experiments. Thus, researchers must identify the appropriate validation mechanism for any topic to address an issue. From this point, one direction for a considerable opportunity for researchers to conduct more empirical research addressing this issue to overcome the defined implementation of the analyzed approaches in practice [101] based on the combination from different system contexts such as academic and student systems that are not addressed in prior studies in terms of various sizes, fields, and architecture types.

The comparative analysis performed by many researchers through different metrics in their study showed a significant lack of empirical validation. However, few studies appear to be concerned with the comparative analysis, and the surprising thing is that metrics strategy generally refers to a comparison analysis of metrics. Therefore, this is an open opportunity that provides researchers to conduct further studies on comparative analysis to identify important factors that may provide highly desirable results, such as the effort required for assessing architectural erosion, identifying reasonable accuracy in terms of detecting errors, and predicting critical architectural problems, improving performance and responding to determine the characterization of the extent of architecture erosion.

From an academic point of view, we did not find in the included studies that the maximum benefit of the tool was offered when calculating the metrics using the tools, but rather that it was limited to focusing only on specific metrics and calculating them through a supported tool. This explains why these tools have not been provided with a comprehensive concept to investigate the importance of the tools and their strength in comparison to each other and to achieve the maximum benefit, including determining the development or erosion of the architecture using metrics in which it included. Thus, several researchers have only dealt with the tools that include metrics to achieve the required study and data analysis.

VII. THREATS TO VALIDITY OF THE STUDY

Although our study was performed following the predefined set of protocol rules that might reduce the research bias within the mapping study, threats affect the study's validity concerning the correlation between the conclusion and results obtained. For example, we can identify some threats represented by search incompleteness, a process bias, and inaccuracy in data extraction and synthesis. These threats are described [102] as follows:

**Internal validity:** the threat of incompleteness research lies in the comprehensiveness of the research process and the obtaining studies relevant to the defined research question in this study. The study indicated [103], [104] that the common threat is implementing the appropriate research mechanism by exploring search terms related to the specific topic of the study and identifying the relevant studies to answer the particular research question. Therefore, we tried to maximize internal validity using convenient online libraries involving relevant studies. Thus, seven well-known online libraries were identified, such as Science Direct, Web of Science, Springer, Scopus, Wiley. Additionally, ACM, IEEE libraries are specialized in our domain, including all journal and conferences articles. We also looked for ways to increase internal validity by identifying, researching, and exploring everything related to search terms, synonyms, abbreviations, and alternative words to form an adequate and sufficient research chain from all scientific papers that have been previously identified to reduce research bias as much as possible. Also, the snowballing search strategy was implemented based on the backward and forward concept of the included studies to discover the relevant articles that may be missed in the case of the research chain that did not appear in these articles.

**Construct validity:** this threat lies in a bias of the process and inaccuracy in data extraction and synthesis. We tried to maximize the construct validity to mitigate process bias through a protocol of the study was identified and planned to make sure a mutual understanding, relevant research questions were chosen. In addition, explicit inclusion and exclusion criteria were specified, which may occur when the authors conflict about understanding a process and obstruct reaching common agreements. We also attempted to maximize the construct validity to mitigate inaccuracy in data extraction and synthesis that requires an understanding of the topic to avoid inaccurate information can be introduced, as the threat of data extraction lies in the inference of non-explicit data from detailed data in the included studies. The first author extracted the primary studies' data, then passed them on to the other authors to evaluate. We already identified attributes of the necessary data extraction to gather information and established a checklist to arrange and investigate the needed information based on a discussion among researchers to reduce the gap difference between them and maximize data extraction and synthesis accuracy.

**External validity:** this threat concerns the extent to which a study's findings can be generalized in relation to the mapping study's key objectives based on the papers chosen. By ensuring the representativeness of the included studies, we attempted to maximize results' generalization using the systematic protocol. Furthermore, we organized...
research steps to obtain a comprehensive representation that could be used to view all contexts. As a result, the study findings concerning thorough knowledge of the architectural erosion metrics domain were deemed to meet the validity of all studies on the topic.

VIII. CONCLUSION
In this article, we conducted a systematic mapping study on characterizing metrics of architectural erosion. The key objective was to systematically describe measures that related to the architectural decay to analyze adopted metrics from different perspectives: identifying mechanism of adopted metrics approaches to address the issues of architectural erosion and their classification, describing the mapping metrics to each quality attribute that addressed by researchers in their study by relating metrics to them to handle architectural deterioration, exploring the validation approaches and levels that associated with the nature of the metrics mechanism, using tools that support metrics of the architectural erosion, investigating comparative analysis to identify the effectiveness of the metrics, and approach of the context of the metric, and extent of the applicability of metrics.

We pursued implementing search protocol through seven online libraries (Science direct, web of science, ACM, IEEE, Willey, Scopus, and Springer). Several processes and criteria were applied to reach the studies related to the research question in our research. A total of 43 primary studies were specified to characterize measures of architectural erosion.

The analysis revealed that metrics could be used directly or indirectly to address architectural erosion, defined as the study of change or the development of a phenomenon to determine this deterioration. This led to the classification of architectural decay into several categories, as seen in this study, all of which serve to address this phenomenon.

The analysis also revealed that metrics of architectural erosion have a strong relationship with the quality attributes that identify success or failure in the architecture software. It has also been observed that the architectural erosion metrics are a rapid, vital feedback enabler that is primary in meeting software architecture quality. Also, this study found that measures of the architectural erosion in software quality attributes centralize on maintainability, analyzability, modularity, reusability, and modifiability that are the most attributes addressed by researchers in their study since the strong correlation between these properties and identification of the degradation. Finally, this study presented evidence that architectural erosion metrics provide several research approaches for detecting or predicting architectural erosion problems.

We assert that the findings of SMS could benefit the practitioners and researchers to identify and understand the nature of architectural erosion metrics, as well as their approaches adopted and classification to provide evidence for the future direction of the research, significantly narrow the gap between industry and academia and address the current challenges.

It can be concluded that mechanisms of adopted metrics to address the architectural erosion are still open research issues. Therefore, there is a need for further studies on measures of the architecture decay based on various aspects: new metrics establishment based on the particular set of metrics to adopt a new approach that may enhance from the detection of the architectural degradation, conducting different experiments to ensure applicability and utility of the metrics, and assessing the metrics development that map with the quality factors related to architectural erosion that appeared in this study with a few metrics such as usability, efficiency, and functional suitability.

APPENDIX

APPENDIX A: STUDY QUALITY ASSESSMENT CHECKLIST
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3150847, IEEE Access

**APPENDIX B: METRICS DEFINITIONS**

| Label          | Full name metric                              | Full Definition                                                                 |
|----------------|-----------------------------------------------|---------------------------------------------------------------------------------|
| Abstractness (A) | Abstractness (A)                              | Ratio of abstract classes (and interfaces) to total classes in the analyzed package. |
| ATFD           | Access to Foreign Data                        | The number of external classes from which a given class directly or via accessor methods accesses attributes. |
| Ca             | Afferent Coupling                             | The number of classes from outside package A depends on at least one class from package A. |
| COC            | Architectural Concern Cohesion                | The relative number of methods in a given class implements the same set of concerns. |
| COL            | Architectural Concern Locality                | The relative number of architectural concerns that a code element (class or method) modularizes in its component. |
| AEL            | Architectural Element Locality                | The relative number of dependencies that the measured code element (class or method) has in its component. |
| a2a            | Architecture to Architecture                 | Computes the similarity of two architectures.                                   |
| ASM            | Architecture Stability Metric                | The ratio of the total external calls to the unchanged inter-package calls.        |
| BCF            | Bug change frequency                          | The number of bug-fixing commits in which a file participated.                   |
| BC             | Bug Churn                                     | The number of lines of code added and removed by bug-fixing commits.             |
| Class elegance | Class elegance                                | Calculates the standard deviation for the number of classes in each package.      |
| CCH            | Change Churn                                  | The number of changed LOC in a file committed for any issues.                    |
| CD             | Changes dispersion                            | The number of classes and packages involved in each commit.                     |
| CF             | Change frequency                              | The number of times a file has been checked in.                                 |

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
| CLD       | Comment line density          | Counts comment lines divided by code plus the comment lines and multiply by 100. |
|-----------|-------------------------------|--------------------------------------------------------------------------------|
| CDLOC     | Concern Diffusion in LOC      | The number of transition points through the lines of code for each concern.       |
| CDC       | Concern Diffusion over Components | The number of classes affected by concern implementation.                        |
| CDO       | Concern Diffusion over Operations | The number of methods affected by the concern implementation.                      |
| CC        | Cognitive Complexity          | Measures the current level of comprehension of the software system's various classes. |
| CBO       | Coupling between objects      | The number of other objects to which it is coupled.                               |
| Cvg       | Cluster Coverage              | Calculates the extent to which the clusters of two architectures overlap.         |
| CMC       | Cross-module co-changes       | The number of co-changes for a file in which the co-changes are made across more than one architectural module. |
| DAC       | Data Abstraction Coupling     | A class's number of abstract data types.                                          |
| DP        | Defect persistence            | The number of system development phases (and releases) that this defect spans.     |
| DC        | Defect Complexity             | The number of accompanying changes needed in various components to fix it.         |
| DL        | Decoupling Level              | The degree to which software is decoupled into separate modules.                  |
| DFSM      | Dependency frequency support metric | Investigating frequency in which classes of a module depend on classes of another. |
| Ce        | Efferent Coupling             | The number of classes from outside package A upon which classes reside on package A depends on implementing their features. |
| XS        | Excessive complexity          | The total number of dependencies in the graph divided by the minimal feedback set (MFS) is reported as the percent XS. |
| EWSOC     | Exponentially Weighted Sum of Coupling | Shared transactions are weighted exponentially according to their distance in time, highlighting current modifications rather than those that occurred previously. |
| EMD       | External module dependencies  | The total number of outgoing dependencies from a module to other modules.          |
| CMD       | Incoming module dependency    | The ratio of non-dependency package pairs to all possible package pairs.           |
| IPCI      | Index of Package Changing Impact | A binary metric for a module m1 that returns 1 if at least one module m2 is dependent on m1, and 0 otherwise. |
| IPGF      | Index of Package Goal Focus   | The ratio of non-dependency package pairs to all possible package pairs.           |
| IIPE      | Index of Inter-Package Extending | The average overlap of service sets offered by a component to other components in a software system. |
| IPED      | Index of Inter-Package Extending | The ratio of extended dependencies between classes within a local package to the overall number of Extend dependencies between classes throughout the entire software system. |
| IPU       | Diversion                     | The average diversity of classes extended by a package in other packages.          |
| IIPUD     | Index of Inter-Package Usage  | The ratio of use dependencies between classes within a local package to the total number of use dependents between classes throughout the entire software system. |
| IPMC      | Diversity                    | A package in other packages extends the average diversity of classes.              |
| IPMC      | Inter-Package Modularization  | Measures common reuse of modularization for a package that is reused together.     |
| IPMCC     | Connections                  | Measures the extent of modularization to which cyclic connections among the package are minimized. |
| IPMCD     | Inter-Package Modularization Cyclic Connections | Measures the extent of modularization to minimize cyclic dependencies between the classes. |
| IPMD      | Inter-Package Modularization Dependencies | Measures the common closure of modularization for the classes that change among the packages. |
| IMD       | Internal module dependencies  | The number of co-changes for a file is at least another co-changed file in the same architectural module. |
| Instability (I) | Linearity Weighed Sum of Coupling Line of code | The total number of dependencies among all files within a module. This rating measures the package's adaptability to change. |
| LWSOC     | Instability (I)              | Shared transactions are linearly weighted according to their distance in time.     |
| LOC       | Linearity Weighed Sum of Coupling | Measures the file size determined by counting the number of non-empty non-comment lines. |
| LCOMP     | Lack of Cohesion Metric for Packages | Refers to measure the lack of cohesion of the package. Measures the lack of cohesion in a class according to its common usage of attributes methods. |
| LCOM      | Lack of Cohesion in Methods   | Identifies architectural, technical debt based on Halstead Volume, McCabe Cyclomatic Complexity, Lines of Code, and Comments. |
| MI        | Maintainability Index         | Measures independent paths in software source code using control flow graphs of functions, modules, methods, or classes. |
| MCC       | Maintenance Index             | The total number of transactions in the software system's change history is divided by a total number of transactions involving the code smell I class C. |
| Severity  | McCabe's Cyclomatic complexity | The number of method calls from one class to another. Measures the complete system to simplify understanding and limit modifications to specific system areas. |
| MPC       | Measures change-history of a software system | Measures how strongly the dependency between two modules is using dependency intensity and dependency distribution. |
| MQ        | Message Passing Coupling      | Measures the modularity of social networks represented in graphical structures based on a theoretical heuristic for edges within a module greater than expected ones. |
| MDS       | Modularization Quality        | Measures the modularity of physical entities using a difference of inter and intra... |
| Acronym | Definition |
|---------|------------|
| Mbunch | Modularity Relative Clustered Cost edge densities. Measures the modularity of evolving software systems based on the relative clustered cost of software systems. Measures software modularity that represented intra-edges of modules and inter-edges between modules. |
| MoJo | Modularity Bunch Clustering |
| NAC | Move-Join |
| FAN-OUT | Number of Architectural Concerns Determines the similarity between two different architectures with the same set of implementation-level entities. |
| FAN-IN | Number of called object |
| NC | Number of modules of architectural concerns a given code element (class or method) realizes. The number of modules that the given module depends on. |
| NOCL | Number of changes The number of modules that depend on a given module. |
| NCMs | Number of clusters The number of times a file is committed to a repository. |
| NCONN | Number of co-changed files The number of clusters associated with the package. |
| NCC | Number of commits The number of other files that a given file is changed with. |
| NCM | Number of Connectors The total number of commits made by each developer or contributor. |
| NOD | Number of Concerns per Component The total number of connectors in the architecture. |
| NELEM | The number of dependencies The number of concerns in each class. The number of class-to-class use dependencies relates to the number of links among classes and, indirectly, among packages. The total number of elements in the architecture (summing up the number of components and the number of connectors). |
| NOS | Number of Elements |
| OMD | Number of statements The number of statements in a software component. |
| PCQ | Outgoing module dependency An indicator of whether or not there is at least one dependency between two modules (m1 and m2). |
| PCC | Package Cohesion Quality Measures ratio of internal dependencies of a package to all dependencies among and within a package. |
| PCB | Package Cyclic Connections The ratio of class cyclic dependencies within the package to all package dependencies. |
| PCD | Package Cyclic Dependencies The number of times in which file pairs and versions are modified in the same commit. |
| PClpk | Pair change frequency |
| PDIpk | Project Call Instability of package |
| PCIcl | Project Design Instability of package |
| PDICl | Project Call Instability of class |
| PCIcl | Project Design Instability of class |
| PC | Propagation Cost |
| Public API | Public application programming interface |
| RCI | The ratio of Cohesive Interactions |
| ROC | Ratio of Coupling |
| RFC | The ratio of actual cluster interactions to possible cluster interactions. |
| RBMS | Response For Class |
| SIG | Relationship-Based Similarity Metric |
| SQALE | Software Improvement Group |
| SCC | Software Quality Enhancement |
| SOC | Sum cyclomatic complexity |
| SD | Sum of Coupling |
| Structural distance |
| TF | Ticket frequency |
| TCC | Tight Class Cohesion |
| TCMD | Total incoming module dependencies The number of different bug or issue tracking tickets for which files are modified. The number of different methods is directly connected via accesses of attributes. |
| TOMD | Total outgoing module dependencies |
| TNC | Total Number of Components |
| TWD | Two-way dependencies |
| WMC | Weighted Methods per Class |

The number of dependencies that a class has with other classes. The number of outgoing module dependencies. The number of incoming module dependencies. The number of connections between modules. The number of methods, classes, and attributes that are accessed by a method. The sum of the statistical complexity of all methods in a class.
### APPENDIX C: APPROACHES TO CATEGORIZING ARCHITECTURAL EROSION METRICS

| Classification approach                                      | Definitions                                                                                           |
|-------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| Historical data revision                                   | Refers to analyzing project files to determine how frequently they are bug-prone or change-prone       |
| Architectural bad smell                                     | Refers to an architectural decision that harms system quality and could lead to architectural erosion |
| Architecture modularization                                 | Refers to the degree to which a system or software comprises discrete components with minimal        |
| Architectural change                                        | Refers to the modifications that occur at different levels of abstraction and from multiple            |
| Architectural technical debt                                 | Refers to incomplete, immature, inadequate artifact, or suboptimal designs in the software           |
| Architectural dependency coupling analysis                  | Refers to correlation relationships among abstraction-level elements within a module based on a        |
| Architectural cohesion                                      | Refers to the degree to which the module elements are functionally related and keep the module         |
| Architectural complexity                                    | Refers to the complexity of the system's structure, stored information on how it works, and its        |
| Software architecture size                                  | Refers to estimating the size of a software application or component to identify software              |

### DECLARATION OF COMPETING INTEREST

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

### ACKNOWLEDGMENT

The authors would like to acknowledge Universiti Putra Malaysia (UPM) for facilitating support to get open access publications. The first author would also like to express his gratitude to Hadramout University and the Hadramout Foundation, Yemen, for their support in tuition fees.

### REFERENCES

[1] D. Garlan, "Software architecture: a roadmap," presented at the Proceedings of the Conference on The Future of Software Engineering, Limerick, Ireland, 2000. Available: https://doi.org/10.1145/336512.336537

[2] J. Knodel, M. Lindvall, D. Muthig, and M. Naab, "Static evaluation of software architectures," in Conference on Software Maintenance and Reengineering (CSMR’06), Bari, Italy, 2006, pp. 10 pp.-294: IEEE.

[3] M. M. Lehman, "On understanding laws, evolution, and conservation in the large-program life cycle," Journal of Systems and Software, vol. 1, pp. 213-221, 1979/01/01/ 1979.

[4] M. M. Lehman, "Laws of software evolution revisited," in Proceedings of the 8th European Workshop on Software Process Technology, Berlin, Heidelberg, 1996, pp. 108-124: Springer Berlin Heidelberg.

[5] O. P. N. Syngstad, J. Li, R. Conradi, and M. A. Babar, "Identifying and Understanding Architectural Risks in Software Evolution: An Empirical Study," Berlin, Heidelberg, 2008, pp. 400-414: Springer Berlin Heidelberg.

[6] L. Dobrica and E. Niemela, "A survey on software architecture analysis methods," IEEE Transactions on Software Engineering, vol. 28, no. 7, pp. 638-653, 2002.

[7] A. Baabat, H. B. Zulzalil, S. Hassan, and S. B. Baharom, "Software Architecture Degradation in Open Source Software: A Systematic Literature Review," IEEE Access, vol. 8, pp. 173681-173709, 2020.

[8] A. Baabat, H. B. Zulzalil, S. Hassan, and S. B. Baharom, "A Survey on Characterizing the Empirical Analysis, Proposed Approaches, and Research Trends for Architectural Decay (Accepted for publication)," 2022.

[9] R. Li, P. Liang, M. Soliman, and P. Avergiou, "Understanding Software Architecture Erosion: A Systematic Mapping Study," Journal of Software: Evolution and Process, 12/21 2021.

[10] S. Misra, A. Adewumi, L. Fernandez-Sanz, and R. Damasevicius, "A Suite of Object Oriented Cognitive Complexity Metrics," IEEE Access, vol. 6, pp. 8782-8796, 2018.

[11] S. Misra, "An Approach for the Empirical Validation of Software Complexity Measures," Acta Polytechnica Hungarica, vol. 8, 01/01 2011.

[12] T. DeMarco, Controlling Software Projects: Management, Measurement, and Estimates, Prentice Hall PTR, 1986.

[13] S. Ducasse and D. Pollet, "Software Architecture Reconstruction: A Process-Oriented Taxonomy," IEEE Transactions on Software Engineering, vol. 35, no. 4, pp. 573-591, 2009.

[14] M. Shaw and P. Clements, "The golden age of software architecture," IEEE Software, vol. 23, no. 2, pp. 31-39, 2006.

[15] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice. Addison-Wesley Professional, 2012.

[16] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," vol. 17, no. 4, 1992.

[17] M. Shaw and D. Garlan, Software architecture: perspectives on an emerging discipline. Prentice-Hall, Inc., 1996.
[18] M. H. Klein, R. Kazman, L. Bass, J. Carriere, M. Barbacci, and H. Lipson, "Attribute-Based Architecture Styles," in Software Architecture: TC2 First Working IFIP Conference on Software Architecture (WICSA'99), 22–24 February 1999, San Antonio, Texas, USA, P. Donohoe, Ed. Boston, MA: Springer US, 1999, pp. 225-243.

[19] A. Jensen, J. v. d. Ven, P. Avgeriou, and D. K. Hammer, "Tool Support for Architectural Decisions," in 2007 Working IEEE/IFIP Conference on Software Architecture (WICSA'07), Mumbai, India, 2007, pp. 4-4: IEEE.

[20] D. L. Pamas, "Software aging," presented at the 16th International conference Software Engineering, Sorrento, Italy, 1994.

[21] L. Hochstein and M. Lindvall, "Combating architectural degeneration: a survey," Inf. Softw. Technol., vol. 47, no. 10, pp. 643–656, 2005.

[22] J. Bosch, "Software Architecture: The Next Step," in Software Architecture, First European Workshop, Berlin, Heidelberg, 2004, vol. 3047, pp. 194-199: Springer Berlin Heidelberg.

[23] M. Riaz, M. Sulayman, and H. Naqvi, "Architectural Decay during Continuous Software Evolution and Impact of "Design for Change" on Software Architecture," in International Conference on Advanced Software Engineering and Its Applications, Berlin, Heidelberg, 2009, pp. 119-126: Springer Berlin Heidelberg.

[24] M. Lindvall, R. Tesoriero, and P. Costa, "Avoiding architectural degeneration: an evaluation process for software architecture," in Proceedings Eighth IEEE Symposium on Software Metrics, Ottawa, Ontario, Canada, 2002, pp. 77-86: IEEE.

[25] N. Medvidovic and R. N. Taylor, "Software architecture: foundations, theory, and practice," in 2010 ACM/IEEE 32nd International Conference on Software Engineering, 2010, vol. 2, pp. 471-472.

[26] J. Lenhard, M. Blom, and S. Herold, "Exploring the suitability of source code metrics for indicating architectural inconsistencies," Software Quality Journal, Article vol. 27, no. 1, pp. 241-274, 2019.

[27] V. Bandara and I. Perera, "Identifying Software Architecture Erosion Through Code Comments," in 2018 18th International Conference on Advances in ICT for Emerging Regions (ICTer), 2018, pp. 62-69.

[28] Z. Li and J. Long, "A Case Study of Measuring Degeneration of Software Architectures from a Defect Perspective," in 2011 18th Asia-Pacific Software Engineering Conference, 2011, pp. 242-249.

[29] H. P. Breivold and I. Crnkovic, "A Systematic Review on Architected for Software Evolvability," presented at the Proceedings of the 2010 21st Australian Software Engineering Conference, Auckland, New Zealand, 2010. Available: https://doi.org/10.1109/ASWEC.2010.11

[30] W. J. Dradek, E. Arisholm, and L. C. Briand, "A Realistic Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance," IEEE Transactions on Software Engineering, vol. 34, no. 3, pp. 407-432, 2008.

[31] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: a survey," IEEE Transactions on Software Engineering, vol. 30, no. 5, pp. 295-310, 2004.

[32] F. Brosig, P. Meier, S. Becker, A. Koziolek, H. Koziolek, and S. Kounev, "Quantitative Evaluation of Model-Driven Performance Analysis and Simulation of Component-Based Architectures," IEEE Transactions on Software Engineering, vol. 41, no. 2, pp. 157-175, 2015.

[33] H. Koziolek, B. Schlicht, S. Becker, and M. Hauck, "Performance and reliability prediction for evolving service-oriented software systems," vol. 18, no. 4: 5 Empirical Softw. Engng., pp. 746-790, 2013.

[34] M. Berry and M. F. Vandenbroek, "A Targeted Assessment of the Software Measurement Process," presented at the Proceedings of the 7th International Symposium on Software Metrics, London, UK, 2001.

[35] L. C. Briand, C. M. Differding, and H. D. Rombach, "Practical guidelines for measurement-based process improvement," Softw. Process, vol. 2, no. 4, pp. 253-280, 1996.

[36] M. Unterkalmsteiner, T. Gorschek, A. K. M. M. Islam, C. K. Cheng, R. B. Permaid, and R. Feldt, "Evaluation and Measurement of Software Process Improvement—A Systematic Literature Review," IEEE Transactions on Software Engineering, vol. 38, no. 2, pp. 398-424, 2012.

[37] S. Misra, I. Akman, and R. Colombo-Palacios, "Framework for evaluation and validation of software complexity measures," Software, IET, vol. 6, pp. 323-334, 08/01 2012.

[38] "ISO/IEC/IEEE International Standard - Systems and software engineering—Measurement process," ISO/IEC/IEEE 15939:2017(E), pp. 1-49, 2017.

[39] R. Malhotra, Empirical Research in Software Engineering: Concepts, Analysis, and Applications. Chapman & Hall/CRC, 2015.

[40] P. Goodman, The Practical Implementation of Software Metrics. McGraw-Hill, Inc., 1993.

[41] M. Abdelali, T. A. M. Sultan, A. A. G. Ani, and M. A. Jaber, "A mapping study to investigate component-based software system metrics," Journal of Systems and Software, vol. 86, no. 3, pp. 587-603, 2013/03/01 2013.

[42] M. Staron and W. Meding, "Metrics for Software Design and Architectures," in Automotive Software Architectures: An Introduction, M. Staron, Ed. Cham: Springer International Publishing, 2017, pp. 179-199.

[43] T. Tahir, G. Rasool, and C. Gencel, "A systematic literature review on software measurement programs," Information and Software Technology, vol. 73, pp. 101-121, 2016/05/01 2016.

[44] T. Coulin, M. Detante, W. Mouchère, and F. J. A. Petrillo, "Software Architecture Metrics: A literature review," vol. abs/1901.09050, 2019.

[45] S. Stevanetic and U. Zdun, "Software metrics for measuring the understandability of architectural structures: a systematic mapping study," presented at the Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering, Nanjing, China, 2015. Available: https://doi.org/10.1145/2745802.2745822

[46] M. Alenzi, "Software Architecture Quality Measurement Stability and Understandability," International Journal of Advanced Computer Science and Applications(IJACSA), vol. 7, no. 3, pp. 1-10, 2016.

[47] A. S. Nuñez-Varela, H. G. Pérez-Gonzalez, F. E. Martínez-Perez, and C. Soubberviere-Montalvo, "Source code metrics: A systematic mapping study," Journal of Systems and Software, vol. 128, pp. 164-197, 2017/06/01 2017.

[48] H. Koziolek, "Sustainability evaluation of software architecture: a systematic review," presented at the Proceedings of the joint ACM SIGSOFT conference – QoS and ACM SIGSOFT Symposium -- ISARCS on Quality of software architectures -- QoSA and architecting critical systems -- ISARCS, Boulder, Colorado, USA, 2011. Available: https://doi.org/10.1145/2000259.2000263

[49] B. A. Kitchenham, D. Budgen, and O. Pearl Brereton, "Using mapping studies as the basis for further research – A participant-observer case study," Information and Software Technology, vol. 53, no. 6, pp. 638-651, 2011/06/01 2011.

[50] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," presented at the Proceedings of the 12th international conference on Evaluation and Assessment in Software Engineering, Italy, 2008.

[51] B. Kitchenham and S. Charters, "Guidelines for conducting systematic mapping studies in software engineering: An update," Information and Software Technology, vol. 64, pp. 1-18, 2015/08/01 2015.

[52] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Technical report, Ver. 2.3 EBSE Technical Report. EBSE2007.

[53] B. Badampudi, C. Wohlin, and K. Petersen, "Experiences from using snowballing and database searches in systematic
literature studies,” presented at the Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering, Nanjing, China, 2015. Available: https://doi.org/10.1145/2745802.2745818

[54] E. Mourão, M. Kalinowski, L. Murta, E. Mendes, and C. Wohlin, “Investigating the Use of a Hybrid Search Strategy for Systematic Reviews,” in 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2017, pp. 193-198.

[55] J. Webster and R. T. Watson, “Analyzing the past to prepare for the future: writing a literature review,” vol. 26, no. 2 IJ MIS Q., pp. xiii–xiiiıı, 2002.

[56] O. Dieste and A. G. Padua, “Developing Search Strategies for Detecting Relevant Experiments for Systematic Reviews,” in First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), Madrid, Spain, 2007, pp. 215-224: IEEE.

[57] L. Aversano, D. Guardabascio, and M. Tortorella, "An Empirical Study on the Architecture Instability of Software Projects," vol. 29, no. 04, pp. 515-545, 2019.

[58] A. Sejfa, "A pilot study on architecture and vulnerabilities: Lessons learned," in Proceedings - 2019 IEEE/ACM 2nd International Workshop on Establishing a Community-Wide Infrastructure for Architecture-Based Software Engineering, ECASE 2019, Montreal, QC, Canada, 2019, pp. 42-47: IEEE.

[59] R. Mo, Y. Cai, R. Kazman, L. Xiao, and Q. Feng, “Architecture Anti-patterns: Automatically Detectable Violations of Design Principles,” IEEE Transactions on Software Engineering, pp. 1-1, 2019.

[60] S. G. Maisikeli, "Measuring Architectural Stability and Instability in the Evolution of Software Systems," in IFT 2018 - Information Technology Trends: Emerging Technologies for Artificial Intelligence, Dubai, United Arab Emirates, 2019, pp. 263-275: IEEE.

[61] L. P. d. S. Carvalho, R. Novais, and M. Mendonça, "Investigating the Relationship between Code Smell Agglomerations and Architectural Concerns: Similarities and Dissimilarities from Distributed, Service-Oriented, and Mobile Systems," presented at the Proceedings of the VII Brazilian Symposium on Software Components, Architectures, and Reuse, Sao Carlos, Brazil, 2018. Available: https://doi.org/10.1145/3267183.3267184

[62] A. Shahbazian, D. Nam, and N. Medvidovic, "Toward predicting architectural significance of implementation issues," presented at the Proceedings of the 15th International Conference on Mining Software Repositories, Gothenburg, Sweden, 2018. Available: https://doi.org/10.1145/3196398.3196440

[63] P. Behnamghader, D. M. Le, J. Garcia, D. Link, A. Shahbazian, and N. Medvidovic, "A large-scale study of architectural evolution in open-source software systems," Empirical Software Engineering, Article vol. 22, no. 3, pp. 1146-1213, 2017.

[64] S. Mohsin, H. Akhtar, Jalbani., A. Adil, A. Ahmed, and M. Kashif, "Evaluating Dependency based Package-level Metrics for Multi-objective Maintenance Tasks," International Journal of Advanced Computer Science and Applications(IJACSA), vol. 8, no. 10, 2017.

[65] H. Rocha, R. S. Durelli, R. Terra, S. Bessa, and M. T. Valente, "DCL 2.0: modular and reusable specification of architectural constraints," Journal of the Brazilian Computer Society, vol. 23, no. 1, p. 12, 2017:08/16 2017.

[66] F. A. Fontana, R. Roveda, M. Zanoni, C. Raibulet, and R. Capilla, "An Experience Report on Detecting and Repairing Software Architecture Erosion," in 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA), 2016, pp. 21-30.

[67] R. Mo, Y. Cai, R. Kazman, L. Xiao, and Q. Feng, "Decoupling level: a new metric for architectural maintenance complexity," presented at the Proceedings of the 38th International Conference on Software Engineering, Austin, Texas, 2016. Available: https://doi.org/10.1145/2884871.2884825

[68] M. De Oliveira Barros, F. De Almeida Farzat, and G. H. Travassos, “Learning from optimization: A case study with apache ant,” Information and Software Technology, Conference Paper vol. 57, no. 1, pp. 684-704, 2015.

[69] E. Guimarães, A. Garcia, and Y. Cai, "Architecture-sensitive heuristics for prioritizing critical code anomalies," presented at the Proceedings of the 14th International Conference on Modularity, Fort Collins, CO, USA, 2015. Available: https://doi.org/10.1145/2724525.2724567

[70] F. A. Fontana, V. Ferme, and M. Zanoni, "Towards assessing software architecture quality by exploiting code smell relationships," presented at the Proceedings of the Second International Workshop on Software Architecture and Metrics, Florence, Italy, 2015.

[71] Z. Li, P. Liang, P. Avgeriou, N. Guelci, and A. Ampatzoglou, "An empirical investigation of modularity metrics for indicating architectural technical debt," presented at the Proceedings of the 10th international ACM Sigsoft conference on Quality of software architectures, Marçq-en-Barœul, France, 2014. Available: https://doi.org/10.1145/2602576.2602581

[72] M. Ferreira, E. Barbosa, I. Macia, R. Arcoverde, and A. Garcia, "Detecting architecturally-relevant code anomalies: a case study of effectiveness and effort," presented at the Proceedings of the 29th Annual ACM Symposium on Applied Computing, Gyeongju, Republic of Korea, 2014. Available: https://doi.org/10.1145/2554850.2555036

[73] I. Macia, A. Garcia, C. Chavez, and A. Von Staa, "Enhancing the detection of code anomalies with architecture-sensitive strategies," in Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR, Genova, Italy, 2013, pp. 177-186: IEEE.

[74] E. Guimarães, A. Garcia, E. Figueiredo, and Y. Cai, "Prioritizing software anomalies with software metrics and architecture blueprints: a controlled experiment," presented at the Proceedings of the 5th International Workshop on Modeling in Software Engineering, San Francisco, California, 2013.

[75] I. Macia, J. Garcia, D. Popescu, A. Garcia, N. Medvidovic, and A. v. Staa, "Are automatically-detected code anomalies relevant to architectural modularity? an exploratory analysis of evolving systems," presented at the Proceedings of the 11th annual international conference on Aspect-oriented Software Development, Potsdam, Germany, 2012. Available: https://doi.org/10.1145/2162049.2162069

[76] Z. Li et al., "Characteristics of multiple-component defects and architectural hotspots: a large scale study," Empirical Software Engineering, vol. 16, no. 5, pp. 667-702, 2011:10/01 2011.

[77] M. Steff and B. Russo, "Measuring architectural change for defect estimation and localization," in International Symposium on Empirical Software Engineering and Measurement, anfl, AB, Canada, 2011, pp. 225-234: IEEE.

[78] R. S. Sangwan, P. Vercellone-Smith, and C. J. Neill, "Use of a multidimensional approach to study the evolution of software complexity," Innovations in Systems and Software Engineering, Article vol. 6, no. 4, pp. 299-310, 2010.

[79] A. Capiluppi and T. Knowles, "Software Engineering in Practice: Design and Architectures of FLOSS Systems," in Proceedings: The International Conference on Software Maintenance and Reengineering, ICSEIP.2019.00026

[80] M. Nayebi et al., "A longitudinal study of identifying and paying down architecture debt," presented at the Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, Montreal, Quebec, Canada, 2019. Available: https://doi.org/10.1109/ICSE-SEIP.2019.00026

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3150847, IEEE Access
[82] V. Zapalowski, I. Nunes, and D. J. Nunes, “The WGB method to recover implemented architectural rules,” Information and Software Technology, vol. 103, pp. 125-137, 2018/11/01/ 2018.

[83] V. Zapalowski, D. J. Nunes, and I. Nunes, “Understanding architecture non-conformance: why is there a gap between conceptual architectural rules and source code dependencies?”, presented at the Proceedings of the XXXII Brazilian Symposium on Software Engineering, Sao Carlos, Brazil, 2018. Available: https://doi.org/10.1145/3266237.3266261.

[84] R. Roveda, F. A. Fontana, I. Pigazzini, and M. Zanoni, “Towards an architectural debt index,” in Proceedings - 44th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2018, Prague, Czech Republic, 2018, pp. 408-416: IEEE.

[85] L. Rizzi, F. A. Fontana, and R. Roveda, “Support for architectural smell refactoring,” presented at the Proceedings of the 2nd International Workshop on Refactoring, Montpellier, France, 2018. Available: https://doi.org/10.1145/3242163.3242165.

[86] A. Biaggi, F. A. Fontana, and R. Roveda, “An architectural smells detection tool for C and C++ projects,” in Proceedings - 44th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2018, Prague, Czech Republic, 2018, pp. 417-420: IEEE.

[87] F. A. Fontana, I. Pigazzini, R. Roveda, and M. Zanoni, “Automatic detection of instability architectural smells,” in Proceedings - 2016 IEEE International Conference on Software Maintenance and Evolution, ICSEME 2016, Raleigh, NC, USA, 2016, pp. 433-437: IEEE.

[88] M. Altunışık, A. Ersoy, and H. Sözer, “Evaluating software architecture erosion for PL/SQL programs,” presented at the Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings, Canterbury, United Kingdom, 2017. Available: https://doi.org/10.1145/3129790.3129811.

[89] F. A. Fontana, R. Roveda, S. Vittori, A. Metelli, S. Saldarini, and F. Mazzei, “On evaluating the impact of the refactoring of architectural problems on software quality,” presented at the Proceedings of the Scientific Workshop Proceedings of XP2016, Edinburgh, Scotland, UK, 2016. Available: https://doi.org/10.1145/2962695.2962716.

[90] S. Stevanovic, T. Haitzer, and U. Zdun, “Supporting Software Evolution by Integrating DSL-based Architectural Abstraction and Understandability Related Metrics,” presented at the Proceedings of the 2014 European Conference on Software Architecture Workshops, Vienna, Austria, 2014. Available: https://doi.org/10.1145/2642803.2642822.

[91] D. Reimanis, C. Izurieta, R. Luhr, L. Xiao, Y. Cai, and G. Rudy, “A replication case study to measure the architectural quality of a commercial system,” presented at the Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Torino, Italy, 2014. Available: https://doi.org/10.1145/2652524.2652581.

[92] E. Guimaraes, A. Garcia, and Y. Cai, “Exploring Blueprints on the Prioritization of Architecturally Relevant Code Anomalies – A Controlled Experiment,” in 2014 IEEE 38th Annual Computer Software and Applications Conference, Vasteras, Sweden, 2014, pp. 344-353: IEEE.

[93] R. Schwanke, L. Xiao, and Y. Cai, “Measuring architecture quality by structure plus history analysis,” presented at the Proceedings of the 2013 International Conference on Software Engineering, San Francisco, CA, USA, 2013.

[94] M. D. Ambros, M. Lanza, and R. Robbes, “On the Relationship Between Change Coupling and Software Defects,” in 2009 16th Working Conference on Reverse Engineering, Lille, France, 2009, pp. 135-144: IEEE.

[95] A. MacCormack and D. J. Sturtevant, “Technical debt and system architecture: The impact of coupling on defect-related activity,” Journal of Systems and Software, Article vol. 120, pp. 170-182, 2016.

[96] J. A. Díaz-Pace, A. Tommasel, I. Pigazzini, and F. A. Fontana, “Sen4Smells: A Tool for Ranking Sensitive Smells for an Architecture Debt Index,” in 2020 IEEE Congreso Bienal de Argentina (ARGENCON), Resistencia, Argentina, 2020, pp. 1-7: IEEE.

[97] J. Garcia, E. Kouroshfar, N. Ghorbani, and S. Malek, “Forecasting Architectural Decay from Evolutionary History,” IEEE Transactions on Software Engineering, pp. 1-1, 2021.

[98] P. W. Lawrence JD, Sicherman A, Johnson GL, “Assessment of software reliability measurement methods for use in probabilistic risk assessment,” in Fission Energy and Systems Safety Program,” Lawrence Livermore National LaboratoryTechnical Report UCRLID-136035, 1988.

[99] K. P. Srinivasan and D. Thirupathi, “Software Metrics Validation Methodologies in Software Engineering,” International Journal of Software Engineering & Applications, vol. 5, no. 6, pp. 87-112, 11/30 2014.

[100] B. Kitchenham, S. L. Pfleeger, and N. Fenton, “Towards a framework for software measurement validation,” IEEE Transactions on Software Engineering, vol. 21, no. 12, pp. 929-944, 1995.

[101] M. A. Babar and I. Gorton, "Software Architecture Review: The State of Practice," vol. 42, no. 7 9/1 Computer, pp. 26–32, 2009.

[102] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wesslin, Experimentation in Software Engineering, Springer Publishing Company, Incorporated, 2012.

[103] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Technical report. Ver. 2.3 EBSE Technical Report. EBSE2007.

[104] A. Nguyen-Phuoc, D. S. Cruzes, and R. Conradi, “The impact of global dispersion on coordination, team performance and software quality – A systematic literature review,” Information and Software Technology, vol. 57, pp. 277-294, 2015/01/01/ 2015.