Received February 2, 2022, accepted February 8, 2022, date of publication February 10, 2022, date of current version March 4, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3150847

Characterizing the Architectural Erosion Metrics: A Systematic Mapping Study

AHMED BAABAD[1,2], HAZURA BINTI ZULZALIL[1], SA'ADAH HASSAN[1], AND SALMI BINTI BAHAROM[1]

[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, Al 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] to combat architectural erosion. Consequently, the metrics strategy is the most common and successful among the available solutions [7], [8]. 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[9]. The basic logic originates from this concept " you cannot control what you cannot measure" [10].

Despite the vast number of measures and their distribution across different topics in software engineering, there

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana.

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]. 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], [7]. 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], [11], [12].

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. [13]. SA is concerned with the high-level structure and system attributes [14], [15]. It involves and interacts with software families studying component-based reuse, limited classes of components, domain-particular design, and software analysis [12]. 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 [16].

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 [17]–[20], architectural decay [21], architectural degeneration [19], [22], or architectural degradation [14], [23], [24].

Architectural erosion can be defined as a continuous divergence between prescriptive and descriptive software architecture as intended and implemented. [14], [23]. 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 [22], developer mistakes, bad practices [25], 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 [25], [26].

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 [27], [28], minimizing software performance [29], [30], and declining software quality properties [31] like maintainability, adaptability, or reusability because of an erosion factor contributing to software aging.

C. 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 [9, 32-35]. 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 [36], 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 [37] since it's an essential task to a process of software measurement and quality attributes. Goodman [38] 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.

Baabad *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. Abdellatief *et al.* [39] 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 [40] 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 *et al.* [41] conducted a systematic literature review (SLR) to provide a comprehensive overview of software measurement programs (MPs). They highlighted used tools and current measurement planning models for carrying out MPs and mitigation strategies to encounter the challenges and essential information on success/failure factors of MPs. Coulin *et al.* [42] performed 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 [43] 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. Mamdouh Alenezi [44] 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 *et al.* [45] 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 [46] conducted a systematic literature review on measuring the sustainability of software architecture. He evaluated sustainability through evolvement 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.

TABLE 1. Review summary of prior studies literature.

Reference	Focus	Period-span	Primary paper	Review method	Domain
Baabad, Zulzalil 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 [39]	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 [40]	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. [41]	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 [42]	Analysis of existing architecture metrics for evaluating the quality.	Up to 2019	56	SLR	Architecture metrics
Stevanetic and Zdun [43]	To measure the understandability of architectural structure concepts of high-level architecture.	1990- June 2013	25	SMR	Understandability of architectural structure
Mamdouh Alenezi [44]	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 [45]	To collect source code metrics and analyze the current state and their current trends.	2010- 2015	226	SMR	Source code metrics
Koziolek [46]	To measure the sustainability of a software architecture using scenarios and metrics	Up to 2011	-	SLR	Sustainability of software architecture

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.

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 [47]–[49]. 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.* [48] 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.* [49]. Fig 1 depicts a flowchart of the mapping process followed by us, confirmed through guidelines and instructions by [48-50].

A. PLANNING PROCESS

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

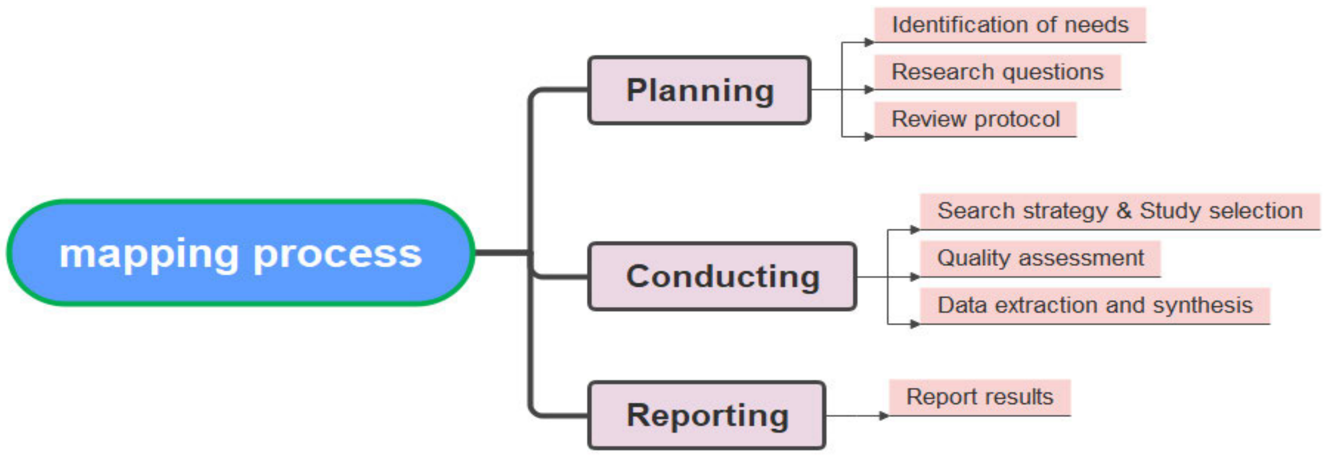

FIGURE 1. The systematic mapping protocol process.

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.* [49] and Kitchenham and Charters [50] 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.

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 [51, 52]. 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.* [49] and Kitchenham and Charters [50], 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:

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 in the context of architectural erosion?	Applied metrics can contribute to Identifying the extent of ease and difficulty relating to applicability.

- 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.

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. [53]. Finally, the Google Scholar engine was also employed to define related studies citations based on the selected primary studies.

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,

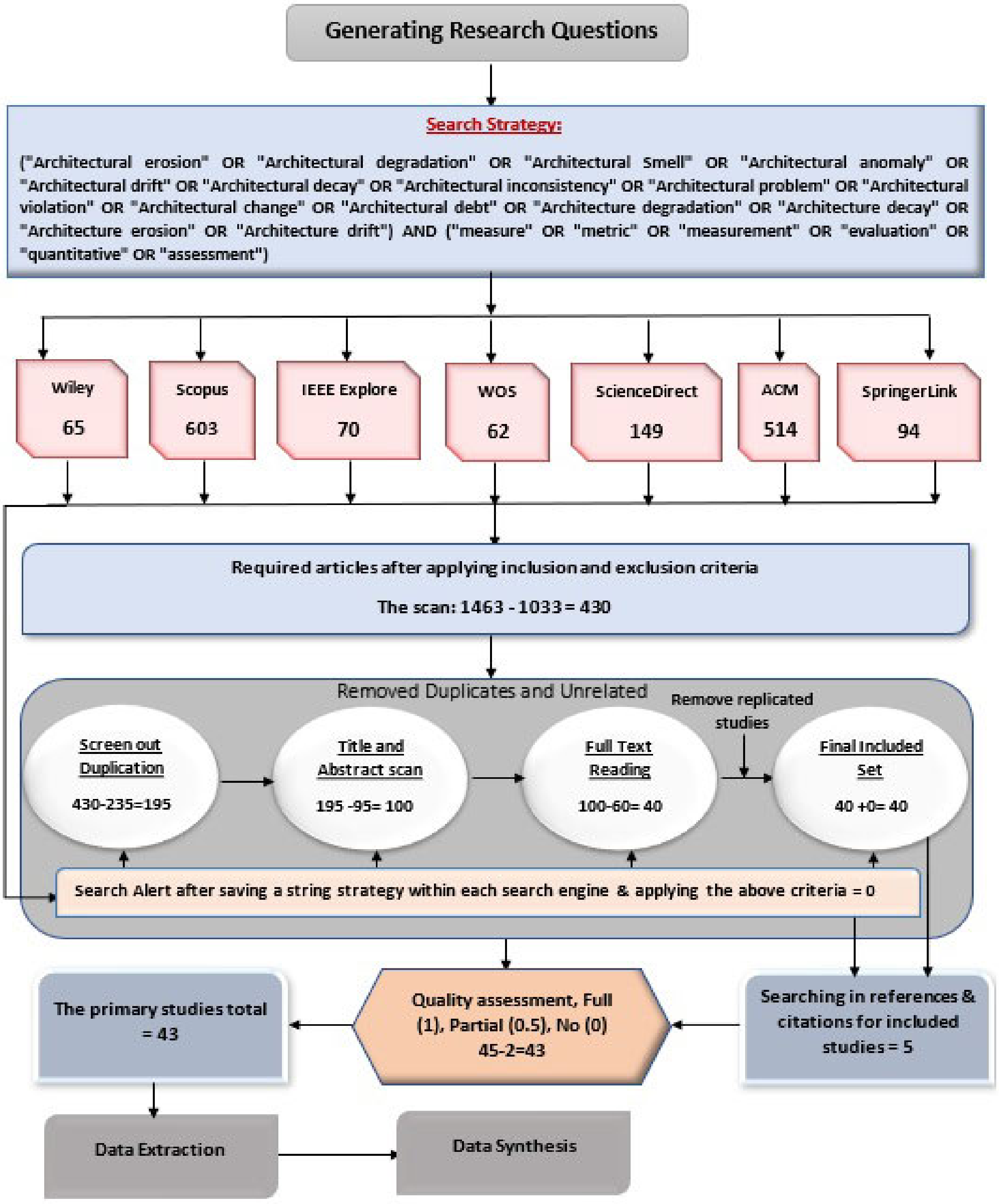
Generating Research Questions
Search Strategy:
("Architectural erosion" OR "Architectural degradation" OR "Architectural Smell" OR "Architectural anomaly" OR "Architectural drift" OR "Architectural decay" OR "Architectural inconsistency" OR "Architectural problem" OR "Architectural violation" OR "Architectural 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
Removed Duplicates and Unrelated
Remove replicated studies
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
Searching in references & citations for included studies = 5
Data Extraction
Data Synthesis

FIGURE 2. Review protocol stages.

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,

TABLE 3. Online databases.

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/

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 [50], Petersen *et al.* [49], and developing search strategies including some recommendations to find out the related studies [53], [54].

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 [52]. 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 [49], [50], 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,

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 [55]	An Empirical Study on the Architecture Instability of Software Projects.	2019	Journal
S02	Lenhard et al. [24]	Exploring the suitability of source code metrics for indicating architectural inconsistencies	2019	Journal
S03	Sejfia [56]	A Pilot Study on Architecture and Vulnerabilities: Lessons Learned.	2019	Conference
S04	Mo et al. [57]	Architecture Anti-patterns: Automatically Detectable Violations of Design Principles.	2019	Journal
S05	Maisikeli [58]	Measuring Architectural Stability and Instability in the Evolution of Software Systems.	2019	Conference
S06	Carvalho et al. [59]	Investigating the Relationship between Code Smell Agglomerations and Architectural Concerns.	2018	Conference
S07	Shahbazian et al. [60]	Toward Predicting Architectural Significance of Implementation Issues.	2018	Conference
S08	Behnamghader et al. [61]	A large-scale study of architectural evolution in open-source software systems.	2017	Journal
S09	Mohsin et al. [62]	Evaluating Dependency based Package-level Metrics for Multi-objective Maintenance Tasks.	2017	Journal
S10	Henrique et al. [63]	DCL 2.0: modular and reusable specification of architectural constraints.	2017	Journal
S11	Fontana et al. [64]	An Experience Report on Detecting and Repairing Software Architecture Erosion.	2016	Conference
S12	Mo et al. [65]	Decoupling Level: A New Metric for Architectural Maintenance Complexity.	2016	Conference
S13	De Oliveira Barros et al [66]	Learning from optimization: A case study with Apache Ant.	2015	Journal
S14	Guimarães et al. [67]	Architecture-Sensitive Heuristics for Prioritizing Critical Code Anomalies.	2015	Conference
S15	Fontana et al. [68]	Towards assessing software architecture quality by exploiting code smell relations.	2015	Conference
S16	Zengyang et al. [69]	An Empirical Investigation of Modularity Metrics for Indicating Architectural Technical Debt.	2014	Conference
S17	Ferreira et al. [70]	Detecting Architecturally-Relevant Code Anomalies: A Case Study of Effectiveness and Effort.	2014	Conference
S18	Macia et al. [71]	Enhancing the Detection of Code Anomalies with Architecture-Sensitive Strategies.	2013	Conference
S19	Guimaraes et al. [72]	Prioritizing Software Anomalies with Software Metrics and Architecture Blueprints.	2013	Conference
S20	Macia et al. [73]	Are Automatically-Detected Code Anomalies Relevant to Architectural Modularity? An Exploratory Analysis of Evolving Systems.	2012	Conference
S21	Zude et al. [74]	Characteristics of multiple-component defects and architectural hotspots a large system case study.	2011	Journal
S22	Steff and Russo [75]	Measuring Architectural Change for Defect Estimation and Localization.	2011	Conference
S23	Sangwan et al. [76]	Use of a multidimensional approach to study the evolution of software complexity.	2010	Journal
S24	Andrea and Thomas [77]	Software Engineering in Practice: Design and Architectures of FLOSS Systems.	2009	Conference
S25	Singh et al. [78]	Reducing Maintenance Efforts of Developers by Prioritizing Different Code Smells.	2019	Journal
S26	Nayebi et al. [79]	A Longitudinal Study of Identifying and Paying Down architecture debt.	2019	Conference
S27	Vanius et al. [80]	The WGB method to recover implemented architectural rules.	2018	Journal
S28	Zapalowski et al. [81]	Understanding architecture non-conformance: Why is there a gap between conceptual architectural rules and source code dependencies?	2018	Conference
S29	Roveda et al. [82]	Towards an Architectural Debt Index.	2018	Conference
S30	Rizzi et al [83]	Support for Architectural Smell Refactoring.	2018	Conference
S31	Biaggi et al [84]	An Architectural Smells Detection Tool for C and C++ projects.	2018	Conference
S32	Fontana et al. [85]	Automatic Detection of Instability Architectural Smells.	2017	Conference
S33	Altınışık et al. [86]	Evaluating Software Architecture Erosion for PL/SQL Programs.	2017	Conference
S34	Fontana et al. [87]	On evaluating the impact of the refactoring of architectural problems on software quality.	2016	Conference
S35	Stevanetic et al. [88]	Supporting Software Evolution by Integrating DSL-based Architectural Abstraction and Understandability Related Metrics.	2014	Conference
S36	Reimanis et al. [89]	A Replication Case Study to Measure the Architectural Quality of a Commercial System.	2014	Conference
S37	Guimaraes et al. [90]	Exploring Blueprints on the Prioritization of Architecturally Relevant Code Anomalies: A Controlled Experiment.	2014	Conference

TABLE 5. *(Continued.)* **Details of primary studies.**

S38	Schwanke et al. [91]	Measuring Architecture Quality by Structure Plus History Analysis.	2013	Conference
S39	Ambros et al. [92]	On the Relationship Between Change Coupling and Software Defects.	2009	Conference
S40	MacCormack and Sturtevant [93]	Technical debt and system architecture: The impact of coupling on defect-related activity.	2016	Journal
S41	Diaz-Pace et al. [94]	Sen4Smells: A Tool for Ranking Sensitive Smells for an Architecture Debt Index	2020	Conference
S42	Garcia et al. [95]	Forecasting Architectural Decay from Evolutionary History	2021	Journal
S43	Lindvall et al. [22]	Avoiding Architectural Degeneration: An Evaluation Process for Software Architecture	2002	Conference

TABLE 6. **Quality assessment criteria.**

QA ID	Quality checklist questions	Marked Score
QA 1	Are the goal/(s) of the research clearly declared?	The score "Yes" =1 / "No" = 0 / "partial" = 0.5
QA 2	Does paper add well-motivated value about characterizing the metrics for architecture erosion?	
QA 3	Does the paper provide in-depth detail of the architectural erosion metrics?	
QA 4	Is the metrics assessment explicitly described?	
QA 5	Is the paper well-referenced (i.e., article references from various journals and peer-reviewed conferences)?	
QA 6	Does the paper independently depend on metrics of architectural erosion?	

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.

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.

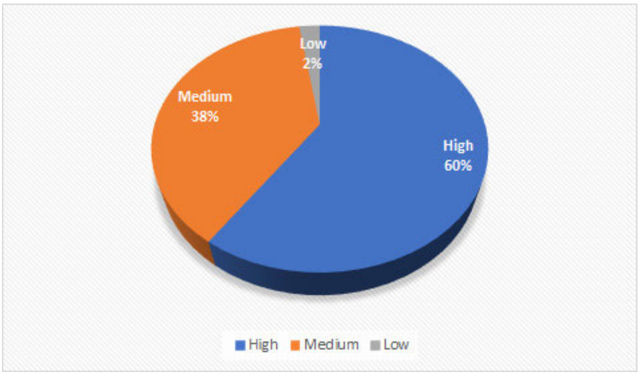

FIGURE 3. **Distribution of study quality assessment levels.**

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 [49], [50] 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

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?

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).

V. 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, Architecture modularization, 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

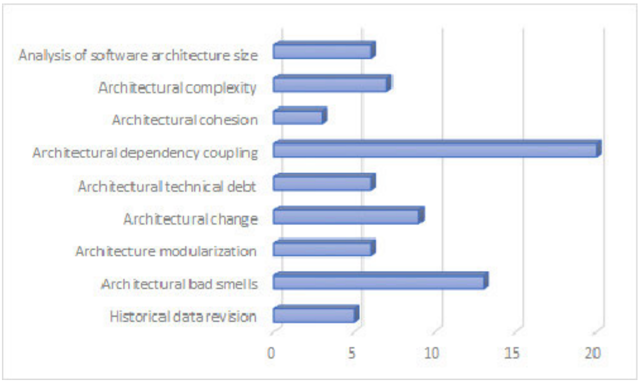

FIGURE 4. Number of metrics by classification.

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%.

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, COC.
[S03], [S06], [S09], [S12], [S16], [S26].	Architecture modularization	NOCL, IPMD, IPMC, IPMCD, IPMCC, Mnewm, Mg&g, 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, IIPE, IPCI, IIPUD, IIPED, 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, LWSOC, RFC, ROC, Ca, Ce, MPC, 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, DC, DP.
[S02], [S05], [S10], [S13], [S35], [S37].	Analysis of software architecture size	LOC, TNC, NCONN, NELEM, NCML, CLD, NOS, Public API, Class elegance.

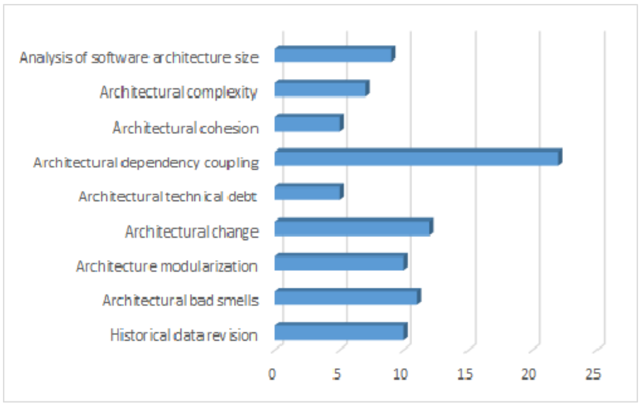

FIGURE 5. **Number of papers by classification.**

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 criteria of metrics mapping based on two identified key criteria [96]: 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) 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

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, PCIpk, PDIcl, PCIcl, PkDI, PkCI, 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, PCIpk, PDIcl, PCIcl, PkDI, PkCI, 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, BF, PCF, NAC, COL, COC. IPMD, IPMC, IPMCD, PCD, CDLOC, IPMCC, Mnewm, Mg&g, Mrcc, Mbunch, MQ, DL, PC, PCQ, LCOMP, IIPU, IIPE, IPCI, 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, TOMD, 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, PkCI, PCIpk, PDIpk, PCIcl, PDIcl, 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, IMC, RBMS, ASM, MoJo, CD, SD, SIG, SQALE, CMD, OMD, TOMD, IMD, EMD, PCQ, LCOMP, RCI, SCC, DC, DP, NELEM, IIPU, IIPE, IPCI, IIPUD, IIPED, 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	3
WMC.	5
NOC, LCOM, Ca, Ce.	6
CBO.	7
FAN-IN.	9
LOC.	10
FAN-OUT.	11

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.

1) 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 [97], [98].

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.

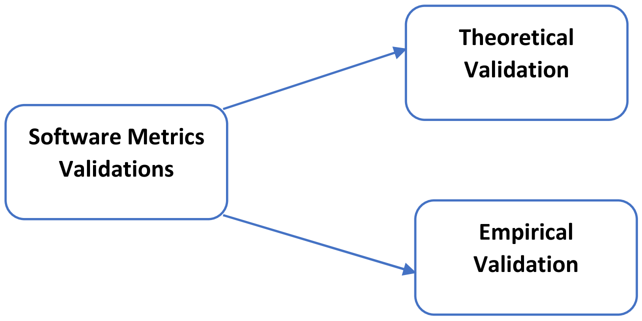

FIGURE 6. Depicts the two approaches of software metrics validations.

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 [43], [46].

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, IIPE, 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]	Empirical validation
[S06], [S09], [S10], [S14], [S17], [S20], [S27], [S29], [S31].	Complete validation

2) 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 [43].

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.

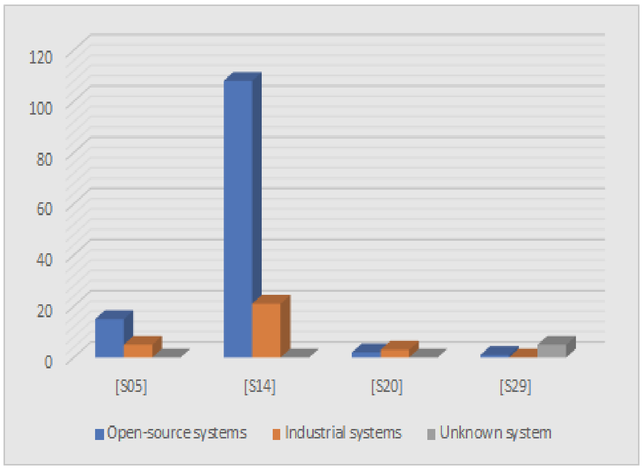

FIGURE 7. Common case study of systems among limited studies.

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.

TABLE 12. Validation level.

Study ID	Validation level
[S01], [S16], [S20], [S21], [S26], [S27], [S32], [S34], [S36], [S37], [S39], [S40], [S43],	Small experiment
[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].	Large experiment
[S10], [S18], [S22].	Independently validated
[S12], [S33].	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],	Open-source system case study
[S05], [S11], [S14], [S19], [S20], [S27], [S28], [S35], [S38], [S40].	Industrial system case study
[S13], [S16], [S22], [S29], [S30], [S32], [S33], [S41], [S42], [S43].	Unknown system case study

3) 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 [10]. 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.

TABLE 14. Tools used for calculating architectural erosion metrics.

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

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, CKJM, 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 erosion metrics.

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 [43]: i) internal structure-based metrics, which refer to metrics based on the internal structure and their relationships of the higher-level artifacts (components,

TABLE 15. The approach of the metrics type.

Metric	The approach of metrics type
PDIpk, PCIpk, PDIcl, PCIcl, PkDI, PkCI, 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, MPC, DAC, NOD, ATFD, PCQ, LCOMP, LCOM, TCC, RCI, SCC, WMC, Class elegance, LOC, TNC, NCONN, NELEM, NCML, CLD, NOS.	Internal structure-based metrics
NCC, CDC, CDO, CDLOC, NAC, COL, COC, Mrcc, MQ, RBMS, cvg, a2a, MoJo, PC, MI, SIG, SQALE, MDS, CC, DC, DP, Public API.	Specific model-based metrics
Mnewm, Mg&g, Mbunch, SD, MCC, XS.	Graph-based metrics

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.

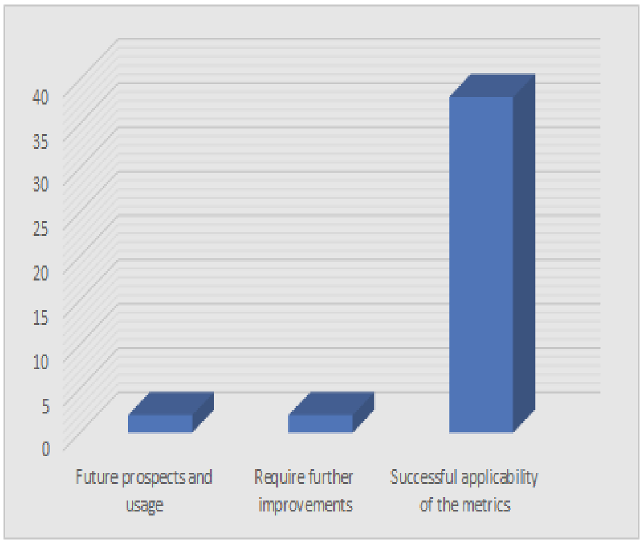

FIGURE 8. The extent of the applicability of the metrics within the context of architectural erosion.

VI. DISCUSSION

In this section, the discussion of the findings that have already been shown in the results section based on research questions and the study's goal will be analyzed. Additionally, some recommendations will be stated to identify future research directions in this domain.

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 metric 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.

F. USED TOOLS FOR CALCULATING THE ARCHITECTURAL EROSION (RQ4)

Based on the tools that appeared to be used within the included studies, we observed that most studies did not state the tools when calculating metrics. However, this does not mean that the validated metrics did not involve any tool, which explains that there are metrics used within the tools in the studies. At the same time, the same metrics were used 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 [43] 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.

VII. 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 [99] 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.

VIII. 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 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 [100], [101] 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.

TABLE 16. Study quality assessment checklist.

S_ ID	QA1	QA2	QA3	QA4	QA5	QA6	Score
S01	1	1	1	1	1	1	6
S02	1	1	1	1	1	1	6
S03	1	1	0.5	1	1	0.5	5
S04	1	1	1	1	1	0.5	5.5
S05	1	1	1	1	1	1	6
S06	1	1	1	1	1	0.5	5.5
S07	1	1	0.5	0.5	1	0	4
S08	1	1	1	1	1	0.5	5.5
S09	1	1	1	1	1	1	6
S10	1	0	0	0.5	1	0	2.5
S11	1	1	1	1	1	0.5	5.5
S12	1	0.5	1	1	1	0.5	5
S13	1	1	1	1	1	1	6
S14	1	1	0.5	0.5	1	0	4
S15	1	1	1	0.5	1	1	5.5
S16	1	1	0.5	1	1	1	5.5
S17	1	1	0.5	1	1	1	5.5
S18	1	1	1	1	1	1	6
S19	1	0.5	0.5	1	1	0.5	4.5
S20	1	0.5	0.5	1	1	0.5	4.5
S21	1	0.5	0.5	1	1	0	4
S22	1	1	0.5	1	1	1	5.5
S23	1	1	1	1	1	1	6
S24	1	0.5	0.5	0.5	1	0.5	4
S25	1	0.5	0.5	1	0	1	4
S26	1	1	0.5	1	1	0.5	5
S27	1	0.5	0.5	1	1	0.5	4.5
S28	1	0.5	0.5	1	1	0	4
S29	1	0.5	1	1	1	0.5	5
S30	1	0.5	0.5	1	1	0	4
S31	1	0.5	0.5	0.5	1	0	3.5
S32	0.5	0.5	0.5	0.5	1	0.5	3.5
S33	0.5	1	1	1	1	0.5	4
S34	0.5	0.5	1	1	1	0.5	4.5
S35	0.5	1	0.5	1	1	0.5	4.5
S36	1	1	0.5	1	1	1	5.5
S37	1	1	1	1	1	0	5
S38	0.5	1	1	1	1	1	5.5
S39	1	1	1	1	1	1	6
S40	1	1	0.5	1	1	0.5	5
S41	1	0.5	0.5	0.5	1	0.5	4
S42	1	1	1	1	1	1	6
S43	1	0.5	1	1	1	1	5.5

IX. 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

TABLE 17. 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.
CLD	Comment line density	Counts comment lines divided by code plus the comment lines and multiply
CDLOC	Concern Diffusion in LOC	by100.
CDC	Concern Diffusion over Components	The number of transition points through the lines of code for each concern.
CDO	Concern Diffusion over Operations	The number of classes affected by concern implementation.
CC	Cognitive Complexity	The number of methods affected by the concern implementation.
		Measures the current level of comprehension of the software system's various classes.
CBO	Coupling between objects	
Cvg	Cluster Coverage	The number of other objects to which it is coupled.
CMC	Cross-module co-changes	Calculates the extent to which the clusters of two architectures overlap.
		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	
DP	Defect persistence	A class's number of abstract data types.
DC	Defect Complexity	The number of system development phases (and releases) that this defect spans.
DL	Decoupling Level	The number of accompanying changes needed in various components to fix it.
DFSM	Dependency frequency support metric	The degree to which software is decoupled into separate modules.
		Investigating frequency in which classes of a module depend on classes of another.
Ce	Efferent Coupling	
XS		The number of classes from outside package A upon which classes reside on package A depends on implementing their features.
	Excessive complexity	
EWSOC		The total number of dependencies in the graph divided by the minimal feedback set (MFS) is reported as the percent XS.
	Exponentially Weighted Sum of Coupling	
EMD		Shared transactions are weighted exponentially according to their distance in time, highlighting current modifications rather than those that occurred previously.
CMD	External module dependencies	
	Incoming module dependency	
IPCI		The total number of outgoing dependencies from a module to other modules.
IPGF	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.
	Index of Package Goal Focus	
IIPE		The ratio of non-dependency package pairs to all possible package pairs.
	Index of Inter-Package Extending	The average overlap of service sets offered by a component to other components in a software system
IIPED		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.
	Index of Inter-Package Extending Diversion	
IIPU		
	Index of Inter-Package Usage	The average diversity of classes extended by a package in other packages
IIPUD		
	Index of Inter-Package Usage Diversion	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		
	Inter-Package Modularization Connections	A package in other packages extends the average diversity of classes.
IPMCC		
	Inter-Package Modularization Cyclic Connections	Measures common reuse of modularization for a package that is reused together.
IPMCD		
	Inter-Package Modularization Cyclic Dependencies	Measures the extent of modularization to which cyclic connections among the package are minimized.
IPMD		
	Inter-Package Modularization Dependencies	Measures the extent of modularization to minimize cyclic dependencies between the classes.
IMC		
	Inner-module co-changes	Measures the common closure of modularization for the classes that change among the packages.
IMD		
Instability (I)	Internal module dependencies	The number of co-changes for a file is at least another co-changed file in the same architectural module.
LWSOC	Instability (I)	
LOC	Linearly Weighted Sum of Coupling	The total number of dependencies among all files within a module.

TABLE 17. *(Continued.)* Metrics, definitions.

	Line of code	This rating measures the package's adaptability to change.
LCOMP		Shared transactions are linearly weighted according to their distance in time.
LCOM	Lack of Cohesion Metric for	Measures the file size determined by counting the number of non-empty non-
	Packages	comment lines.
MI	Lack of Cohesion in Methods	Refers to measure the lack of cohesion of the package.
		Measures the lack of cohesion in a class according to its common usage of
MCC	Maintainability Index	attributes methods.
		Identifies architectural, technical debt based on Halstead Volume, McCabe
Severity	Mccabe's Cyclomatic complexity	Cyclomatic Complexity, Lines of Code, and Comments.
		Quantifies independent paths in software source code using control flow graphs
MPC	Measures change-history of a	of functions, modules, methods, or classes.
MQ	software system	The total number of transactions in the software system's change history is
	Message Passing Coupling	divided by a total number of transactions involving the code smell I class C.
MDS	Modularization Quality	The number of method calls from one class to another.
		Measures the complete system to simplify understanding and limit modifications
Mnewm	Module dependency strength	to specific system areas.
		Measures how strongly the dependency between two modules is using
	Modularity's Newman	dependency intensity and dependency distribution.
Mg&g		Measures the modularity of social networks represented in graphical structures
		based on a theoretical heuristic for edges within a module greater than expected
Mrcc	Modularity's Guo and Gershenson	ones.
		Measures the modularity of physical entities using a difference of inter and intra
Mbunch	Modularity Relative Clustered Cost	edge densities.
		Measures the modularity of evolving software systems based on the relative
MoJo	Modularity Bunch Clustering	clustered cost of software systems.
		Measures software modularity that represented intra-edges of modules and inter-
NAC	Move-Join	edges between modules.
FAN-OUT		Determines the similarity between two different architectures with the same set
FAN-IN	Number of Architectural Concerns	of implementation-level entities.
NC	Number of called object	Number of architectural concerns a given code element (class or method) realizes.
NOCL	Number of caller object	The number of modules that the given module depends on.
NCL	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.
NCML	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.
	Number of comment lines	The number of concerns in each class.
NELEM	The number of dependencies	The number of lines containing either comment or commented-out code.
		The number of class-to-class use dependencies relates to the number of links
NOS	Number of Elements	among classes and, indirectly, among packages.
OMD		The total number of elements in the architecture (summing up the number of
	Number of statements	components and the number of connectors).
PCQ	Outgoing module dependency	The number of statements in a software component.
		An indicator of whether or not there is at least one dependency between two
PCC	Package Cohesion Quality	modules (m1 and m2).
		Measures ratio of internal dependencies of a package to all dependencies among
PCD	Package Cyclic Connections	and within a package.
		The ratio package cyclic connections among the packages to all package
PCF	Package Cyclic Dependencies	dependencies.
		The ratio of class cyclic dependencies within the package to all package
PCIpk	Pair change frequency	dependencies.
		The number of times in which file pairs and versions are modified in the same
PDIpk	Project Call Instability of package	commit.
		Analyzes the interaction between two software components that affect the
PCIcl	Project Design Instability of package	number of interactions at the package level.
	Project Call Instability of class	Measures how much a software system's components change from release N to
PDIcl		release N+1 at package levels.
		Analyzes the interaction between two software components that affect the
PC	Project Design Instability of class	number of interactions at the class level.
		Measures how much a software system's components change from release N to
Public API	Propagation Cost	release N+1 at class levels.
		Measures how tightly coupled a system is, and it is calculated using a matrix
RCI	Public application programming	model of the dependencies among files.
ROC	interface	A publicly available application programming interface enables data transmission
	The ratio of Cohesive Interactions	between one software product and another.
RFC	Ratio of Coupling	The ratio between actual cluster interactions to possible cluster interactions.
		The number of existing dependencies among modules to the number of all
RBMS	Response For Class	possible dependencies among modules.
		The total number of methods that can potentially be executed in response to a
SIG	Relationship-Based Similarity Metric	message received by an object of a class
SQALE		measures the change in the inheritance relationship. It takes the ratio of the total

TABLE 17. ***(Continued.)*** **Metrics, definitions.**

	Software Improvement Group	inheritance relationships to measure architecture stability.
SCC	Software Quality Enhancement	Quantifies TD based on an estimation of repair effort and maintenance effort.
SOC		computing the Remediation Cost (RC), which is the cost of correcting violations
	Sum cyclomatic complexity	of the rules set for each category.
SD	Sum of Coupling	The total of the Cyclomatic Complexity all nested functions or methods.
		The sum of the shared transactions between a given class c and all the classes n-
TF	Structural distance	coupled with c.
TCC		Measures the architecture between two revisions of a software system using a
TCMD	Ticket frequency	graph kernel.
	Tight Class Cohesion	The number of different bug or issue tracking tickets for which files are modified.
TOMD	Total incoming module dependencies	The relative number of methods is directly connected via accesses of attributes.
TNC		The total number of dependencies to a module and originating from other
	Total outgoing module dependencies	modules in a software system.
TWD	Total Number of Components	The total number of dependencies from a module m1 to other modules in a
		system.
WMC	Two-way dependencies	The number of components in the system appears in different levels of
		abstraction, including packages, classes, methods, and attributes.
	Weighted Methods per Class	The ratio between the number of two-way dependent clusters over all possible
		two-way dependencies.
		The sum of the statistical complexity of all methods in a class.

TABLE 18. **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 during maintenance, revealing architectural issues with severe consequences.
Architectural bad smell	Refers to an architectural decision that harms system quality and could lead to architectural erosion in the future.
Architecture modularization	Refers to the degree to which a system or software comprises discrete components with minimal impact on other components when one is changed.
Architectural change	Refers to the modifications that occur at different levels of abstraction and from multiple architectural views, which may appear as vague factors that cause architectural decay and instability.
Architectural technical debt	Refers to incomplete, immature, inadequate artifact, or suboptimal designs in the software development lifecycle.
Architectural dependency coupling analysis	Refers to correlation relationships among abstraction-level elements within a module based on a module with a direct or indirect dependency on another module.
Architectural cohesion	Refers to the degree to which the module elements are functionally related and keep the module together. A well-designed architecture will have high cohesion.
Architectural complexity	Refers to the complexity of the system's structure, stored information on how it works, and its makeup, which may reflect a state of having many parts and being difficult to understand or find an answer to.
Software architecture size	Refers to estimating the size of a software application or component to identify software productivity and predict fault locations for testing and maintenance effort. It has to do with the system's bulk size, whether as an overall metric, data, or functional measure.

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

See Table 16.

APPENDIX B

See Table 17.

APPENDIX C

See Table 18.

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. Ahmed Baabad would also like to express his gratitude to Hadhramout University and the Hadhramout Foundation, Yemen, for their support in tuition fees.

REFERENCES

[1] D. Garlan, "Software architecture: A roadmap," in *Proc. Conf. Future Softw. Eng.*, Limerick, Ireland, 2000, pp. 91–101, doi: 10.1145/336512.336537.

[2] J. Knodel, M. Lindvall, D. Muthig, and M. Naab, "Static evaluation of software architectures," in *Proc. Conf. Softw. Maintenance Reeng.*, Bari, Italy, 2006, p. 10.

[3] M. M. Lehman, "On understanding laws, evolution, and conservation in the large-program life cycle," *J. Syst. Softw.*, vol. 1, pp. 213–221, Jan. 1979.

[4] M. M. Lehman, "Laws of software evolution revisited," in *Proc. 5th Eur. Workshop Softw. Process. Technol.* Berlin, Germany: Springer, 1996, pp. 108–124.

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

[6] L. Dobrica and E. Niemelä, "A survey on software architecture analysis methods," *IEEE Trans. Softw. Eng.*, vol. 28, no. 7, pp. 638–653, Jul. 2002.

[7] A. Baabad, 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] 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.

[9] S. Misra, "An approach for the empirical validation of software complexity measures," *Acta Polytechnica Hungarica*, vol. 8, pp. 141–160, Jan. 2011.

[10] T. DeMarco, *Controlling Software Projects: Management, Measurement, and Estimates*. Upper Saddle River, NJ, USA: Prentice-Hall, 1986.

[11] S. Ducasse and D. Pollet, "Software architecture reconstruction: A process-oriented taxonomy," *IEEE Trans. Softw. Eng.*, vol. 35, no. 4, pp. 573–591, Jul. 2009.

[12] M. Shaw and P. Clements, "The golden age of software architecture," *IEEE Softw.*, vol. 23, no. 2, pp. 31–39, Mar. 2006.

[13] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Reading, MA, USA: Addison-Wesley, 2012.

[14] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *SIGSOFT Softw. Eng. Notes*, vol. 17, no. 4, pp. 40–52, Oct. 1992.

[15] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*. Upper Saddle River, NJ, USA: Prentice-Hall, 1996.

[16] M. H. Klein, R. Kazman, L. Bass, J. Carriere, M. Barbacci, and H. Lipson, "Attribute-based architecture styles," in *Proc. Conf. Softw. Archit.*, San Antonio, TX, USA, P. Donohoe, Ed. Boston, MA, USA: Springer, 1999, pp. 225–243.

[17] A. Jansen, J. V. D. Ven, P. Avgeriou, and D. K. Hammer, "Tool support for architectural decisions," in *Proc. Workshop Conf. Softw. Archit. (WICSA)*, Mumbai, India, 2007, p. 4.

[18] D. L. Parnas, "Software aging," in *Proc. 16th Int. Conf. Softw. Eng.*, Sorrento, Italy, 1994, pp. 279–287.

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

[20] J. Bosch, "Software Architecture: The next step," in *Proc. 1st Eur. Workshop Softw. Archit.*, vol. 3047, Berlin, Germany: Springer, 2004, pp. 194–199.

[21] M. Riaz, M. Sulayman, and H. Naqvi, "Architectural decay during continuous software evolution and impact of 'design for change' on software architecture," in *Proc. Int. Conf. Adv. Softw. Eng. Appl.* Berlin, Germany: Springer, 2009, pp. 119–126.

[22] M. Lindvall, R. Tesoriero, and P. Costa, "Avoiding architectural degeneration: An evaluation process for software architecture," in *Proc. 8th Symp. Softw. Metrics*, Ottawa, ON, Canada, 2002, pp. 77–86.

[23] N. Medvidovic and R. N. Taylor, "Software architecture: Foundations, theory, and practice," in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng.*, 2010, pp. 471–472.

[24] J. Lenhard, M. Blom, and S. Herold, "Exploring the suitability of source code metrics for indicating architectural inconsistencies," *Softw. Qual. J.*, vol. 27, no. 1, pp. 241–274, Mar. 2019.

[25] V. Bandara and I. Perera, "Identifying software architecture erosion through code comments," in *Proc. 18th Int. Conf. Adv. Emerg. Regions (ICTer)*, Sep. 2018, pp. 62–69.

[26] Z. Li and J. Long, "A case study of measuring degeneration of software architectures from a defect perspective," in *Proc. 18th Asia–Pacific Softw. Eng. Conf.*, 2011, pp. 242–249.

[27] H. P. Breivold and I. Crnkovic, "A systematic review on architecting for software evolvability," in *Proc. 21st Austral. Softw. Eng. Conf.*, Auckland, New Zealand, Apr. 2010, pp. 13–22, doi: 10.1109/ASWEC.2010.11.

[28] W. J. Dzidek, E. Arisholm, and L. C. Briand, "A realistic empirical evaluation of the costs and benefits of UML in software maintenance," *IEEE Trans. Softw. Eng.*, vol. 34, no. 3, pp. 407–432, May 2008.

[29] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: A survey," *IEEE Trans. Softw. Eng.*, vol. 30, no. 5, pp. 295–310, May 2004.

[30] 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 Trans. Softw. Eng.*, vol. 41, no. 2, pp. 157–175, Feb. 2015.

[31] H. Koziolek, B. Schlich, S. Becker, and M. Hauck, "Performance and reliability prediction for evolving service-oriented software systems," *Empirical Softw. Eng.*, vol. 18, no. 4, pp. 746–790, 2013.

[32] M. Berry and M. F. Vandenbroek, "A targeted assessment of the software measurement process," in *Proc. 7th Int. Softw. Metrics Symp.*, London, U.K., 2001, pp. 222–235.

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

[34] M. Unterkalmsteiner, T. Gorschek, A. K. Islam, C. K. Cheng, R. B. Permadi, and R. Feldt, "Evaluation and measurement of software process improvement—A systematic literature review," *IEEE Trans. Softw. Eng.*, vol. 38, no. 2, pp. 398–424, Mar. 2012.

[35] S. Misra, I. Akman, and R. Colomo-Palacios, "Framework for evaluation and validation of software complexity measures," *IET Softw.*, vol. 6, no. 4, pp. 323–334, Aug. 2012.

[36] *ISO/IEC/IEEE International Standard—Systems and Software Engineering–Measurement Process*, Standard ISO/IEC/IEEE 15939, 2017, pp. 1–49.

[37] R. Malhotra, *Empirical Research in Software Engineering: Concepts, Analysis, and Applications*. Boca Raton, FL, USA: CRC Press, 2015.

[38] P. Goodman, *The Practical Implementation of Software Metrics*. New York, NY, USA: McGraw-Hill, 1993.

[39] M. Abdellatief, A. B. M. Sultan, A. A. A. Ghani, and M. A. Jabar, "A mapping study to investigate component-based software system metrics," *J. Syst. Softw.*, vol. 86, no. 3, pp. 587–603, Mar. 2013.

[40] M. Staron and W. Meding, "Metrics for software design and architectures," in *Automotive Software Architectures: An Introduction*, M. Staron, Ed. Cham, Switzerland: Springer, 2017, pp. 179–199.

[41] T. Tahir, G. Rasool, and C. Gencel, "A systematic literature review on software measurement programs," *Inf. Softw. Technol.*, vol. 73, pp. 101–121, May 2016.

[42] T. Coulin, M. Detante, and F. J. A. Petrillo, "Software architecture metrics: A literature review," 2019, *arXiv:1901.09050*.

[43] S. Stevanetic and U. Zdun, "Software metrics for measuring the understandability of architectural structures: A systematic mapping study," in *Proc. 19th Int. Conf. Eval. Assessment Softw. Eng.*, Nanjing, China, Apr. 2015, pp. 1–14, doi: 10.1145/2745802.2745822.

[44] M. Alenezi, "Software architecture quality measurement stability and understandability," *Int. J. Adv. Comput. Sci. Appl.*, vol. 7, no. 7, pp. 550–559, 2016.

[45] A. S. Nuñez-Varela, H. G. Pérez-Gonzalez, F. E. Martínez-Perez, and C. Soubervielle-Montalvo, "Source code metrics: A systematic mapping study," *J. Syst. Softw.*, vol. 128, pp. 164–197, Jun. 2017.

[46] H. Koziolek, "Sustainability evaluation of software architectures: A systematic review," in *Proc. SIGSOFT Conf.-QoSA ACM SIGSOFT Symp.*, Boulder, CO, USA, 2011, pp. 3–12, doi: 10.1145/2000259.2000263.

[47] B. A. Kitchenham, D. Budgen, and O. Pearl Brereton, "Using mapping studies as the basis for further research—A participant-observer case study," *Inf. Softw. Technol.*, vol. 53, no. 6, pp. 638–651, Jun. 2011.

[48] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *Proc. 12th Int. Conf. Eval. Assessment Softw. Eng.*, Rome, Italy, 2008, pp. 1–10.

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

[50] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," EBSE, Chennai, India, Tech. Rep. Ver. 2.3, 2007.

[51] D. Badampudi, C. Wohlin, and K. Petersen, "Experiences from using snowballing and database searches in systematic literature studies," in *Proc. 19th Int. Conf. Eval. Assessment Softw. Eng.*, Nanjing, China, Apr. 2015, pp. 1–10, doi: 10.1145/2745802.2745818.

[52] E. Mourao, M. Kalinowski, L. Murta, E. Mendes, and C. Wohlin, "Investigating the use of a hybrid search strategy for systematic reviews," in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, Nov. 2017, pp. 193–198.

[53] J. Webster and R. T. Watson, "Analyzing the past to prepare for the future: Writing a literature review," *MIS Quart.*, vol. 26, no. 2, pp. 23–30, 2002.

[54] O. Dieste and A. G. Padua, "Developing search strategies for detecting relevant experiments for systematic reviews," in *Proc. 1st Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, Madrid, Spain, 2007, pp. 215–224.

[55] L. Aversano, D. Guardabascio, and M. Tortorella, "An empirical study on the architecture instability of software projects," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 29, no. 4, pp. 515–545, 2019.

[56] A. Sejfia, "A pilot study on architecture and vulnerabilities: Lessons learned," in *Proc. IEEE/ACM 2nd Int. Workshop Establishing Community-Wide Infrastruct. Archit.*, Montreal, QC, Canada, May 2019, pp. 42–47.

[57] R. Mo, Y. Cai, R. Kazman, L. Xiao, and Q. Feng, "Architecture anti-patterns: Automatically detectable violations of design principles," *IEEE Trans. Softw. Eng.*, vol. 47, no. 5, pp. 1008–1028, May 2019.

[58] S. G. Maisikeli, "Measuring architectural stability and instability in the evolution of software systems," in *Proc. Inf. Technol. Trends, Emerg. Technol. Artif. Intell.*, Dubai, United Arab Emirates, 2019, pp. 263–275.

[59] L. P. D. S. Carvalho and R. Novais, "Investigating the relationship between code smell agglomerations and architectural concerns: Similarities and dissimilarities from distributed, service-oriented, and mobile systems," in *Proc. Brazilian Symp. Softw. Compon., Archit., Reuse*, Sao Carlos, Brazil, 2018, pp. 1–12, doi: 10.1145/3267183.3267184.

[60] A. Shahbazian, D. Nam, and N. Medvidovic, "Toward predicting architectural significance of implementation issues," in *Proc. 15th Int. Conf. Mining Softw. Repositories*, Gothenburg, Sweden, May 2018, pp. 215–219, doi: 10.1145/3196398.3196440.

[61] 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 Softw. Eng.*, vol. 22, no. 3, pp. 1146–1193, Jun. 2017.

[62] S. H. M. A. Jalbani and A. A. M. Kashif, "Evaluating dependency based package-level metrics for multi-objective maintenance tasks," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 10, pp. 345–354, 2017.

[63] H. Rocha, R. S. Durelli, R. Terra, S. Bessa, and M. T. Valente, "DCL 2.0: Modular and reusable specification of architectural constraints," *J. Brazilian Comput. Soc.*, vol. 23, no. 1, p. 12, Dec. 2017.

[64] F. A. Fontana, R. Roveda, M. Zanoni, C. Raibulet, and R. Capilla, "An experience report on detecting and repairing software architecture erosion," in *Proc. 13th Work. IEEE/IFIP Conf. Softw. Archit. (WICSA)*, Apr. 2016, pp. 21–30.

[65] R. Mo, Y. Cai, R. Kazman, L. Xiao, and Q. Feng, "Decoupling level: A new metric for architectural maintenance complexity," in *Proc. 38th Int. Conf. Softw. Eng.*, Austin, TX, USA, May 2016, pp. 499–510, doi: 10.1145/2884781.2884825.

[66] M. De Oliveira Barros, F. De Almeida Farzat, and G. H. Travassos, "Learning from optimization: A case study with apache ant," *Inf. Softw. Technol.*, vol. 57, no. 1, pp. 684–704, 2015.

[67] E. Guimarães, A. Garcia, and Y. Cai, "Architecture-sensitive heuristics for prioritizing critical code anomalies," in *Proc. 14th Int. Conf. Modularity*, Fort Collins, CO, USA, Mar. 2015, pp. 68–80, doi: 10.1145/2724525.2724567.

[68] F. A. Fontana, V. Ferme, and M. Zanoni, "Towards assessing software architecture quality by exploiting code smell relations," in *Proc. IEEE/ACM 2nd Int. Workshop Softw. Archit. Metrics*, Florence, Italy, May 2015, pp. 1–7.

[69] Z. Li, P. Liang, P. Avgeriou, N. Guelfi, and A. Ampatzoglou, "An empirical investigation of modularity metrics for indicating architectural technical debt," in *Proc. 10th Int. ACM SIGSOFT Conf. Qual. Softw. Archit.*, Marcq-En-Bareul, France, 2014, pp. 119–128, doi: 10.1145/2602576.2602581.

[70] M. Ferreira, E. Barbosa, I. Macia, R. Arcoverde, and A. Garcia, "Detecting architecturally-relevant code anomalies: A case study of effectiveness and effort," in *Proc. 29th Annu. ACM Symp. Appl. Comput.*, Gyeongju, Republic Korea, Mar. 2014, doi: 10.1145/2554850.2555036.

[71] I. Macia, A. Garcia, C. Chavez, and A. Von Staa, "Enhancing the detection of code anomalies with architecture-sensitive strategies," in *Proc. 17th Eur. Conf. Softw. Maintenance Reeng.*, Genova, Italy, Mar. 2013, pp. 177–186.

[72] E. Guimaraes, A. Garcia, E. Figueiredo, and Y. Cai, "Prioritizing software anomalies with software metrics and architecture blueprints: A controlled experiment," in *Proc. 5th Int. Workshop Modeling Softw. Eng.*, San Francisco, CA, USA, May 2013, pp. 82–88.

[73] 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," in *Proc. 11th Annu. Int. Conf. Aspect-oriented Softw. Develop.*, Potsdam, Germany, 2012, pp. 167–178, doi: 10.1145/2162049.2162069.

[74] Z. Li, N. H. Madhavji, S. S. Murtaza, M. Gittens, A. V. Miranskyy, D. Godwin, and E. Cialini, "Characteristics of multiple-component defects and architectural hotspots: A large system case study," *Empirical Softw. Eng.*, vol. 16, no. 5, pp. 667–702, Oct. 2011.

[75] M. Steff and B. Russo, "Measuring architectural change for defect estimation and localization," in *Proc. Int. Symp. Empirical Softw. Eng. Meas.*, anff, AB, Canada, Sep. 2011, pp. 225–234.

[76] R. S. Sangwan, P. Vercellone-Smith, and C. J. Neill, "Use of a multidimensional approach to study the evolution of software complexity," *Innov. Syst. Softw. Eng.*, vol. 6, no. 4, pp. 299–310, Dec. 2010.

[77] A. Capiluppi and T. Knowles, "Software engineering in practice: Design and architectures of FLOSS systems," in *Open Source Ecosystems: Diverse Communities Interacting*. Berlin, Germany: Springer, 2009, pp. 34–46.

[78] R. Singh, A. Bindal, and A. Kumar, "Reducing maintenance efforts of developers by prioritizing different code smells," *Int. J. Innov. Technol. Exploring Eng.*, vol. 8, no. 8, pp. 139–144, 2019.

[79] M. Nayebi, Y. Cai, R. Kazman, G. Ruhe, Q. Feng, C. Carlson, and F. Chew, "A longitudinal study of identifying and paying down architecture debt," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng., Softw. Eng. Pract. (ICSE-SEIP)*, Montreal, QC, Canada, May 2019, pp. 171–180, doi: 10.1109/ICSE-SEIP.2019.00026.

[80] V. Zapalowski, I. Nunes, and D. J. Nunes, "The WGB method to recover implemented architectural rules," *Inf. Softw. Technol.*, vol. 103, pp. 125–137, Nov. 2018.

[81] 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?" in *Proc. Brazilian Symp. Softw. Eng.*, Sao Carlos, Brazil, 2018, pp. 22–31, doi: 10.1145/3266237.3266261.

[82] R. Roveda, F. Arcelli Fontana, I. Pigazzini, and M. Zanoni, "Towards an architectural debt index," in *Proc. 44th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Prague, Czech Republic, Aug. 2018, pp. 408–416.

[83] L. Rizzi, F. A. Fontana, and R. Roveda, "Support for architectural smell refactoring," in *Proc. 2nd Int. Workshop Refactoring*, Montpellier, France, Sep. 2018, pp. 1–10, doi: 10.1145/3242163.3242165.

[84] A. Biaggi, F. Arcelli Fontana, and R. Roveda, "An architectural smells detection tool for c and C++ projects," in *Proc. 44th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Prague, Czech Republic, Aug. 2018, pp. 417–420.

[85] F. A. Fontana, I. Pigazzini, R. Roveda, and M. Zanoni, "Automatic detection of instability architectural smells," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Raleigh, NC, USA, Oct. 2016, pp. 433–437.

[86] E. Ersoy and H. Sözer, "Evaluating software architecture erosion for PL/SQL programs," in *Proc. 11th Eur. Conf. Softw. Archit., Companion*, Canterbury, U.K., Sep. 2017, pp. 159–165, doi: 10.1145/3129790.3129811.

[87] 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," in *Proc. Sci. Workshop*, Edinburgh, Scotland, May 2016, pp. 1–8, doi: 10.1145/2962695.2962716.

[88] S. Stevanetic, T. Haitzer, and U. Zdun, "Supporting software evolution by integrating DSL-based architectural abstraction and understandability related metrics," in *Proc. Eur. Conf. Softw. Archit. Workshops*, Vienna, Austria, 2007, pp. 1–8, doi: 10.1145/2642803.2642822.

[89] 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," in *Proc. 8th ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, Turin, Italy, 2014, pp. 1–8, doi: 10.1145/2652524.2652581.

[90] E. Guimaraes, A. Garcia, and Y. Cai, "Exploring blueprints on the prioritization of architecturally relevant code anomalies—A controlled experiment," in *Proc. IEEE 38th Annu. Comput. Softw. Appl. Conf.*, Vasteras, Sweden, Jul. 2014, pp. 344–353.

[91] R. Schwanke, L. Xiao, and Y. Cai, "Measuring architecture quality by structure plus history analysis," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, San Francisco, CA, USA, May 2013, pp. 891–900.

[92] M. D'Ambros, M. Lanza, and R. Robbes, "On the relationship between change coupling and software defects," in *Proc. 16th Work. Conf. Reverse Eng.*, Lille, France, 2009, pp. 135–144.

[93] A. MacCormack and D. J. Sturtevant, "Technical debt and system architecture: The impact of coupling on defect-related activity," *J. Syst. Softw.*, vol. 120, pp. 170–182, Oct. 2016.

[94] J. A. Diaz-Pace, A. Tommasel, I. Pigazzini, and F. A. Fontana, "Sen4Smells: A tool for ranking sensitive smells for an architecture debt index," in *Proc. IEEE Congreso Bienal Argentina (ARGENCON)*, Resistencia, Argentina, 2020, pp. 1–7.

[95] J. Garcia, E. Kouroshfar, N. Ghorbani, and S. Malek, "Forecasting architectural decay from evolutionary history," *IEEE Trans. Softw. Eng.*, early access, Feb. 18, 2021, doi: 10.1109/TSE.2021.3060068.

[96] P. W. Lawrence JD, A. Sicherman, and J. GL, "Assessment of software reliability measurement methods for use in probabilistic risk assessment," Fission Energy Syst. Saf. Program, Lawrence Livermore Nat. Lab., Livermore, CA, USA, Tech. Rep. UCRLID-136035, 1988.

[97] K. P. Srinivasan and T. Devi, "Software metrics validation methodologies in software engineering," *Int. J. Softw. Eng. Appl.*, vol. 5, no. 6, pp. 87–102, Nov. 2014.

[98] B. Kitchenham, S. L. Pfleeger, and N. Fenton, "Towards a framework for software measurement validation," *IEEE Trans. Softw. Eng.*, vol. 21, no. 12, pp. 929–944, Dec. 1995.

[99] M. A. Babar and I. Gorton, "Software architecture review: The state of practice," *J. Comput.*, vol. 42, no. 7, pp. 26–32, 2009.

[100] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," EBSE, Chennai, India, Tech. Rep. Ver. 2.3, 2007.

[101] A. Nguyen-Duc, D. S. Cruzes, and R. Conradi, "The impact of global dispersion on coordination, team performance and software quality—A systematic literature review," *Inf. Softw. Technol.*, vol. 57, pp. 277–294, Jan. 2015.