Information and Software Technology 181 (2025) 107677

Contents lists available at ScienceDirect

Information and Software Technology

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

Mining software repositories for software architecture — A systematic mapping study

Mohamed Soliman [a], Michel Albonico [b], Ivano Malavolta [c,*], Andreas Wortmann [d]

[a] Heinz Nixdorf Institut, Paderborn University, Paderborn, Germany
[b] Federal University of Technology, Paraná (UTFPR), IntelAgir Research Group, Francisco Beltrão PR, Brazil
[c] Vrije Universiteit Amsterdam, Software and Sustainability Research Group, Amsterdam, The Netherlands
[d] University of Stuttgart, Institute for Control Engineering of Machine Tools and Manufacturing Units (ISW), Stuttgart, Germany

ARTICLE INFO

Dataset link: https://doi.org/10.5281/zenodo.14614882

Keywords:
Mining software repositories
Software architecture
Empirical research

ABSTRACT

Context: A growing number of researchers are investigating how Mining Software Repositories (MSR) approaches can support software architecture activities, such as architecture recovery, tactics identification, architectural smell detection, and others. However, as of today, it is difficult to have a clear view of existing research on MSR for software architecture.
Objectives: The objective of this study is to identify, classify, and summarize the state-of-the-art MSR approaches applied to software architecture (MSR4SA).
Methods: This study is designed according to the *systematic mapping study* research method. Specifically, out of 2442 potentially relevant studies, we systematically identify 151 primary studies where MSR approaches are applied to perform software architecture activities. Then, we rigorously extract relevant data from each primary study and synthesize the obtained results to produce a clear map of reasons for adopting MSR approaches to support architecting activities, used data sources, applied MSR techniques, and captured architectural information.
Results: The major reasons to adopt MSR4SA techniques are about addressing industrial concerns like *achieving quality attributes* and *minimizing practitioners' efforts*. Most MSR4SA studies support architectural analysis, while architectural synthesis and evaluation are not commonly supported in MSR4SA studies. The most frequently mined data sources are *source code repositories* and *issue trackers*, which are also commonly mined together. Most of the MSR4SA studies apply more than one mining technique, where the most common MSR techniques are: (*source code analysis*, *model analysis*, *statistical analysis*), (*machine learning*, *NLP*). *Architectural quality issues* and *components* are the mostly mined type of information.
Conclusion: Our results give a solid foundation for researchers and practitioners towards future research and applications of MSR approaches for software architecture.

Contents

1. Introduction 2
2. Study design 2
 2.1. Goal and research questions 3
 2.2. Search and selection 3
 2.3. Data extraction 4
 2.4. Data synthesis 4
3. Overview of the selected primary studies 5
4. Goals of mining software repositories to support software architecture (RQ1) 5
 4.1. Reasons of MSR4SA 5
 4.2. Study types 8
 4.3. Architectural design activities 8
5. Data sources mined to support software architecture (RQ2) 8

* Corresponding author.
E-mail address: i.malavolta@vu.nl (I. Malavolta).

https://doi.org/10.1016/j.infsof.2025.107677
Received 3 August 2024; Received in revised form 22 January 2025; Accepted 24 January 2025
Available online 12 February 2025

5.1. Studied systems ... 10
6. MSR techniques to support software architecture (RQ3) ... 10
7. Architectural information mined from software repositories (RQ4) ... 11
8. Discussion ... 11
8.1. Implications for researchers ... 11
8.2. Implications for practitioners ... 13
9. Limitations and threats to validity ... 13
10. Related work ... 13
11. Conclusion and future work ... 15
CRediT authorship contribution statement ... 15
Declaration of competing interest ... 15
Data availability ... 15
References ... 15
Primary studies ... 16

1. Introduction

Mining software repositories (MSR) is concerned with "uncovering interesting and actionable information about software systems and projects using the vast amounts of software data such as source control systems, defect tracking systems, code review repositories".[1] Software engineering researchers mine software repositories to support different development tasks, such as analyzing requirements, making design decisions, fixing bugs, and testing.

Some of the development tasks are concerned with *Architectural Design Activities* (ADAs), which software developers perform to make design decisions (*e.g.,* structuring the system into multiple layers, selecting technologies, detecting architectural smells). Hofmeister et al. defined three common ADAs [1]: (1) *architectural analysis*: software developers identify significant requirements, such as the main software features or quality attributes (*e.g.,* security or performance). To this end, they interact with stakeholders and analyze documents as well as source code; (2) *architectural synthesis*: software developers identify architectural solutions to fulfill the requirement, such as the design of components, technologies to be employed, or architectural tactics [2]. Therefore they re-use their knowledge and search for architectural solutions, e.g., in the Web or in developer communities; and (3) *architectural evaluation*: software developers validate the proposed architectural solutions against the identified requirements. The goal of researchers in the software architecture field is to support software developers in performing ADAs.

Recently, software architecture (SA) researchers are proposing *approaches to utilize MSR to support software developers in performing ADAs*, we call these MSR4SA approaches (*e.g.,* [P1,P2,P3]). These approaches mean to support researchers and practitioners in extracting architecturally-relevant information from large and complex software repositories. For example, source code analysis and machine learning have been applied to automatically capture architectural design decisions from source control and issue tracking systems [P4,P5]. Software architects can re-use the captured design decisions from [P4,P5] to perform the architectural synthesis ADA. In another study, multiple versions of source control repositories have been analyzed and compared for tracking architectural smells and their impact on the overall system quality [P6]. Software architects can use the approach from [P6] to identify quality requirements when performing the architectural analysis ADA.

While software architecture researchers have recently been proposing MSR4SA approaches, there is neither a comprehensive overview of MSR4SA approaches, nor an overview about when and how researchers develop MSR4SA approaches. The main reasons for such impediments are: (i) the landscape on MSR4SA approaches is extremely heterogeneous and (ii) MSR and software architecture are two separate communities, with a relatively small intersection in terms of publication venues. In consequence, it is difficult for researchers and practitioners to have a good overview of MSR4SA approaches, of their characteristics, and of the specific context in which they can be applied.

The main goal of this study is to identify, classify, and summarize the state of the art on MSR4SA approaches. By achieving this goal, we provide the first comprehensive view on the current state of the art for applying MSR techniques to support software architecture.

We carry out a systematic mapping study [3], where we first consider the proceedings of the top-tier publication venues dedicated to the MSR and SA research areas (*i.e.,* MSR, ECSA, ICSA) across all their editions, resulting in a set of *387 potentially-relevant studies*. Then, we systematically apply a rigorously defined selection procedure, which leads to a starting set of *66 primary studies*. From the starting set, we conduct a forward and backward snowballing, which results in an additional 2055 non-duplicate studies (*i.e.,* a total of 2442 studies). After applying the same systematic selection as for the starting set, we select 85 additional studies, summing up to a total of 151 included primary studies. Then, we define a data extraction form for categorizing MSR4SA approaches and collaboratively apply it to each of the 151 primary studies. Finally, we synthesize the obtained data into a clear map of the MSR4SA state of the art. The map includes: the goals for adopting MSR techniques to support software architecture, used data sources, applied MSR techniques, and captured architectural information.

This study's contributions are: (i) *evidence-based maps* of the goals, data sources, MSR techniques, and types of architectural information considered in MSR4SA approaches; (ii) a discussion of the obtained map in terms of *implications for both researchers and practitioners*; (iii) the *replication package* of the study for independent verification and replication [4].

The target audience of this study includes primarily researchers who are working at the intersection of MSR and SA (*i.e.,* MSR4SA researchers). Researchers can use our maps to position their own studies and identify promising research gaps to be filled in their future studies. Practitioners can use our maps as well, mainly as a catalog of research-rooted MSR4SA approaches which might be applied in their own projects and organizations.

2. Study design

This study is carried out according to established guidelines on empirical software engineering [5] and systematic mapping studies [3]. As shown in Fig. 1, the study is designed around four main phases [5]:

- *Phase 1* comprises planning the study; specifically, we firstly produce a precise definition of MSR4SA (see Section 1), then we define the goal and research questions of the study (Section 2.1), and finally, we create a plan for carrying out all the other activities of this study (they are described in the remainder of this

[1] Call for papers of MSR 2025: https://conf.researchr.org/home/msr-2025.

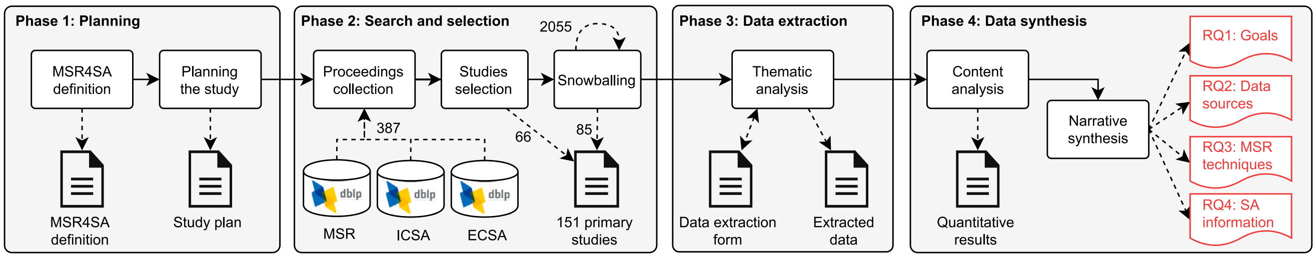


Fig. 1. Our study comprises four phases: (1) planning, (2) search and selection, (3) data extraction, and (4) data synthesis.

section). To mitigate potential threats to validity, the plan has been iteratively discussed among the authors and defined *a priori*, before proceeding with the other phases of the study.

- *Phase 2* aims at identifying a set of representative studies on MSR4SA (Section 2.2). Therefore, in this phase, we apply our search terms to the selected databases and apply criteria for inclusion and exclusion of articles.
- *Phase 3* deals with the creation and the continuous refinement of a data extraction form to compare the selected primary studies (Section 2.3). For this, we designed a data extraction form and applied it to the articles that advanced to this phase. During this phase, further articles may be eliminated from the final corpus.
- *Phase 4* is the thematic analysis:, in which we analyze the data quantitatively and qualitatively, which leads to answers for the research question (Section 2.4).

A *replication package* is publicly available [4] for independent replication and verification of the study. The replication package includes the raw data of our search and selection phase, the list of selected primary studies, the raw data extracted from each primary study, and the themes extracted during the thematic analysis.

We assessed the quality of this study according to the checklist proposed by Petersen et al. [3]. The checklist contains the main features of an ideal systematic mapping study and proposes a scoring system based on the ratio of the number of features covered by a given study and the total number of features in the checklist. This study achieves a score of 52%, exceeding both the median (33%) and the maximum (48%) scores reported in the tertiary study by Petersen et al. [6].

2.1. Goal and research questions

According to the GQM (goal question metric) template proposed by Basili et al. [7], the goal of this study can be characterized as in Table 1. After the study goal, we identified the following four research questions, each dealing with a specific type of characteristics of MSR4SA approaches.

RQ1*: Why do researchers mine software repositories to support software architecture?*

This research question aims to elicit the goals motivating the usage of MSR to support SA. Indeed, as it emerged in our study, SA researchers mine software repositories to achieve various goals, such as: for extracting architecturally-relevant information (*e.g.,* capturing architectural knowledge), for solving well-known challenges in applying empirical research in the SA field (*e.g.,* extensive manual effort), and to support certain ADAs [1] (*i.e.,* architectural analysis, synthesis, and evaluation). Answering this RQ can help SA researchers in determining when to mine software repositories, and for which specific activities. Moreover, the answer to this RQ helps in identifying possible research gaps and future work.

RQ2*: Which data sources are mined to support software architecture?*

Software repositories can be used for different purposes and can contain different types of data, *e.g.,* GitHub repositories contain source code over multiple versions, Jira boards contain issues, feature requests, and bugs, and mailing lists contain developers' discussions.

Table 1
GQM definition.

Purpose	Identify, classify, and summarize
Issue	the characteristics
Object	of existing MSR4SA approaches
Viewpoint	from the researcher's and practitioner's point of view

Answering this RQ allows us to build a map of the different data sources mined for SA. Moreover, we also cross-check how the identified data sources support different goals, activities, and mining techniques. Our results support researchers in (i) making better-informed decisions about which data sources to use for their specific goal, and (ii) identifying promising data sources that have not yet been used for specific software architecture activities.

RQ3*: Which mining techniques are applied to support software architecture?*

Mining software repositories requires using several analysis techniques such as natural language processing [8], source code analysis [9], and statistical methods [10]. This RQ aims to determine common analysis techniques used by researchers for mining software repositories to support architectural activities. Answering this RQ supports both (1) software architecture researchers and practitioners in identifying those techniques that have been successfully applied in the context of software architecture, and (2) MSR researchers in identifying promising avenues for improving MSR techniques.

RQ4*: Which architectural information is mined to support software architecture?*

MSR techniques allow researchers and architects to capture different types of information (*e.g.,* architectural components, design decisions, patterns, and architecture guidelines). This RQ aims to determine the types of information which have been successfully mined to achieve architecturally-relevant goals. Answering this RQ supports SA researchers who need to position their work with respect to the state of the art. Also, it supports practitioners via a map of MSR4SA approaches to extract the architectural information they need in their own projects and organizations.

2.2. Search and selection

To answer our research questions, it is important to search and select a representative sample of MSR4SA studies [3,5]. Our search and selection procedure is based on a combination of (1) manual search on DBLP (the *Proceedings collection* step in Fig. 1), (2) a manual selection of the primary studies according to well-defined criteria (the *Studies collection* step in Fig. 1), and (3) backward/forward snowballing (the *Snowballing* step in Fig. 1).

As shown in Fig. 1, the first step of our search and selection phase is the proceedings collection, where we download all publications from the proceedings of all targeted conferences. The venues we selected for this study are: the Mining Software Repositories conference (MSR), the International Conference on Software Architecture (ICSA), and the European Conference on Software Architecture (ECSA). Each selected venue is centered on either the MSR or SA research fields and has high-quality publications (all venues are ranked *A* in the CORE Conference

Ranking [11]). Our focus on a representative set of publication venues is mainly due to the lack of unified vocabulary among the MSR4SA publications, which makes it challenging to build up a complete and correct search query to identify MSR4SA papers. We verified such phenomenon through pilot searches, where we aimed to identify suitable search keywords using the PICO (Population, Intervention, Comparison, Outcomes) method [12]. In those searches (i) the term MSR does not commonly occur in most of the identified MSR4SA papers and (ii) researchers tend to use a wide variety of terms regarding their goals (*e.g.,* mitigate architecture erosion), the mined sources (*e.g.,* Stack Overflow), and mining techniques (*e.g.,* machine learning). Moreover, applying the SSSMS method is a natural choice for our study since there are research venues that are very much focused on either MSR or SA. These venues provide a controllable and highly precise population from which we can sample our primary studies with relatively low effort.

We consider all editions of MSR (*i.e.,* from 2004 to 2023), all editions of ICSA (*i.e.,* from 2017 to 2023), and all editions of ECSA (*i.e.,* from 2007 to 2023). For the sake of completeness, we consider also former versions of ICSA, namely: WICSA, QoSA, and CBSE. In total, we obtained 387 potentially-relevant studies from the selected venues.

The second step of phase 2 is the studies selection, where we manually consider each of the 387 potentially-relevant studies and rigorously assess whether it is an MSR4SA study. Following the guidelines for systematic literature review for software engineering [12], we define a *a priori* a set of inclusion and exclusion criteria to reduce the likelihood of bias. We defined the following *inclusion criteria*:

- *The study supports at least one ADA [1]*: architectural analysis (*e.g.,* identify requirements), or architectural synthesis (*e.g.,* identify tactics), or architectural evaluation (*e.g.,* evaluate performance).
- *The study mines a software repository*: This can be either a single software system with many source code versions, or many software systems with one or more versions. Moreover, a software repository can include one or more of the following repositories: issue tracking systems, developers' communities (*e.g.,* Stack Overflow), and other natural language resources such as documentations and web pages.

Moreover, we defined the following *exclusion criteria*:

- The study supports other development tasks (*e.g.,* fixing bugs or testing), and does not support ADAs.
- The study does not apply MSR (*e.g.,* it is based on interviews or surveys).
- The study performs a case study on a single version of an application or uses toy examples to evaluate an approach.
- The study does not provide any information about the mined software repositories (*e.g.,* number of systems, their versions, sizes, etc..).

A study is selected if it satisfies *all* inclusion criteria and *none* of the exclusion criteria. The potentially relevant studies are rigorously examined by adopting multiple selection rounds and the adaptive reading depth method [3]. Specifically, the selection is conducted in three steps by the first and third authors of this paper. First, one of them identifies candidate MSR4SA studies in ICSA and MSR, while the other identifies candidate MSR4SA studies in ECSA. During this step, each study is evaluated as follows: the title of the study is examined, then the abstract is inspected, and finally, the full text is considered to ensure that only the studies relevant for answering the research questions are selected. Second, the researchers switch their roles such that the first one evaluates the ECSA publications and the second one evaluates the ICSA and MSR publications. Similar to the first step, here they also apply the adaptive reading depth method. Third, differences in classifications between both researchers are discussed and full agreement on the papers selected has been established, with the help of the second author acting as arbiter. As a result, we identified 66 MSR4SA papers from the three selected venues: ICSA (39), MSR (15) and ECSA (12).

In the last step of our search and selection phase, we carry out backward and forward snowballing. The reasons for adding the snowballing phase are (1) to widen the scope of our search so to include potentially-relevant studies published in venues different from those we consider in the manual search on DBLP [13] and (2) to benefit from the advantages of the individual search processes (as suggested by Petersen et al. in 2015 [6] and Mourao et al. in 2020 [14]). Specifically, we perform a *closed recursive backward and forward snowballing* procedure [15]. Practically, we analyze each of the 66 studies selected in the manual search on DBLP and we collect all studies either cited by or citing it (based on Google Scholar) [15]. This collection step leads to 2055 additional potentially-relevant studies. Then, we apply the same selection procedure (and criteria) discussed in the previous paragraph to each of the newly-collected studies. We apply snowballing *recursively*, meaning that if a study is included, snowballing is applied iteratively until no new studies are found. Duplicates are removed at each iteration of the snowballing activity. The application of the snowballing procedure led to the selection of 85 additional primary studies, leading to the final set of 151 primary studies considered in this work.

2.3. Data extraction

In this phase, we manually collect data from each primary study, then classify them in the subsequent data synthesis phase. The data extraction phase is performed collaboratively by the first two authors of this study. To have a rigorous process and to ease the extracted data management, we design a well-structured data extraction form upfront.

Table 2 gives an overview of the data extraction form and the covered research questions, as defined in Section 2.1. The first three data items are used for building the demographics about the publication trends of the selected MSR4SA studies (see Section 3). To extract the rest of the data items, we apply a process inspired by the thematic analysis method [16]. Specifically, first two authors of this study extract textual segments from the MSR4SA papers and define themes, while the third author resolves conflicts and reviews the identified themes. We chose thematic analysis as architectural information can be strongly dependent on project-specific and system-specific characteristics and thematic analysis cope well with context-dependent data [17].

We applied the thematic analysis in two main rounds:
(1) Preliminary extraction and themes identification: In this step, the first two authors of this study analyze 20 randomly-selected primary studies to ensure agreement on the data items and their overall themes. Each author analyzes the primary studies independently, identifies relevant textual segments, and defines themes for each data item. Then, they discuss their findings to reconcile differences and determine common themes for each data item. Disagreements between them are discussed together with the third author. Moreover, the third author reviews the data items and the emerging themes to ensure that they are well-formed and clearly defined.
(2) Full data extraction: Based on the defined data items and their themes, the two authors analyze the rest of the primary studies. New themes can emerge from this, which are recorded and discussed to ensure consistency between them. All the resulting themes are explained in Sections Section 4, 5, 6, and 7. Each theme is supported with examples from the primary studies. The full map of primary studies to the themes is provided online [4].

2.4. Data synthesis

The data synthesis phase involves collating and summarizing the data extracted from the primary studies [12] with the main goal of understanding, classifying and summarizing current research on MSR4SA approaches. It is performed by all of the researchers of this study. Specifically, we perform a combination of *content analysis* and *narrative synthesis* [12]. The content analysis consists in a quantitative

Table 2
Information extracted from the articles selected during phase 3.

Item	Description	RQ
Year	Publication year	–
Venue	Publication conference	–
Title	The title of the paper	–
Reason	The reasons of applying MSR techniques	RQ1
Study type	The type of empirical study (either exploratory or evaluation [18])	RQ1
Architectural activity	The supported architectural design activity: analysis, synthesis, and evaluation [1]	RQ1
Data-source type	Types of mined data source (*e.g.,* version control, issue tracker, StackOverflow)	RQ2
Mining techniques	The applied mining techniques (*e.g.,* source code analysis, machine learning)	RQ3
Mined architectural information	The mined architectural information from software repositories (*e.g.,* architectural smells, architectural knowledge)	RQ4

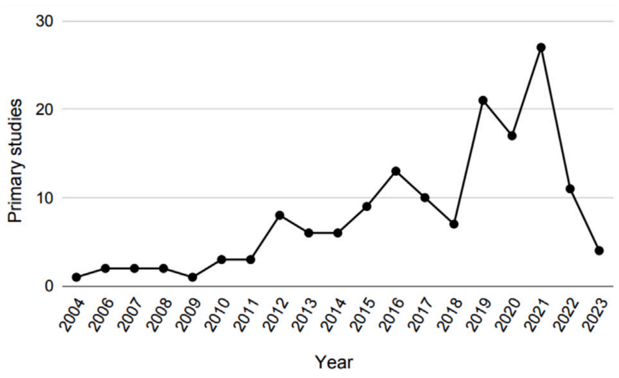


Fig. 2. Number of primary studies per year.

assessment of the extracted data (*e.g.,* the frequency of approaches targeting Stack Overflow compared to those targeting GitHub repositories), while narrative synthesis refers to the systematic method where a textual narrative summary is adopted to (1) explain the quantitative information emerging from the content analysis, and (2) identifying emerging patterns and trends [12]. We group the results of our data synthesis phase according to the research questions of the study and include in our analysis. Then, we used contingency tables to identify interesting co-occurrences across the considered data items.

3. Overview of the selected primary studies

In this section, we provide an overview of the primary studies selected in this research. Fig. 2 illustrates the distribution of the selected primary studies over the years, which among others helps us to understand the historical development of the field and the interest of the Software Engineering community on it. The first primary study in our dataset was published in 2004 [19], where the authors propose a mechanism for formulating metrics-based rules that capture deviations from good design principles. Then, we observe in the graph a trend of at least 1 primary study on MSR4SA per year. The number of primary studies starts to increase considerably in 2012, from when we observe at least 6 publications per year. We can also observe a higher number of primary studies from 2019 to 2021 (counting 21, 17, and 27 primary studies each year, respectively), which is the period where MSR for SA publications reached their peak. Furthermore, in the subsequent years, the number of primary studies decreased considerably, summing to 11 primary studies in 2022 and only 4 primary studies in 2023. This may indicate a mature state of such a research method, or simply a lack of interest from the Software Engineering community, which can be confirmed with further observations in the next years.

Fig. 3 illustrates as a word-cloud the most frequent terms used in the titles of the primary studies, providing an overview of the research topics considered in the MSR4SA studies. Given the essence of this study, some terms in the front of the cloud are expected to be frequent, such as *architecture*, *software/system*, *analysis* and *design*. Other frequent terms are more related to the essence of the study. For instance, *smells* refers to the type of observation (*i.e.,* to identify architectural/code smells), *code* refers to source code repositories, while *evolution* are usually related to identifying architectural decay/erosion.

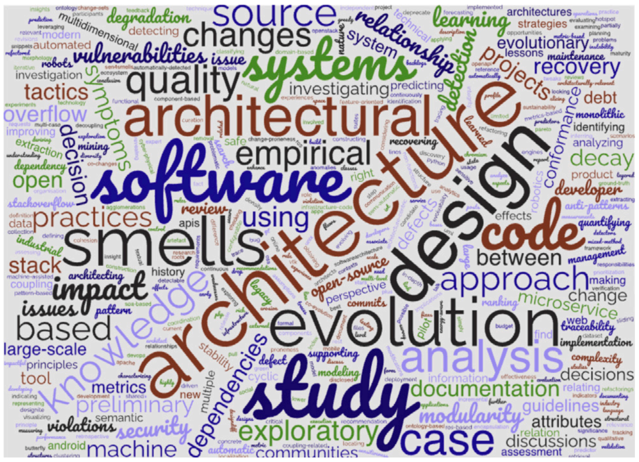


Fig. 3. Recurrent terms in the titles of the primary studies.

Table 3 lists all the venues ranked by the number of primary studies, indicating the potential targets for MSR4SA publications. The first four venues sum up together 66 publications, which corresponds to 43.7% of the total, and are indeed the ones with a higher correlation with this paper research subject. The other 57.3% of the primary studies can be separated into two groups of venues: (1) the ones with more than 1 paper (37.74%), which recurrence indicates a target venue for MSR4SA, and (2) the ones with only one paper published (18.56%), not consider as a potential target venue for this sort of research. We also observe that most of the venues are conferences (76.15%), while three high-impact Software Engineering journals (*i.e.,* EMSE, JSS, and TSE) can also be considered as a target venue, grouping most (75%) of journal publications.

4. Goals of mining software repositories to support software architecture (RQ1)

For answering RQ1 we extracted the reasons for applying MSR techniques to support software architecture (Section 4.1), the type of study (Section 4.2), and the supported architectural design activity (Section 4.3).

4.1. Reasons of MSR4SA

Fig. 4 shows that MSR techniques are applied to support software architecture tasks for several reasons, ranging from achieving quality attributes (*e.g.,* [P7,P8,P9,P10,P11]) to the establishment of a ground truth for an empirical study (*e.g.,* [P4,P12]). Our thematic analysis reveals three main overarching groups of reasons for applying MSR techniques to support architecture: *address industrial concerns*

Table 3
List of venues ranked by the number of papers (* acronym suggested by the authors).

Venue	Type	Acronym	# Papers
International Conference on Software Architecture	Conference	ICSA	26
Mining Software Repositories (MSR) Conference	Conference	MSR	15
Working International Conference on Software Architecture	Conference	WICSA	13
European Conference on Software Architecture	Conference	ECSA	12
Empirical Software Engineering	Journal	EMSE	8
International Conference on Software Engineering	Conference	ICSE	8
Journal of Systems and Software	Journal	JSS	7
Transactions on Software Engineering	Journal	TSE	7
International Conference on Software Maintenance and Evolution	Conference	ICSME	4
Conference on Software Maintenance and Reengineering	Conference	CSMR	4
Working Conference on Reverse Engineering	Conference	WCRE	3
International Symposium on Software Reliability Engineering	Conference	ISSRE	2
International Symposium on Empirical Software Engineering and Measurement	Conference	ESEM	2
Automated Software Engineering	Conference	ASE	2
International Conference on Program Comprehension	Conference	ICPC	2
Science of Computer Programming	Journal	SCICO	2
Brazilian Conference on Software	Conference	SBES	2
Symposium On Applied Computing	Conference	SAC	2
Journal of Software: Evolution and Process	Journal	JSEP*	2
International Conference on Evaluation and Assessment in Software Engineering	Conference	EASE	1
Aspect-Oriented Software Development	Conference	AOSD	1
International Workshop on Bringing Architectural Design Thinking Into Developers' Daily Activities	Conference	BRIDGE	1
International Symposium on Systems Engineering	Conference	ISSE	1
International Conference the Quality of Software Architectures	Conference	QoSA	1
Journal of Software: Evolution and Process	Journal	JSEP	1
Euromicro Conference Series on Software Engineering and Advanced Applications	Conference	SEAA	1
Concurrency and Computation: Practice and Experience	Journal	CPE	1
International Conference on Software Quality, Reliability and Security	Conference	QRS	1
International Conference on Web Services	Conference	ICWS	1
Asia-Pacific Symposium on Internetware	Conference	Internetware	1
International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering	Conference	ECASE	1
Congreso Bienal de Argentina	Conference	ARGENCON	1
SN Computer Science	Journal	SNCS*	1
Asia-Pacific Software Engineering Conference	Conference	APSEC	1
Informatics and Intelligent Applications	Journal	ICIIA	1
Americas Conference on Information Systems	Conference	AMCIS	1
International Working Conference on Variability Modeling of Software-Intensive Systems	Conference	VAMOS	1
Frontiers of Information Technology & Electronic Engineering	Journal	FITEE	1
Security & Privacy	Journal	MSP	1
International Conference on Computer Systems and Applications	Conference	AICCSA	1
International Conference on Open Source Systems	Conference	OSS	1
International Conference on Engineering of Complex Computer Systems	Conference	ICECCS	1
International Conference on Web Engineering	Conference	ICWE	1
Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering	Conference	ESEC/FSE	1
International Conference on Software Analysis, Evolution and Reengineering	Conference	SANER	1
Conference on Reconciling Data Analytics, Automation, Privacy, and Security	Conference	RDAAPS	1
Systems Engineering	Journal	SYS	1

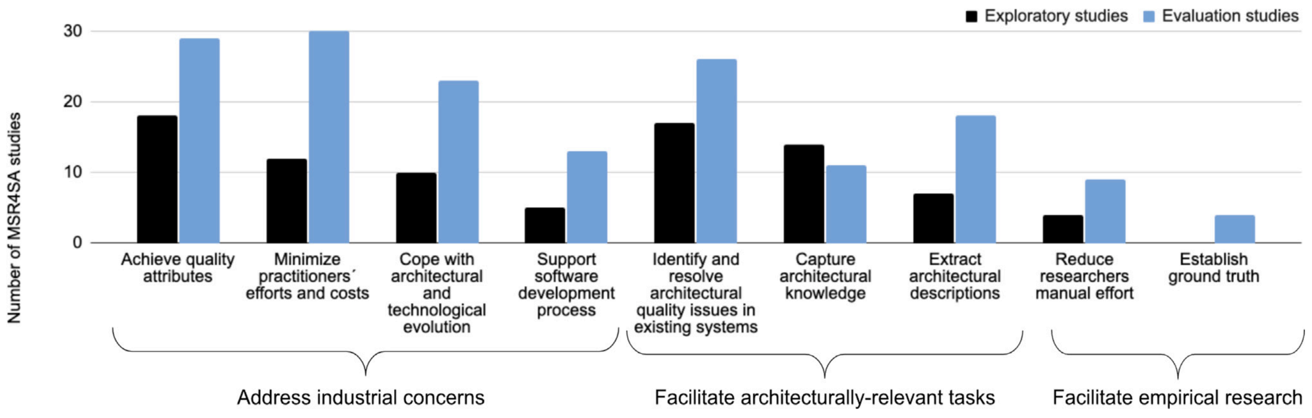


Fig. 4. The number of studies for each MSR4SA reason for exploratory and evaluation studies.

(138/151), *facilitate architecturally-relevant tasks* (91/151), and *facilitate empirical research* (17/151).

Address industrial concerns. The majority of MSR4SA studies (138/151) are motivated by specific industrial concerns, *i.e.,* factors which influence software engineers making architectural design decisions. These concerns could be project constraints (*e.g.,* adopting certain development processes), important quality attribute requirements (*e.g.,* maintainability), or contextual constraints, such as the fast evolution of technologies.

		Address industrial concerns: Achieve quality attributes	Address industrial concerns: Minimize practitioners´ efforts and costs	Address industrial concerns: Cope with architectural and technological evolution	Address industrial concerns: Support software development process	Facilitate architecturally-relevant tasks: Identify and resolve architectural quality issues in existing systems	Facilitate architecturally-relevant tasks: Capture architectural knowledge	Facilitate architecturally-relevant tasks: Extract architectural descriptions	Facilitate empirical research: Reduce researchers manual effort	Facilitate empirical research: Establish ground truth
Address industrial concerns	Achieve quality attributes	5	11	8	4	16	7	7	4	0
	Minimize practitioners´ efforts and costs	11	9	7	8	8	5	3	5	0
	Cope with architectural and technological evolution	8	7	7	2	8	4	5	2	2
	Support software development process	4	8	2	4	3	2	5	2	1
Facilitate architecturally-relevant tasks	Identify and resolve architectural quality issues in existing systems	16	8	8	3	16	2	5	3	2
	Capture architectural knowledge	7	5	4	2	2	5	2	5	0
	Extract architectural descriptions	7	3	5	5	5	2	5	1	3
Facilitate empirical research	Reduce researchers manual effort	4	5	2	2	3	5	1	1	0
	Establish ground truth	0	0	2	1	2	0	3	0	0

Fig. 5. Co-occurrences between the reasons of MSR4SA. The diagonal counts reasons without co-occurrences.

• *Achieve quality attributes (47/151)* — Researchers follow two main strategies to support achieving quality attributes (*e.g.,* re-usability [P13], maintainability [P14], and interoperability [P15]). First, capturing architectural solutions (*e.g.,* patterns [P13] and tactics [P3]) from software repositories (*e.g.,* from source code repositories [P16,P3] or documentation [P13]) supports architects in quantifying and keeping track of the overall quality of their system. Second, MSR techniques are used to identify quality issues (*e.g.,* architectural smells [P17] or violations [P18]) in software repositories.

• *Minimize practitioners' efforts and costs (41/151)* — Practitioners' efforts incur financial costs, which managers try to minimize. For instance, trying to manually document the architecture of an existing system is an expensive activity [P19]. Moreover, evaluating, and refactoring the software architecture of a software system requires significant efforts from various stakeholders [P20]. To save these efforts, researchers apply MSR techniques, which can automate part of practitioner tasks (*e.g.,* via automatic refactoring tools [P21]).

• *Cope with architectural and technological evolution (32/151)* — The fast pace of architectural and technological evolution presents a challenge for practitioners [P22]. This is due to the introduced complexity and limitations of different architectural solutions (*e.g.,* REST [P15], microservices [P23]). To reduce complexity for practitioners,
researchers apply MSR techniques to automatically identify and resolve architectural issues (*e.g.,* architectural violations [P21]) and capture architectural knowledge (*e.g.,* architectural tactics [P3]).

• *Support software development process (18/151)* — Software processes (*e.g.,* agile process [P10], incremental development [P24], and maintenance activities [P12]) have a big influence on the cost and quality of software systems. For instance, agile processes enforce developers to continuously refactor software architectures [P25]. To facilitate this task, in [P24] researchers propose solutions where MSR techniques are applied to determine the costs and benefits of architectural refactoring. Moreover, development processes might enforce using different tools (*e.g.,* backlogs and issue trackers in agile). This makes architectural knowledge scattered in different sources. Thus, MSR4SA studies capture architectural knowledge (*e.g.,* [P5]) from different sources.

Facilitate architecturally-relevant tasks. The second reason for applying MSR techniques is to facilitate conducting certain architecturally-relevant tasks (91/151), specifically:

• *Identify and resolve architectural quality issues in existing systems (42/151)* — As software systems evolve, software engineers might make sub-optimal design decisions or provide limited maintenance efforts. This could lead to architectural issues such as knowledge vaporization [P5], security issues (*e.g.,* vulnerabilities [P26], and architectural erosion symptoms (*e.g.,* architectural smells [P27] and violations [P21]). Tackling these issues requires applying MSR techniques to identify and resolve issues (*e.g.,* through refactoring [P21]).

• *Capture architectural knowledge (25/151)* — MSR4SA studies apply MSR techniques to capture architectural knowledge, such as reusable architectural solutions (*e.g.,* architectural patterns [P13] or tactics [P3]), as well as design decisions extracted from existing systems [P5]. Capturing architectural knowledge depends mostly on textual data sources, such as documentation [P13,P28], Stack Overflow [P22,P29], and issue tracking systems [P5].

• *Extract architectural descriptions (24/151)* — To support maintaining and analyzing a software system, MSR4SA studies apply MSR techniques on software repositories to recover the software architecture of existing systems [20]. In the analyzed MSR4SA studies, the extraction of architectural descriptions involves using architectural recovery techniques on source code repositories to capture the evolution of architectural components and dependencies [P4], as well as using other sources such as issue tracking systems [P4] or practitioners [P12] to capture the rationale of design decisions.

Facilitate empirical research. Software architecture researchers face challenges to apply empirical methods (*e.g.,* manual efforts to analyze big systems). These challenges are tackled by some researchers using MSR techniques (17/151). We identified the following reasons to facilitate empirical research using MSR techniques:

• *Reduce researchers manual effort (13/151)* — To conduct studies, researchers might be required to spend considerable efforts to manually analyze data (*e.g.,* manually classifying architectural issues [P5]). These efforts could be significant and might not be feasible in a limited research budget or time constraints . Thus, researchers in MSR4SA studies apply MSR techniques (*e.g.,* information retrieval [P30]) to save researchers' efforts and time. For instance, applying information retrieval to create a training data set for architectural tactics [P30], or developing shared infrastructure [P31].

• *Establish ground truth (4/151)* — The lack of architectural documentation and rationale of design decisions make it challenging to evaluate proposed approaches (*e.g.,* conformance checking) against a commonly-accepted ground truth. Moreover, it is challenging to access the architects of the systems. Therefore, researchers apply MSR techniques to identify the ground truth of software architecture. For example, Garcia et al. [P12] applied multiple architecture recovery techniques on source code, and mined architectural documentations to recreate the ground truth of architecture.

Fig. 5 shows the co-occurrences between the reasons of MSR4SA studies (*i.e.,* address industrial concerns, facilitate architecturally-relevant tasks, and facilitate empirical research). We can observe that the majority of MSR4SA studies target more than one goal. Moreover, goals of different types (*e.g.,* facilitate architecturally-relevant tasks and address industrial concerns) co-occur often. We can interpret this phenomenon that the goals of MSR4SA studies are rather intertwined. On the one hand, MSR4SA studies aim to achieve practitioners' industrial concerns. On the other hand, they aim to facilitate empirical research.

4.2. Study types

Fig. 4 shows that the majority of MSR4SA studies are evaluation studies. This indicates that MSR4SA researchers are keen to propose solutions for practitioners. On the one hand, most reasons of MSR4SA studies (*e.g., identify and resolve architectural quality issues*) lead to evaluation studies. On the other hand, *achieving quality attributes* and *capturing architectural knowledge* rely on both types of studies (*i.e.,* exploratory and evaluation). This choice between exploratory or evaluation studies might be related to the maturity of existing approaches. For example, approaches for capturing architectural knowledge could be still immature to be used by practitioners. Thus, researchers conduct exploratory studies to have a better understanding of this task. Conversely, approaches to identify and resolve architectural quality issues have industrial tools (*e.g.,* Structure101 and Designite). Thus, researchers tend to propose and evaluate new approaches.

4.3. Architectural design activities

Fig. 6 shows the architectural activities explicitly considered by the primary studies. *Architectural analysis* is significantly the most considered activity in MSR4SA studies, and co-occurring with *identify and resolve architectural quality issues* and *extract architectural description* tasks.

At the second place comes *architectural synthesis*, which co-occur with *capturing architectural knowledge* task. At the least comes *architectural evaluation*, which co-occur with industrial concerns like achieving quality attributes. These results show that architectural synthesis and evaluation require further support from researchers when applying MSR techniques.

RQ1 takeaways: • MSR4SA studies tend to be mostly motivated by a combination of reasons of different types (see Fig. 5). The major reasons to adopt MSR4SA techniques are about addressing industrial concerns, such as *achieving quality attributes* and *minimizing practitioners' efforts* (see Fig. 4). Facilitating empirical research, *e.g., achieving generalizability*, are the least mentioned reasons. • MSR4SA architectural tasks are different in terms of the study type (*i.e.,* exploratory or evaluation studies). Studies for *capturing architectural knowledge* are both exploratory and evaluation studies, while studies for *extracting architecture description* are mostly evaluation studies. • Most MSR4SA studies support architectural analysis, while architectural synthesis and evaluation are rarely supported (see Fig. 6).

5. Data sources mined to support software architecture (RQ2)

Our analysis reveals that MSR4SA researchers are mining a plethora of data sources to support software architecture. We identified the following main data sources:

• *Source code repositories (98/151)* such as in version control systems (*e.g.,* Github), which contain the history of source code changes across multiple versions of a system. Source code repositories are commonly mined to identify architectural quality issues (*e.g.,* smells [P6] or vulnerabilities [P26]), to extract architectural descriptions (*e.g.,* through clustering classes [P4,P12]), or to capture architectural knowledge (*e.g.,* design decisions [P32] or tactics [P3]).

• *Issue tracking systems (43/151)* (*e.g.,* Jira) are used for issue tracking and project management tasks (*e.g.,* new features, fix bugs). They are commonly mined to capture the rationale of design decisions (*e.g.,* [P5,P4]), or to determine correlations between architectural changes and bugs (*e.g.,* [P16,P27,P6]), or to identify suitable team members for making design decisions [P33].

• *Architectural documentation (22/151)* is commonly analyzed to capture architectural solutions (*e.g.,* patterns [P13], tactics [P3] or architectural components [P12]). These studies aim to support achieving re-usability of solutions (*e.g.,* [P13,P3]), and to support architectural program comprehension (*e.g.,* [P12]).

• *Stack Overflow (12/151)* is one of the most popular software forums in industry, and is commonly analyzed in academia. Stack Overflow discussions are commonly analyzed to capture architectural knowledge (*e.g.,* [P22,P29,P23,P34]). Moreover, Stack Overflow is analyzed to understand practitioners' perspective of architectural smells [P35].

• *Requirements (6/151)* are usually analyzed to support architectural analysis design activities [1]. For instance, the authors of [P36] analyze the correlation between requirements, architectural component changes, and the results of test cases. Moreover, requirements tend to be analyzed to understand the architectural analysis process (*e.g.,* [P37]), and to automatically identify architecturally significant requirements (*e.g.,* [P10]).

• *The Web (5/151)* is searched using keywords searching (*i.e.,* information retrieval) to find architectural knowledge such as architectural tactics [P30] and patterns [P38]. The Web is usually mined to find architectural knowledge for specific tasks (*e.g.,* [P2]), or to find architectural knowledge for specific domains (*e.g.,* architectural guidelines for mobile apps [P39]).

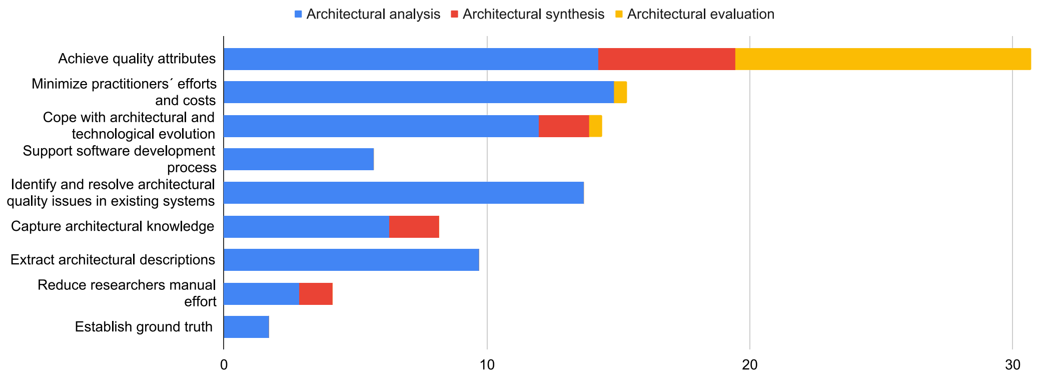
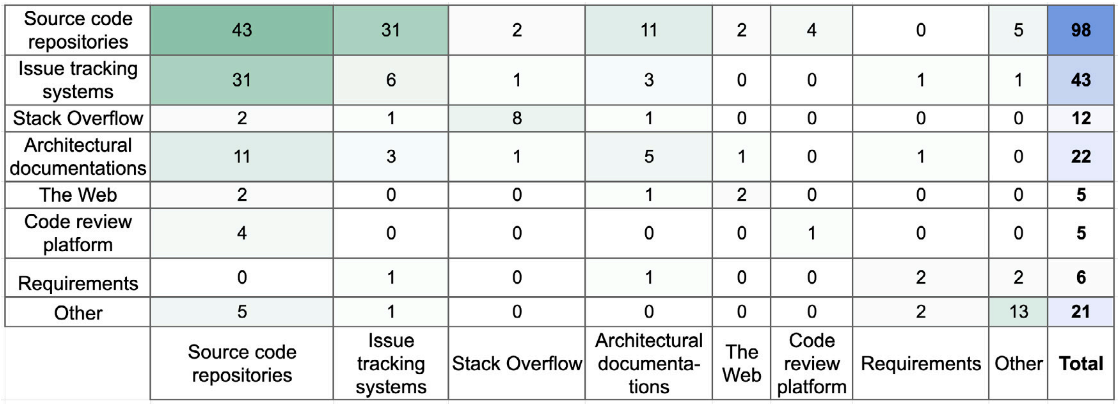
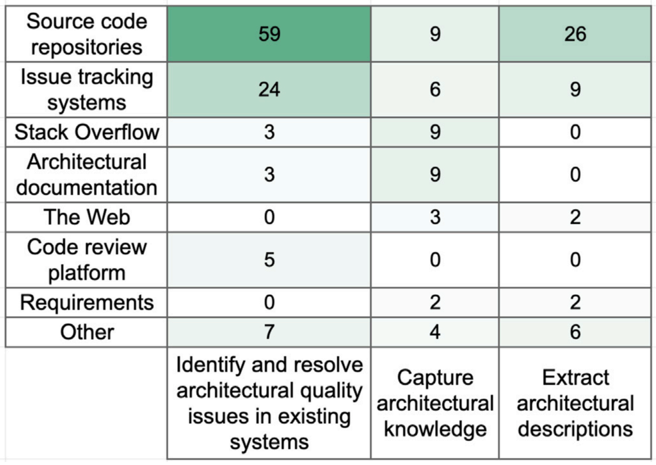


Fig. 6. Distribution of covered design activities, and the goal of their corresponding MSR4SA approaches.



	Source code repositories	Issue tracking systems	Stack Overflow	Architectural documenta-tions	The Web	Code review platform	Requirements	Other	**Total**
Source code repositories	43	31	2	11	2	4	0	5	**98**
Issue tracking systems	31	6	1	3	0	0	1	1	**43**
Stack Overflow	2	1	8	1	0	0	0	0	**12**
Architectural documentations	11	3	1	5	1	0	1	0	**22**
The Web	2	0	0	1	2	0	0	0	**5**
Code review platform	4	0	0	0	0	1	0	0	**5**
Requirements	0	1	0	1	0	0	2	2	**6**
Other	5	1	0	0	0	0	2	13	**21**

Fig. 7. Co-occurrences of data sources in MSR4SA studies. The diagonal counts data sources without any co-occurrences.



	Identify and resolve architectural quality issues in existing systems	Capture architectural knowledge	Extract architectural descriptions
Source code repositories	59	9	26
Issue tracking systems	24	6	9
Stack Overflow	3	9	0
Architectural documentation	3	9	0
The Web	0	3	2
Code review platform	5	0	0
Requirements	0	2	2
Other	7	4	6

Fig. 8. Co-occurrences of data sources with architectural tasks in MSR4SA studies.

• *Code review platforms (5/151)* are software tools or services designed to facilitate the process of reviewing code changes made by developers. This is the case of GitHub/GitLab review tooling (*e.g.,* [21–23]), and ML-based reviews (*e.g.,* [P40]).

• *Other data sources (21/151)* are vulnerability reports (4/151) (*e.g.,* [P26]), API specifications (3/151) (*e.g.,* [P41]), system logs (3/151) (*e.g.,* [P42]), performance logs (2/151) (*e.g.,* [P11]), formal architectural models (2/151) (*e.g.,* [P21]), technology documentations (1/151) (*e.g.,* [P28]), meeting protocols (1/151) (*e.g.,* [P37]), test cases (1/151) (*e.g.,* [P36]), and build traces (1/151) (*e.g.,* [P43]).

Fig. 7 shows co-occurring data sources for the MSR4SA studies. We can observe that most of the MSR4SA studies mine a single data source, where source code repositories are the mined data source in most of the cases, followed by issue tracking systems and Stack Overflow. However, architectural documentation and requirements are mostly mined with other data sources like source code repositories or issue-tracking systems. Moreover, mining both, source code repositories and issue tracking systems, is the most common co-occurrence among data sources. On the other hand, unstructured textual data sources (*i.e.,* issue tracking systems, Stack Overflow, requirements, architectural documents, and the Web) are rarely mined together. This might be due to the complexity and effort required to mine unstructured textual data sources.

Fig. 8 outlines the distribution of data sources for architectural tasks. The two most common data source (*i.e., source code repositories* and *issue tracking systems*) are used for the three architectural tasks, and their usage is also equally ranked, *i.e., identify and resolve architectural quality issues in existing systems*, *extracting architectural descriptions*, and *capture architectural knowledge* task. This indicates a correlation in the purpose of such data sources, which is also corroborated by their co-occurrences in 31 studies (Fig. 7). *Stack Overflow* and *architectural documentations* only contribute with the first two architectural tasks, with a higher occurrence in *capturing architectural knowledge* studies. *The web*, *code review platforms*, and *requirements* play minor roles and distinguish each other in purpose, while *the web* and *requirements* are never used for the *identify and resolve architectural quality issues in existing systems* task, *code review platforms* is only used for that specific task.

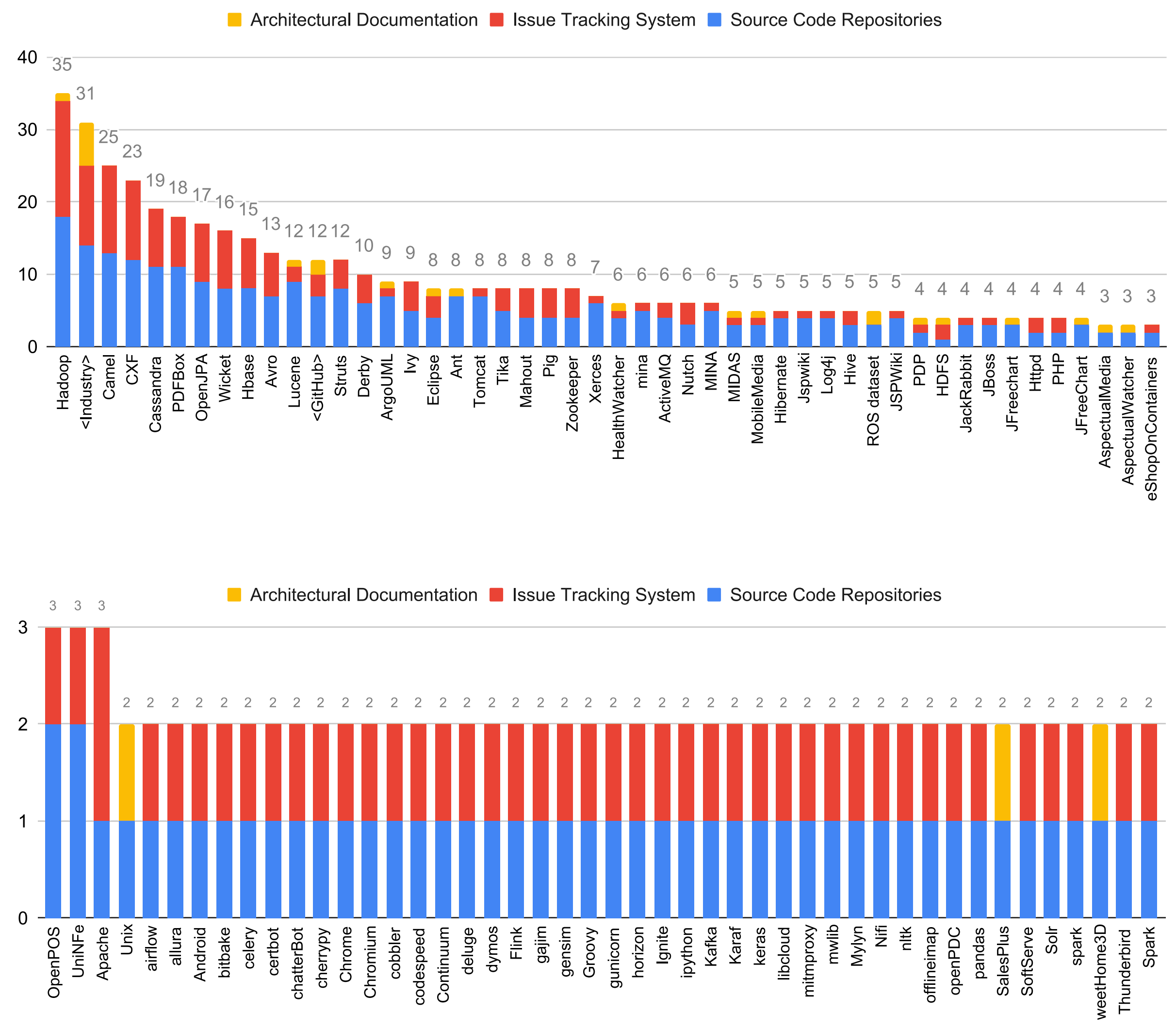


Fig. 9. Case study systems mined from more than one data source.

5.1. *Studied systems*

MSR4SA studies are usually based on specific software systems as case studies. In this mapping study, we identify 354 different software systems, with generic systems being grouped by category, such as: *Industry* groups commercial software, *GitHub* groups public software repositories, *Google Code* groups projects in Google Codelab, and Apache groups generic projects by Apache Foundation.

Out of the 354 different software systems, 94 have their architectures mined from more than one data source, illustrated in Fig. 9. In the figure, we observe that 14 systems stand out as case studies mined from at least 10 data sources: *Hadoop* (35), *Industry* (31), *Camel* (25), *CXF* (23), *Cassandra* (19), *PDFBox* (18), *OpenJPA* (17), *Wicket* (16), *HBase* (15), *Avro* (13), *Lucene* (12), *GitHub* (12), *Struts* (12), and *Derby* (10). *Hadoop* and *Industry* are the the systems studied from the greater number of data sources, counting more than 30 data sources each, and are among the more plural ones, mined from the three data sources. The figure also shows what was discussed in the previous section, with *source code repository* as the most common data source, followed by *issue-tracking system*, while *architectural documentation* is only used in a few studies.

Other 260 systems are objects of architecture study by mining a single data source type. Among them, 41 are present in more than one paper, while most of them (219/260) are only considered in a single study. All the software systems found are listed in a publicly available document.[2]

RQ2 takeaways: • The most frequently mined data sources are *source code repositories* and *issue trackers* (see Fig. 7). • Most of the MSR4SA studies mine a single data source, but *source code repositories* and *issue tracking systems* are commonly mined together. • Most of the case study systems (260/354) have the architecture mined from a single data source type. • Most of the case study systems (218/354) are only studied in a single paper.

6. MSR techniques to support software architecture (RQ3)

MSR4SA studies apply the following families of mining techniques:

• *Source code analysis (66/151)* is the most commonly applied mining technique. This involves applying common static program analysis techniques (*e.g.,* creating syntax trees) and building up graphs (*e.g.,* dependency and control graphs) among packages, classes, and methods. Extracted graphs are then analyzed to extract architectural

[2] https://github.com/S2-group/msr4sa-systematic-mapping-study-rep-pkg/blob/main/data/system_names.csv

descriptions (*e.g.,* capture components using clustering – [P4]) or to identify architectural quality issues (*e.g.,* security vulnerabilities or smells – [P44,P27]) in the source code of the system.

• *Model analysis (60/151)* is commonly used to analyze architectural models, usually captured from source code repositories. It supports identifying architectural components [P18,P43], calculating metrics on models [P15], predicting system quality [P45] and identifying architectural quality issues such as architectural violations [P21] and smells [P6].

• *Statistical analysis (49/151)* techniques such as descriptive statistics and correlation analysis methods (*e.g.,* Pearson and Spearman) are common techniques in the MSR4SA studies to summarize architectural quality metrics (*e.g.,* [P46]) and to determine the correlation of changed architectural elements and issues such as security vulnerabilities [P9] and bugs [P7,P47].

• *Machine learning (44/151)* is used to mine source code and unstructured textual information such as issue tracking and Stack Overflow. Unsupervised machine learning (*e.g.,* clustering) is commonly used to identify architectural components (*e.g.,* [P48]) from source code. In contrast, supervised machine learning is commonly used to capture architectural knowledge from unstructured textual information [P22, P5,P49], and to predict components changes [P40] and architectural issues [P50].

• *Qualitative analysis (42/151)* techniques such as grounded theory [24] and thematic analysis [16] are methods to manually analyze unstructured textual contents as in architectural documentation, issue tracking systems, and Stack Overflow. The most common reason for applying qualitative analysis in MSR4SA studies is to explore architectural knowledge in Stack Overflow [P23,P29], architectural documentations [P13], and issue tracking systems [P5].

• *Natural language processing (NLP) (26/151)* is commonly used to pre-process textual information for supervised machine learning analysis. For example, NLP pre-processes Stack Overflow posts [P22], and issues in issue tracking systems [P5] to automatically identify architectural knowledge.

• *Keywords searching (11/151)* approaches (*i.e.,* information retrieval) is an effective and easily applied technique to find relevant architectural information in source code [P30,P3], as well as in unstructured textual contents like Stack Overflow [P22] and web resources [P2]. Despite the simplicity and effectiveness of information retrieval, researchers complemented information retrieval with qualitative analysis to mitigate false positives in results [P3].

Fig. 10 shows that the majority of MSR4SA studies apply more than one mining technique[2]. We see that qualitative analysis and statistical analysis are commonly applied isolated. On the other hand, model analysis are conducted in conjunction with source code analysis. For example, to identify architectural quality issues (*e.g.,* architectural smells), the authors of [P6] first apply source code analysis on multiple repositories and then perform model analysis to capture architectural smells.

RQ3 takeaways: • Most of the MSR4SA studies apply more than one mining technique, where the most common MSR techniques are: *source code analysis*, *model analysis*, *statistical analysis*.

7. Architectural information mined from software repositories (RQ4)

As shown in Fig. 11, MSR4SA researchers are mining the following architectural information from software repositories:

• *Architectural components traceability (20/151)* involve multiple versions of a system to identify architectural components, *e.g.,* Java modules [P51] or microservices [P52]), and trace their associated design decisions in issue trackers and architectural documentation. This has been done qualitatively with the help of practitioners (*e.g.,* [P12]), and quantitatively using information retrieval (*e.g.,* [P53]) and statistical analysis (*e.g.,* [P54].

• *Architectural quality issues (14/151)* such as architectural smells [P1], *e.g.,* cyclic dependencies, God components, and Hotspot patterns [P27]), performance issues (*e.g.,* [P11]) and security vulnerabilities (*e.g.,* uncontrolled resource consumptions [P26]. For example, in [P9] the authors identified correlations between architectural smells and vulnerability issues.

• *Re-usable architectural solutions (14/151)* such as architectural patterns (*e.g.,* layers) [P13,P55,P41] or tactics, such as availability tactics (*e.g.,* ping echo) [P16] and maintainability tactics (*e.g.,* increase cohesion) [P56]. For example, in [P3] the authors identified energy efficiency tactics from different MSR data sources.

• *Metrics (11/151)* such as interoperability [P15] and maintainability metrics (*e.g.,* coupling [P57] and cohesion [P58]) are calculated to achieve quality attributes. Moreover, new architectural metrics are proposed, such as in [P36], where two metrics to measure the traceability between requirements and architectural components are explored.

• *Architectural design decisions (ADDs) (11/151)* explain architectural solutions and the rationale of their selection. This can also include alternative solutions, as well as the benefits and drawbacks of each solution. For example, [P22] propose an approach to effectively find ADDs in Stack Overflow.

• *Correlations (9/151)* provide evidence to understand certain architectural concepts (*e.g.,* tactics or architectural smells). For example, in [P16] the correlations between tactics, metrics, and bug fixes are calculated, while in [P9], the correlation between architectural smells and security issues is calculated. Moreover, in [P59], correlations between different metrics, such as, size, stability, and encapsulation are calculated to test design principles.

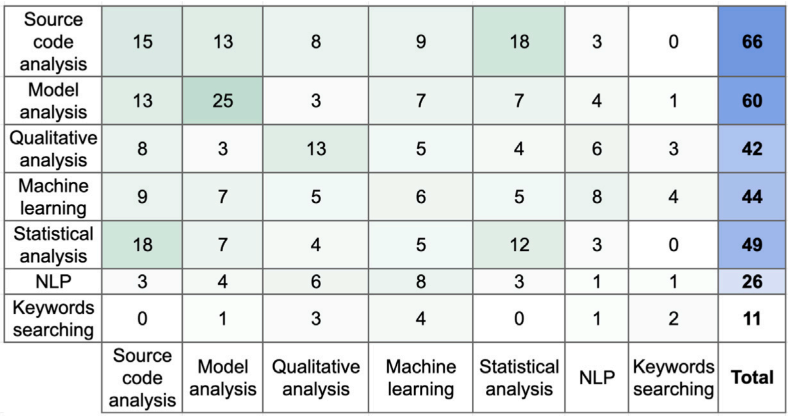
Fig. 11 shows the number of MSR4SA studies which mine each of the aforementioned architectural information types, and their co-occurrences with the data sources[2]. We can observe that *architectural quality issues* is the most mined type of architectural information. This is followed by *architectural components traceability*. We can also observe that *source code repositories* and *issue tracking systems* were mined to capture all types of architectural information. While other data sources are limited to certain types of architectural information. For example, *Stack Overflow* is mined to capture *ADDs* [P22], *architectural solutions* like microservices [P23] and *architectural quality issues* like smells [P35].

RQ4 takeaways: • *Architectural quality issues* and *components* are the mostly mined type of information. • *Source code repositories* and *issue tracking systems* are mined for all types of architectural information. Other data sources like Stack Overflow or Requirements are mined for specific information (*e.g.,* ADDs).

8. Discussion

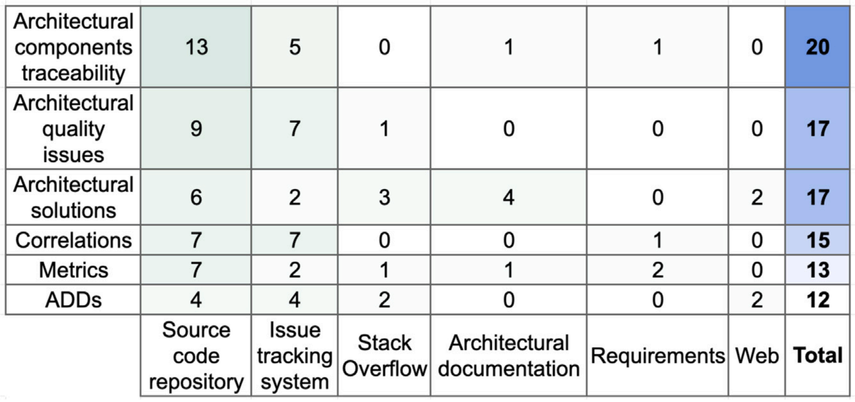
8.1. Implications for researchers

The goals identified in RQ1 (see Section 4) provide an overview of the goals currently achieved for the MSR4SA studies, which can help identify gaps and set priorities for future MSR4SA studies. For example, goals related to *facilitating empirical research*, such as *establishing ground truth* and *reduce researchers manual effort*, seem to be rarely considered (see Fig. 4); therefore, they deserve extra attention from researchers in future MSR4SA studies. Furthermore, our results in Fig. 4 show that most of the goals are achieved through evaluation studies, while only one goal, *i.e., capture architectural knowledge*, is achieved primarily using exploratory studies. These results imply that more exploratory



	Source code analysis	Model analysis	Qualitative analysis	Machine learning	Statistical analysis	NLP	Keywords searching	Total
Source code analysis	15	13	8	9	18	3	0	**66**
Model analysis	13	25	3	7	7	4	1	**60**
Qualitative analysis	8	3	13	5	4	6	3	**42**
Machine learning	9	7	5	6	5	8	4	**44**
Statistical analysis	18	7	4	5	12	3	0	**49**
NLP	3	4	6	8	3	1	1	**26**
Keywords searching	0	1	3	4	0	1	2	**11**

Fig. 10. Co-occurrences of mining techniques in MSR4SA studies. The diagonal counts techniques without co-occurrences.



	Source code repository	Issue tracking system	Stack Overflow	Architectural documentation	Requirements	Web	Total
Architectural components traceability	13	5	0	1	1	0	**20**
Architectural quality issues	9	7	1	0	0	0	**17**
Architectural solutions	6	2	3	4	0	2	**17**
Correlations	7	7	0	0	1	0	**15**
Metrics	7	2	1	1	2	0	**13**
ADDs	4	4	2	0	0	2	**12**

Fig. 11. The (co)-occurrences of architectural information types and data sources in MSR4SA studies.

studies are needed to achieve the MSR4SA goals. In another implication, the results could refer to the preferred methodologies followed by MSR4SA researchers, which support new researchers with standard methodologies for future MSR4SA studies.

The identified co-occurrences between the goals of MSR4SA studies (see Fig. 5) provide a comprehensive map of existing co-occurrences of goals in current MSR4SA studies, which support to determine ideas for new studies that address multiple goals. For example, the two goals *capture architectural knowledge* and *support development process* rarely cooccur in current studies. However, both goals are conceptually related because software developers capture architectural knowledge throughout the software process. Thus, a promising idea for a new study could be to explore how architectural knowledge is generally captured throughout the software development process.

The data sources found in RQ2 and their cooccurrences (see Fig. 7) provide a guide for researchers on the most commonly mined data sources (*e.g., source code repositories*, and *issue tracking systems*), as well as less used data sources. On the one hand, researchers could reuse existing datasets from commonly mined data sources, such as the one containing 335 GitHub repositories containing robotic systems in [P3]. On the other hand, exploring rarely mined data sources might bring new findings and provide additional research opportunities for researchers. The Web and meeting minutes are two examples of promising but still rarely-used data sources. Also, based on our primary studies, *mailing lists* have not been previously mined for software architecture. In addition, most studies are based on a single data source, rather than multiple sources. Although rarely mined data sources, and their co-occurrences could present an opportunity for future MSR4SA studies, it might indicate challenges in terms of data collection and analysis to mine certain data sources or mine multiple data sources together. One challenge could be the lack of a reliable infrastructure (*e.g.,* [25]) to efficiently conduct MSR4SA studies using multiple data sources and for different architectural tasks.

The choice of a data source to execute architectural tasks shows to be mostly influenced by one or two data sources. For instance, *source code repositories* and *issue tracking systems* are the favorite data sources to *identify and resolve architectural quality issues in existing systems* and *extract architectural descriptions*. This result facilitates future MSR4SA studies, as researchers know that source code and issue trackers have been successfully mined in previous MSR4SA studies. In contrast, this result can also motivate researchers to explore rarely-mined data sources for certain tasks. For example, *architectural documentations* and *issue tracking systems* have been rarely mined together. Using both data sources, researchers could support multiple the tasks: *capture architectural knowledge* and *extract architectural descriptions*.

In our study, we identified 354 software systems mined from the primary studies and their data sources (see Section 5.1). These could guide researchers in new MSR4SA studies in selecting the software systems to mine. Furthermore, the list of software systems can support the development of benchmarks, such that when researchers mine the same software systems for a specific data source, they could evaluate their approaches and compare their results among different approaches.

Our results for RQ3 provide an overview for researchers on successfully applied mining techniques. For example, it is common to apply code analysis, model analysis, and statistical analysis in source code repositories and issue trackers. This is a confirmation that these techniques can be successfully (re)used by researchers in future MSR4SA studies. Moreover, the results of RQ3 also show rarely applied mining techniques, such as NLP (see Fig. 10). NLP in MSR4SA studies is only used to preprocess text for machine learning, with little profit from NLP capabilities. Consequently, researchers could explore additional capabilities of NLP such as applying information extraction techniques (*e.g.,* argumentation mining), question-answering, or topic modeling [8].

The results of RQ4 provide a map of the architectural information that can be extracted via MSR techniques (and from which data sources). For example, if a researcher needs to capture architectural quality issues like architectural smells, then they can consider mining source code repositories, issue tracking systems, and Stack Overflow in the first place. Moreover, we also identified types of architectural information that are *not* mined from certain data sources. For example, architectural components are not mined from Stack Overflow or from the Web. This is expected since the components of a specific system are generally stored in a system-specific software repository (*e.g.,* its GitHub repository or its JIRA board). However, Stack Overflow and other Web resources (*e.g.,* blogs [P60]) describe the components of software systems in a generic way [P29], rather than using terms specific to a software system, as in issue trackers [P61]. The advantage of such generic description facilitates finding similar components and design issues, and accordingly facilitates the re-use of architectural knowledge.

Our results can foster the collaboration between MSR and SA researchers. Given the number of studies and their different venues (see Table 3), researchers from both communities could identify potential collaborators working on similar topics and work with them for future MSR4SA studies.

Our study can be considered as a stepping stone for subsequent literature reviews focusing on specific aspects of MSR4SA techniques, where other researchers can empirically evaluate what and how existing MSR4SA techniques can achieve architectural tasks. For example, researchers could conduct systematic literature reviews to determine which source code analysis is more effective to extract architectural knowledge, or which machine learning algorithms achieve the highest accuracy to detect architectural smells, or whether issue trackers or source code are better sources for effectively extracting architectural information to be used in the architectural synthesis phase.

8.2. Implications for practitioners

The goals that emerged in RQ1 show that practitioners' concerns, such as efforts and development process, are widely considered in MSR4SA studies. The identified goals inform practitioners about the problems at hand that are more compatible with MSR techniques. Those can guide practitioners about how to address some of their (architecturally-relevant) concerns. For example, adopting agile development processes might force developers to use different tools (*e.g.,* backlogs and chat tools), while scattered architectural knowledge could be captured using MSR techniques based on keyword-based searches or machine learning.

The number of primary studies reported in Fig. 6 provide practitioners with an overview of recent research trends to achieve architectural design activities: analysis, synthesis, and evaluation. These activities are quite relevant for practitioners, and therefore these results in Fig. 6 inform practitioners about available approaches to support architectural design activities. For example, practitioners can already benefit from the numerous approaches to support architectural analysis, such as identifying quality issues or capturing architectural knowledge in an existing system.

The results of RQ2 inform practitioners about available data sources that might be successfully mined in the context of their projects or organizations. Although we expect that practitioners are familiar with many of these data sources (*e.g.,* GitHub), they may not be fully aware of the benefits of *mining* them. Thus, the results of RQ2 clarify the benefits of mining various data sources for architectural purposes. For example, one of our results is that source code repositories and issue tracking systems are usually mined together; this can motivate practitioners to associate architectural changes from source code repositories with issues in issue tracking systems.

The MSR techniques identified in RQ3 are useful for guiding practitioners about the development methods and tools required to mining software repositories. For example, finding and capturing ADDs from issue trackers can be achieved well using keyword-based searches and NLP techniques. Thus, practitioners might develop tools for their repositories using existing technologies (*e.g.,* Apache Lucene and NLTK library) to find hidden ADDs in issue tracking systems. Furthermore, practitioners can benefit from the tools proposed within some of the primary studies, which could be applied in practice. For example, practitioners can benefit from new source code analysis tools to detect architectural smells [P1].

The map of architectural information extracted from software repositories (see Fig. 11) informs practitioners about what they can expect from each data source and motivates them to search architectural information in (possibly unexpected) data sources. For example, practitioners might not be aware that design decisions are shared in Stack Overflow and issue trackers as well.

9. Limitations and threats to validity

Considering only publications at specific venues may have reduced the general validity of our initial search on DBLP. We are aware that other software engineering conferences (*e.g.,* ICSE and ESEC/FSE) and journals (*e.g.,* TSE, JSS) might have published studies about MSR4SA approaches. As we discussed in Section 2.2, in this study we decided to give higher priority to the correctness of the selected studies in the first manual search on DBLP and then to complement it via backward/forward snowballing. In this context, previous experience tells us that DBLP is a reliable publication database for studies at the intersection between different research areas, which then can be used as starting set for snowballing [26,27]. In summary, by borrowing the terminology from the information retrieval field, in the initial search of this study we selected the primary studies by preferring precision over recall, while keeping the focus on top-quality conferences centered around the MSR4SA topic. Indeed, it is common sense that ICSA, ECSA, and MSR are the top conferences where to publish work related to software architecture and MSR work. This has been a recurrent practice also by other authors [28,29], where specific venues are considered to publish a *representative* sample of the state of the art on a particular topic.

Multiple phases of our study rely on manual analysis, potentially leading to biased results. We mitigate this threat by following a solid and iterative SMS method, extensively discussed *a priori* among the researchers. Indeed, the three researchers were involved in all of the phases, and each phase has been validated by high agreement levels among them. The criteria for identifying MSR4SA studies could be a threat to construct validity, because software architecture researchers may not explicitly label MSR studies. To mitigate this potential threat to validity, we (1) defined a priori a set of precise selection criteria for MSR4SA studies, (2) applied them to all studies, and (3) assessed the inter-rater agreement on a common sample of studies. Finally, we applied well-known guidelines and best practices in each phase of the study and we report the data managed in each phase in a publicly available research protocol, thus making our study easy to be replicated and verified.

10. Related work

There are a few studies in the literature where authors investigate the use of MSR techniques for software architecture. However, their scope tends to be narrower than that of our study, *i.e.,* they focus on a single facet of software architecture. For example, Bi et al. [31] focus only on text analysis techniques for mining architecture-related artifacts. Javed and Zdun [32] consider only the literature on extracting

Table 4
Comparison between the contributions of this study and contributions in the study by Jean de Dieu et al. [30].

	This study	Jean de Dieu et al. [30]
Number of studies and period	151 (from 2004 to 2023)	104 (from 2006 to 2022)
RQ1 Goals of MSR4SA		
Address industrial concerns	✓	–
Considered architectural tasks	1. Identify and resolve architectural quality issues. 2. Capture architectural knowledge. 3. Extract architectural description.	1. Architecture understanding, 2. Architecture maintenance and evolution. 3. Architecture recovery. 4. Architectural description and documentation. 5. Architecture implementation. 6. Architecture re-use. 7. Architecture impact analysis. 8. Architecture conformance check.
Facilitates empirical research	✓	–
Co-occurences between goals	✓	–
Study types (Exploratory versus evaluation)	✓	–
Architectural design activities	✓	✓
RQ2 Data sources for MSR4SA		
Source code repositories	✓	✓→ ranked 1st
Issue tracking systems	✓	✓→ ranked 4th
Architectural documentations	✓	✓→ rank not specified
Stack Overflow	✓	✓→ ranked 2nd
Requirements	✓	✓→ rank not specified
The Web	✓	✓→ ranked 10th
Code review platform	✓	–
Other sources	7. Vulnerability reports 8. API specifications 9. System logs. 10. Byte code. 11. Mailing lists. 12. Formal models. 13. Meeting protocols. 14. Test cases. 15. Blogs 16. Build traces.	3. Wiki 5. Online books and articles. 6. Blogs 7. Chat messages. 8. Mailing lists. 9. App stores.
Co-occurences between data sources	✓	–
Co-occurences between data sources and tasks	✓	✓
Studied systems in MSR4SA studies	✓	–
RQ3 Mining techniques for MSR4SA		
Source code analysis	✓	–
Model analysis	✓	–
Statistical analysis	✓	–
Machine learning	✓	✓→ ranked 1st
Qualitative analysis	✓	✓→ ranked 3rd
Natural language processing	✓	✓→ rank not specified
Keywords searching	✓	✓→ ranked 2nd
Co-occurences among mining techniques	✓	–
Tools employed in studies	–	✓
RQ4 Architectural information from repositories		
Architectural components traceability	✓	✓→ ranked 1st
Architectural quality issues	✓	✓→ ranked 5th
Re-usable architectural solutions	✓	✓→ ranked 2nd
Metrics	✓	–
Architectural design decisions	✓	✓→ ranked 3rd
Correlations	✓	–
Other information in Jean de Dieu et al. [30] but not in this study	System requirement → ranked 3rd, architectural change → ranked 6th, design relationship → ranked 7th	
Co-occurences with data sources	✓	–
Other contributions in Jean de Dieu et al. [30] but not in this study	Challenges in mining architectural information, such as lack of approaches, tools, and datasets.	

software architecture from source code. Bhat et al. [33] conduct a semi-systematic literature review exclusively on design decisions. Finally, Sinka et al. [34] evidences the architecture recovery techniques specifically for software product lines. Compared to all of them, the scope of our study is broader since we consider a comprehensive set of MSR techniques, instead of a single one.

We also found multiple systematic mapping studies (SMS) and systematic literature reviews (SLR) for either software architecture or MSR techniques. However, these studies do not combine both MSR and software architecture. We discuss these in the following two paragraphs.

Software architectural researchers conducted SMS and SLR with a focus on specific aspects of software architecture. For instance, Qureshi et al. [35] synthesize trends, patterns, and knowledge gaps on software architecture using an SLR on empirical software engineering studies. Yang et al. [36] conduct an SMS for analyzing the combination of agile methods and software architecture practice. Galster and Weyns [18] conduct an SMS to explore the application of empirical research in software architecture. Aletie et al. [37] analyze the literature on approaches that propose techniques for searching optimal architecture design. Arshad and Usman [38] systematically synthesize the quantity and type of work that has been reported on security at software architectural level. Finally, Tang et al. [39] study the human aspects involved in software architecture decision making.

In the case of MSR, especially due to its specific nature and recent background, the related work are not that numerous. We find three main work that aim at investigating the MSR strategies and its evolution. Farias et al. [29] conduct a systematic mapping study on the area of MSR aiming at understanding how this field has evolved. On the occasion of the 10th anniversary of the MSR conference, in 2013, Demeyer et al. [28] report what past MSR papers reflect, and discuss the possible future of such methodology. Also in 2013, Hemmati et al. [40], provide a broad scope of the MSR community and summarize the best practices in the past decade of MSR research.

The closest SMS to this study is the recent work done by Jean de Dieu et al. [30]. The authors conduct an SMS on mining *architectural information* with focus on categories of architectural information, sources, activities, approaches, tools, and challenges. Because the study from Jean de Dieu et al. [30] is the closest to this study, we compare the contributions of both papers in Table 4, and discuss the differences below:

- *Goal*: Both studies have two different but intersecting goals. Jean de Dieu et al. [30] focus on architectural information, independent if these are in software repositories or not. In contrast, we focus on mining software repositories to support software architecture.
- *Primary studies*: Jean de Dieu et al. [30] found 104 primary studies, while we found 151 primary studies, with only 49 studies intersect between both SMSs.
- *Results*: There are several similarities and differences among both studies presented in Table 4. For goals of MSR4SA, both studies found different goals. Jean de Dieu et al. [30] focus on architectural activities and tasks, while our study consider other goals, such as industrial concerns and empirical research. Regarding data sources, both studies found similar and different data sources and with different priorities. For instance, issue trackers is the second data source in our primary studies, while in Jean de Dieu et al. [30], it is the 4th data source after Stack Overflow and Wikis. Regarding mining techniques, Jean de Dieu et al. [30] focus on machine learning, keywords searches and qualitative analysis, while other techniques such as source code analysis and statistical analysis were not considered. Also, both studies found similar and different architectural information. For instance, Jean de Dieu et al. [30] did not consider metrics and correlations as architectural information, while they consider other information, such as system requirements and architectural change. Finally, Jean de Dieu et al. [30] identified tools used to mine architectural information, while we identified analyzed software systems.

All in all, due to the difference in goals between both studies, different primary studies were found and different results were obtained, and accordingly different conclusions. Therefore, we argue that both studies are different, do not contradict, and complement each other in several aspects [P62–P151].

11. Conclusion and future work

This paper presents the first systematic mapping study on MSR techniques for software architecture. From 2442 potentially-relevant studies extracted from the proceedings of MSR, ICSA, and ECSA we selected 151 studies and analyzed these to devise an evidence-based map of MSR4SA goals, data sources, MSR techniques, and extracted architectural information. The results of this study benefit both researchers willing to contribute to the area of MSR4SA and practitioners willing to understand existing MSR4SA research.

In the future, we are planning to (1) carry out a mixed-method study targeting practitioners via case studies and a systematic study of the grey literature in order to build an objective map of their needs and practices in concrete projects involving MSR techniques for software architecture, (2) target a selection of the identified research gaps, with a special interest on the application of large language models for supporting topic modeling activities [41] and, more generally, (3) applying generative AI techniques for supporting the extraction of architectural information and architectural reasoning, which has been identified by the software engineering community as one of the most glaring research gaps to be tackled in future software engineering studies [42,43], with only few preliminary studies emerging in the last year [44,45].

CRediT authorship contribution statement

Mohamed Soliman: Writing – original draft, Visualization, Software, Methodology, Data curation, Conceptualization. **Michel Albonico:** Writing – original draft, Software, Methodology, Data curation, Conceptualization. **Ivano Malavolta:** Writing – review & editing, Supervision, Methodology, Investigation, Conceptualization. **Andreas Wortmann:** Writing – review & editing, Supervision.

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.

Data availability

The full replication package of the study is available on Zenodo: https://doi.org/10.5281/zenodo.14614882.

References

[1] C. Hofmeister, P. Kruchten, R.L. Nord, H. Obbink, A. Ran, P. America, A general model of software architecture design derived from five industrial approaches, J. Syst. Softw. (2007).

[2] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, third ed., Addison-Wesley Professional, 2012.

[3] K. Petersen, S. Vakkalanka, L. Kuzniarz, Guidelines for conducting systematic mapping studies in software engineering: An update, Inf. Softw. Technol. (2015).

[4] M. Soliman, M. Albonico, I. Malavolta, A. Wortmann, Replication package of this study, 2024, 10.5281/zenodo.14614882.

[5] C. Wohlin, P. Runeson, M. Host, C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering: An Introduction, Kluver Academic, 2000.

[6] K. Petersen, S. Vakkalanka, L. Kuzniarz, Guidelines for conducting systematic mapping studies in software engineering: An update, Inf. Softw. Technol. 64 (2015) 1–18.

[7] V.R. Basili, H.D. Rombach, The TAME project: Towards improvement-oriented software environments, IEEE Trans. Softw. Eng. 14 (6) (1988) 758–773.

[8] G. Bavota, Mining unstructured data in software repositories: Current and future trends, in: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, Vol. 5, SANER, IEEE, 2016, pp. 1–12.

[9] D. Binkley, Source code analysis: A road map, in: Future of Software Engineering, FOSE'07, IEEE, 2007, pp. 104–119.

[10] F.G. de Oliveira Neto, R. Torkar, R. Feldt, L. Gren, C.A. Furia, Z. Huang, Evolution of statistical analysis in empirical software engineering research: Current state and steps forward, J. Syst. Softw. (2019).

[11] Conference CORE Ranking, 2024, URL http://portal.core.edu.au/conf-ranks.

[12] K. Staff, Guidelines for Performing Systematic Literature Reviews in Software Engineering, Technical Report, ver. 2.3 ebse technical report. ebse, 2007.

[13] C. Wohlin, Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14, Association for Computing Machinery, New York, NY, USA, ISBN: 9781450324762, 2014, http://dx.doi.org/10.1145/2601248.2601268.

[14] E. Mourão, J.F. Pimentel, L. Murta, M. Kalinowski, E. Mendes, C. Wohlin, On the performance of hybrid search strategies for systematic literature reviews in software engineering, Inf. Softw. Technol. 123 (2020) 106294.

[15] C. Wohlin, Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, 2014, pp. 1–10.

[16] V. Braun, V. Clarke, Using thematic analysis in psychology, Qual. Res. Psychol. 3 (2) (2006) 77–101.

[17] D.S. Cruzes, T. Dyba, Research synthesis in software engineering: A tertiary study, Inf. Softw. Technol. (2011).

[18] M. Galster, D. Weyns, Empirical research in software architecture: How far have we come? in: 2016 13th Working IEEE/IFIP Conference on Software Architecture, WICSA, IEEE, 2016, pp. 11–20.

[19] R. Marinescu, D. Ratiu, Quantifying the quality of object-oriented design: the factor-strategy model, in: Proceedings 11th Working Conference on Reverse Engineering, WCRE 2004, IEEE, 2004, pp. 192–201, http://dx.doi.org/10.1109/WCRE.2004.31.

[20] J. Garcia, I. Ivkovic, N. Medvidovic, A comparative analysis of software architecture recovery techniques, in: 2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013 - Proceedings, ISBN: 9781479902156, 2013, pp. 486–496, http://dx.doi.org/10.1109/ASE.2013.6693106.

[21] M. Paixao, J. Krinke, D. Han, C. Ragkhitwetsagul, M. Harman, The impact of code review on architectural changes, IEEE Trans. Softw. Eng. 47 (5) (2021) 1041–1059, http://dx.doi.org/10.1109/TSE.2019.2912113.

[22] R. Li, M. Soliman, P. Liang, P. Avgeriou, Symptoms of architecture erosion in code reviews: A study of two OpenStack projects, in: 2022 IEEE 19th International Conference on Software Architecture, ICSA, IEEE Computer Society, Los Alamitos, CA, USA, 2022, pp. 24–35, http://dx.doi.org/10.1109/ICSA53651.2022.00011, URL https://doi.ieeecomputersociety.org/10.1109/ICSA53651.2022.00011.

[23] R. Morales, s. McIntosh, F. Khomh, Do code review practices impact design quality? A case study of the Qt, VTK, and ITK projects, in: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering, SANER, 2015, pp. 171–180, http://dx.doi.org/10.1109/SANER.2015.7081827.

[24] J. Corbin, A. Strauss, Basics of Qualitative Research, vol. 14, SAGE, 2015.

[25] J. Garcia, M. Mirakhorli, L. Xiao, S. Malek, R. Kazman, Y. Cai, N. Medvidović, SAIN: A community-wide software architecture INfrastructure, in: 2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings, ICSE-Companion, 2023, pp. 336–337, http://dx.doi.org/10.1109/ICSE-Companion58688.2023.00095.

[26] M. Albonico, M. Dordevic, E. Hamer, I. Malavolta, Software engineering research on the robot operating system: A systematic mapping study, J. Syst. Softw. (ISSN: 0164-1212) 197 (2023) 111574, http://dx.doi.org/10.1016/j.jss.2022.111574.

[27] E. Alberts, I. Gerostathopoulos, I. Malavolta, C. Hernández Corbato, P. Lago, Software architecture-based self-adaptation in robotics, J. Syst. Softw. (ISSN: 0164-1212) 219 (2025) 112258, http://dx.doi.org/10.1016/j.jss.2024.112258.

[28] S. Demeyer, A. Murgia, K. Wyckmans, A. Lamkanfi, Happy birthday! a trend analysis on past MSR papers, in: 2013 10th Working Conference on Mining Software Repositories, MSR, IEEE, 2013, pp. 353–362.

[29] M. de F. Farias, R. Novais, M.C. Júnior, L.P. da Silva Carvalho, M. Mendonça, R.O. Spínola, A systematic mapping study on mining software repositories, in: Proceedings of the 31st Annual ACM Symposium on Applied Computing, 2016, pp. 1472–1479.

[30] J. Musengamana, L. Peng, S. Mojtaba, Y. Chen, L. Zengyang, Mining architectural information: A systematic mapping study, Empir. Softw. Eng. (ISSN: 15737616) 29 (4) (2024) 1–59, http://dx.doi.org/10.1007/S10664-024-10480-6/TABLES/8, URL https://link.springer.com/article/10.1007/s10664-024-10480-6.

[31] T. Bi, P. Liang, A. Tang, C. Yang, A systematic mapping study on text analysis techniques in software architecture, J. Syst. Softw. 144 (2018) 533–558.

[32] M. Javed, U. Zdun, A systematic literature review of traceability approaches between software architecture and source code, in: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, 2014, pp. 1–10.

[33] B. Manoj, K. Shumaiev, U. Hohenstein, A. Biesdorf, F. Matthes, The evolution of architectural decision making as a Key Focus Area of software architecture research: A semi-systematic literature study, in: 2020 IEEE International Conference on Software Architecture, ICSA, 2020, pp. 69–80, http://dx.doi.org/10.1109/ICSA47634.2020.00015.

[34] Z. Sinkala, M. Blom, S. Herold, A mapping study of software architecture recovery for software product lines, in: Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings, 2018, pp. 1–7.

[35] N. Qureshi, M. Usman, N. Ikram, Evidence in software architecture, a systematic literature review, in: Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering, 2013, pp. 97–106.

[36] C. Yang, P. Liang, P. Avgeriou, A systematic mapping study on the combination of software architecture and agile development, J. Syst. Softw. (ISSN: 0164-1212) 111 (2016) 157–184, http://dx.doi.org/10.1016/j.jss.2015.09.028.

[37] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, I. Meedeniya, Software architecture optimization methods: A systematic literature review, IEEE Trans. Softw. Eng. 39 (5) (2013) 658–683, http://dx.doi.org/10.1109/TSE.2012.64.

[38] A. Arshad, M. Usman, Security at software architecture level: a systematic mapping study, in: 15th Annual Conference on Evaluation & Assessment in Software Engineering, EASE 2011, IET, 2011, pp. 164–168.

[39] A. Tang, M. Razavian, B. Paech, T. Hesse, Human aspects in software architecture decision making, in: 2017 IEEE International Conference on Software Architecture, ICSA.

[40] H. Hemmati, S. Nadi, O. Baysal, O. Kononenko, W. Wang, R. Holmes, M. Godfrey, The msr cookbook: Mining a decade of research, in: 2013 10th Working Conference on Mining Software Repositories, MSR, IEEE, 2013, pp. 343–352.

[41] A. Abdelrazek, Y. Eid, E. Gawish, W. Medhat, A. Hassan, Topic modeling algorithms and applications: A survey, Inf. Syst. 112 (2023) 102131.

[42] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, J.M. Zhang, Large language models for software engineering: Survey and open problems, in: 2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering, ICSE-FoSE, IEEE, 2023, pp. 31–53.

[43] J.T. Liang, C. Yang, B.A. Myers, A large-scale survey on the usability of ai programming assistants: Successes and challenges, in: Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, 2024, pp. 1–13.

[44] A. Ahmad, M. Waseem, P. Liang, M. Fahmideh, M.S. Aktar, T. Mikkonen, Towards human-bot collaborative software architecting with chatgpt, in: Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering, 2023, pp. 279–285.

[45] R. Dhar, K. Vaidhyanathan, V. Varma, Leveraging generative AI for architecture knowledge management, in: 2024 IEEE 21st International Conference on Software Architecture Companion, ICSA-C, IEEE, 2024, pp. 163–166.

Primary studies

[P1] A.M. Chaniotaki, T. Sharma, Architecture smells and pareto principle: A preliminary empirical exploration, in: MSR, 2021.

[P2] M. Soliman, M. Wiese, Y. Li, M. Riebisch, P. Avgeriou, Exploring Web Search Engines to Find Architectural Knowledge, in: ICSA, 2021.

[P3] I. Malavolta, K. Chinnappan, S. Swanborn, G.A. Lewis, P. Lago, Mining the ROS ecosystem for green architectural tactics in robotics and an empirical evaluation, in: MSR, 2021.

[P4] A. Shahbazian, Y. Kyu Lee, D. Le, Y. Brun, N. Medvidovic, Recovering Architectural Design Decisions, in: ICSA, 2018.

[P5] M. Bhat, K. Shumaiev, A. Biesdorf, U. Hohenstein, F. Matthes, Automatic extraction of design decisions from issue management systems: A machine learning based approach, in: ECSA, 2017.

[P6] D.M. Le, D. Link, A. Shahbazian, N. Medvidovic, An Empirical Study of Architectural Decay in Open-Source Software, in: ICSA, 2018.

[P7] E. Kouroshfar, M. Mirakhorli, H. Bagheri, L. Xiao, S. Malek, Y. Cai, A study on the role of software architecture in the evolution and quality of software, in: MSR, 2015.

[P8] D.M. Le, C. Carrillo, R. Capilla, N. Medvidovic, Relating architectural decay and sustainability of software systems, in: WICSA, 2016.

[P9] Q. Feng, R. Kazman, Y. Cai, R. Mo, L. Xiao, Towards an architecture-centric approach to security analysis, in: WICSA, 2016.

[P10] M. Galster, F. Gilson, F. Georis, What Quality Attributes Can We Find in Product Backlogs? A Machine Learning Perspective, in: ECSA, 2019.

[P11] Y. Zhao, L. Xiao, X. Wang, Z. Chen, B. Chen, Y. Liu, Butterfly space: An architectural approach for investigating performance issues, in: ICSA, 2020.

[P12] J. Garcia, I. Krka, N. Medvidovic, C. Douglas, A framework for obtaining the ground-truth in architectural recovery, in: WICSA/ ECSA, 2012.

[P13] A.W. Kamal, P. Avgeriou, Mining Relationships between the Participants of Architectural Patterns., in: ECSA, 2010.

[P14] L. Xiao, Y. Cai, R. Kazman, R. Mo, Q. Feng, Identifying and quantifying architectural debt, in: ICSE, 2016.

[P15] F. Haupt, F. Leymann, A. Scherer, K. Vukojevic-Haupt, A Framework for the Structural Analysis of REST APIs, in: ICSA, 2017.

[P16] M. Mirakhorli, J. Cleland-Huang, Modifications, tweaks, and bug fixes in architectural tactics, in: MSR, 2015.

[P17] D. Sas, P. Avgeriou, U. Uyumaz, On the evolution and impact of architectural smells - an industrial case study, in: ESEM, 2022.

[P18] R. Weinreich, G. Buchgeher, Automatic reference architecture conformance checking for SOA - Based software systems, in: WICSA, 2014.

[P19] A. Jansen, J. Bosch, P. Avgeriou, Documenting after the fact: Recovering architectural design decisions, in: JSS, 2008.

[P20] A. Eposhi, W. Oizumi, A. Garcia, L. Sousa, R. Oliveira, A. Oliveira, Removal of design problems through refactorings: are we looking at the right symptoms, in: ICPC, 2019.

[P21] E. Ntentos, U. Zdun, K. Plakidas, S. Geiger, Semi-automatic Feedback for Improving Architecture Conformance to Microservice Patterns and Practices, in: ICSA, 2021.

[P22] M. Soliman, A. Rekaby, M. Galster, O. Zimmermann, M. Riebisch, Improving the Search for Architecture Knowledge in Online Developer Communities, in: ICSA, 2018.

[P23] A. Bandeira, C.A. Medeiros, M. Paixao, P.H. Maia, We need to talk about microservices: An analysis from the discussions on stackoverflow, in: MSR, 2019.

[P24] N. Brown, R.L. Nord, I. Ozkaya, M. Pais, Analysis and management of architectural dependencies in iterative release planning, in: WICSA, 2011.

[P25] X. Wang, L. Xiao, K. Huang, B. Chen, Y. Zhao, Y. Liu, Designdiff: Continuously modeling software design difference from code revisions, in: ICSA, 2020.

[P26] A. Sejfia, N. Medvidovic, Strategies for pattern-based detection of architecturally-relevant software vulnerabilities, in: ICSA, 2020.
[P27] R. Mo, Y. Cai, R. Kazman, L. Xiao, Hotspot Patterns: The Formal Definition and Automatic Detection of Architecture Smells, in: WICSA, 2015.
[P28] I. Gorton, R. Xu, Y. Yang, H. Liu, G. Zheng, Experiments in Curation: Towards Machine-Assisted Construction of Software Architecture Knowledge Bases, in: ICSA, 2017.
[P29] M. Soliman, M. Galster, M. Riebisch, Developing an Ontology for Architecture Knowledge from Developer Communities, in: ICSA, 2017.
[P30] J.C. Santos, M. Mirakhorli, I. Mujhid, W. Zogaan, BUDGET: A tool for supporting software architecture traceability research, in: WICSA, 2016.
[P31] J. Garcia, M. Mirakhorli, L. Xiao, Y. Zhao, I. Mujhid, K. Pham, A. Okutan, S. Malek, R. Kazman, Y. Cai, N. Medvidovic, Constructing a Shared Infrastructure for Software Architecture Analysis and Maintenance, in: ICSA, 2021.
[P32] J. van der Ven, J. Bosch, Making the Right Decision: Supporting Architects with Design Decision Data, in: ECSA, 2013.
[P33] M. Bhat, K. Shumaiev, K. Koch, U. Hohenstein, A. Biesdorf, F. Matthes, An Expert Recommendation System for Design Decision Making: Who Should be Involved in Making a Design Decision? in: ICSA, 2018.
[P34] M. Soliman, M. Galster, A. Salama, M. Riebisch, Architectural Knowledge for Technology Decisions in Developer Communities: An Exploratory Study with StackOverflow, in: WICSA, 2016.
[P35] F. Tian, P. Liang, M.A. Babar, How developers discuss architecture smells? An exploratory study on stack overflow, in: ICSA, 2019.
[P36] B. Nassar, R. Scandariato, Traceability Metrics as Early Predictors of Software Defects? in: ICSA, 2017.
[P37] W. Heider, M. Vierhauser, D. Lettner, P. Grunbacher, A case study on the evolution of a component-based product line, in: WICSA/ ECSA, 2012.
[P38] N.B. Harrison, E. Gubler, D. Skinner, Software architecture pattern morphology in open-source systems, in: WICSA, 2016.
[P39] R. Verdecchia, I. Malavolta, P. Lago, Guidelines for architecting android apps: A mixed-method empirical study, in: ICSA, 2019.
[P40] A. Uchoa, C. Barbosa, D. Coutinho, W. Oizumi, W.K. Assuncao, S.R. Vergilio, J.A. Pereira, A. Oliveira, A. Garcia, Predicting design impactful changes in modern code review: A large-scale empirical study, in: MSR, 2021.
[P41] R. Svensson, A. Tatrous, F. Palma, Defining Design Patterns for IoT APIs, in: ECSA, 2020.
[P42] G. Buchgeher, R. Weinreich, Automatic tracing of decisions to architecture and implementation, in: WICSA, 2011.
[P43] W. Hu, D. Han, A. Hindle, K. Wong, The build dependency perspective of Android's concrete architecture, in: MSR, 2012.
[P44] J.C. Santos, A. Peruma, M. Mirakhorli, M. Galstery, J.V. Vidal, A. Sejfia, Understanding Software Vulnerabilities Related to Architectural Security Tactics: An Empirical Investigation of Chromium, PHP and Thunderbird, in: ICSA, 2017.
[P45] M. Langhammer, A. Shahbazian, N. Medvidovic, R.H. Reussner, Automated extraction of rich software models from limited system information, in: WICSA, 2016.
[P46] R. Roeller, P. Lago, H. van Vliet, Recovering architectural assumptions, in: JSS, 2006.
[P47] A. Raghuraman, T. Ho-Quang, M.R. Chaudron, A. Serebrenik, B. Vasilescu, Does UML modeling associate with lower defect proneness?: A preliminary empirical investigation, in: MSR, 2019.
[P48] D.M. Le, P. Behnamghader, J. Garcia, D. Link, A. Shahbazian, N. Medvidovic, An empirical study of architectural change in open-source software systems, in: MSR, 2015.
[P49] A.S.D.N.N. Medvidovic, Toward predicting architectural significance of implementation issues, in: MSR, 2018.
[P50] D.M. Le, S. Karthik, M.S. Laser, N. Medvidovic, Architectural Decay as Predictor of Issue- And Change-Proneness, in: ICSA, 2021.
[P51] M. Wermelinger, Y. Yu, An architectural evolution dataset, in: MSR, 2015.
[P52] Y. Zhang, B. Liu, L. Dai, K. Chen, X. Cao, Automated microservice identification in legacy systems with functional and non-functional metrics, in: ICSA, 2020.
[P53] M. Shabin, A.R. Nasab, M.A. Babar, A qualitative study of architectural design issues in DevOps, in: JSEP, 2021.
[P54] V. Guana, F. Rocha, A. Hindle, E. Stroulia, Do the stars align? Multidimensional analysis of Android's layered architecture, in: MSR, 2012.
[P55] N.B. Harrison, P. Avgeriou, Analysis of architecture pattern usage in legacy system architecture documentation, in: WICSA, 2008.
[P56] C. Kapto, G. El Boussaidi, S. Kpodjedo, C. Tibermacine, Inferring Architectural Evolution from Source Code Analysis, in: ECSA, 2016.
[P57] L.B.L. De Souza, M. De Almeida Maia, Do software categories impact coupling metrics? in: MSR, 2013.
[P58] M. Ahmad, M.O. Cinneide, Impact of stack overflow code snippets on software cohesion: A preliminary study, in: MSR, 2019.
[P59] S. Raemaekers, G.F. Nane, A. Van Deursen, J. Visser, Testing principles, current practices, and effects of change localization, in: MSR, 2013.
[P60] M. Soliman, K. Gericke, P. Avgeriou, Where and what do software architects blog?: An exploratory study on architectural knowledge in blogs, and their relevance to design steps, in: ICSA, 2023.
[P61] M. Soliman, M. Galster, P. Avgeriou, An exploratory study on architectural knowledge in issue tracking systems, in: ECSA, 2021.
[P62] R. Weinreich, G. Buchgeher, Automatic reference architecture conformance checking for SOA-based software systems, in: WICSA, 2014.
[P63] R. Mo, Y. Cai, R. Kazman, L. Xiao, Q. Feng, Architecture anti-patterns: Automatically detectable violations of design principles, in: TSE, 2019.
[P64] T. Sharma, P. Singh, D. Spinellis, An empirical investigation on the relationship between design and architecture smells, in: ESEM, 2020.
[P65] F.A. Fontana, V. Lenarduzzi, R. Roveda, D. Taibi, Are architectural smells independent from code smells an empirical study, in: JSS, 2019.
[P66] I. Malavolta, G. Lewis, B. Schmerl, P. Lago, D. Garlan, How do you architect your robots state of the practice and guidelines for ROS-based systems, in: ICSE, 2020.
[P67] P. Behnamghader, D.M. Le, J. Garcia, D. Link, A. Shahbazian, N. Medvidovic, A large-scale study of architectural evolution in open-source software systems, in: ESEM, 2016.
[P68] R. Kazman, Y. Cai, R. Mo, Q. Feng, L. Xiao, S. Haziyev, V. Fedak, A. Shapochka, A case study in locating the architectural roots of technical debt, in: ICSE, 2015.
[P69] R. Mo, Y. Cai, R. Kazman, L. Xiao, Q. Feng, Decoupling level: A new metric for architectural maintenance complexity, in: ICSE, 2016.
[P70] L. Xiao, Y. Cai, R. Kazman, Design rule spaces: A new form of architecture insight, in: ICSE, 2014.
[P71] A.K. Mondal, C.K. Roy, K.A. Schneider, B. Roy, S.S. Nath, Semantic slicing of architectural change commits: Towards semantic design review, in: ESEM, 2021.
[P72] I. Malavolta, G.A. Lewis, B. Schmerl, P. Lago, D. Garlan, Mining guidelines for architecting robots software, in: JSS, 2021.
[P73] M. Paixao, J. Krinke, D. Han, C. Ragkhitwetsagul, The impact of code review on architectural changes, in: TSE, 2021.
[P74] J. Liu, W. Jin, Q. Feng, X. Zhang, Y. Dai, One step further: Investigating problematic files of architecture anti-patterns, in: ISSRE, 2021.
[P75] Y. Cai, L. Xiao, R. Kazman, R. Mo, Q. Feng, Design rule spaces: A new model for representing and analyzing software architecture, in: TSE, 2019.
[P76] P. Gnoyke, S. Schulze, J. Kruger, An evolutionary analysis of software-architecture smells, in: ICSME, 2021.
[P77] M. Goldstein, I. Segall, Automatic and continuous software architecture validation, in: ICSE, 2015.
[P78] E. Bouwers, A. v. Deursen, J. Visser, Quantifying the encapsulation of implemented software architectures, in: ICSME, 2014.
[P79] R. Mo, W. Snipes, Y. Cai, S. Ramaswamy, R. Kazman, M. Naedele, Experiences applying automated architecture analysis tool suites, in: ASE, 2018.
[P80] M. Waseem, P. Liang, M. Shahin, A. Ahmad, A.R. Nassab, On the nature of issues in five open source microservices systems: An empirical study, in: EASE, 2021.
[P81] T.D. Oyetoyan, D.S. Cruzes, R. Conradi, A study of cyclic dependencies on defect profile of software components, in: JSS, 2013.
[P82] D. Sas, P. Avgeriou, F.A. Fontana, Investigating instability architectural smells evolution: an exploratory case study, in: ICSME, 2019.
[P83] D. Cui, T. Liu, Y. Cai, Q. Zheng, Q. Feng, W. Jin, J. Guo, Y. Qu, Investigating the impact of multiple dependency structures on software defects, in: ICSE, 2019.
[P84] W. Jin, Y. Cai, R. Kazman, G. Zhang, Q. Zheng, T. Liu, Exploring the architectural impact of possible dependencies in python software, in: ASE, 2020.
[P85] L. Sousa, W. Oizumi, A. Garcia, A. Oliveira, D. Cedrim, C. Lucena, When are smells indicators of architectural refactoring opportunities a study of 50 software projects, in: ICPC, 2020.
[P86] S. Vidal, W. Oizumi, A. Garcia, A.D. Pace, C. Marcos, Ranking architecturally critical agglomerations of code smells, in: SCICO, 2019.
[P87] T. Bi, P. Liang, A. Tang, X. Xia, Mining architecture tactics and quality attributes knowledge in stack overflow, in: JSS, 2021.
[P88] N. Dintzer, A. v. Deursen, M. Pinzger, FEVER: An approach to analyze feature-oriented changes and artefact co-evolution in highly configurable systems, in: ESEM, 2017.
[P89] M. Wermelinger, Y. Yu, A. Lozano, A. Capiluppi, Assessing architectural evolution: a case study, in: ESEM, 2011.
[P90] W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, Q. Zheng, Service candidate identification from monolithic systems based on execution traces, in: TSE, 2021.
[P91] I. Macia, J. Garcia, D. Popescu, A. Garcia, N. Medvidovic, A. v. Staa, Are automatically-detected code anomalies relevant to architectural modularity an exploratory analysis of evolving systems, in: AOSD, 2012.
[P92] R. Arcoverde, E. Guimaraes, I. Macia, A. Garcia, Y. Cai, Prioritization of code anomalies based on architecture sensitiveness, in: SBES, 2013.
[P93] T. Sharma, P. Mishra, R. Tiwari, Designite - A software design quality assessment tool, in: BRIDGE, 2016.
[P94] J. Brunet, R.A. Bittencourt, D. Serey, J. Figueiredo, On the evolutionary nature of architectural violations, in: WCRE, 2012.

[P95] S. Hassaine, Y.-G. Gueheneuc, S. Hamel, G. Antoniol, ADvISE: Architectural decay in software evolution, in: CSMR, 2012.
[P96] I. Macia, R. Arcoverde, A. Garcia, C. Chavez, A. v. Staa, On the relevance of code anomalies for identifying architecture degradation symptoms, in: CSMR, 2012.
[P97] R.S. Sangwan, P. Vercellone-Smith, C.J. Neill, Use a of multidimensional approach to study the evolution of software complexity, 2010.
[P98] Z. Li, P. Liang, P. Avgeriou, N. Guelfi, A. Ampatzoglou, An empirical investigation of modularity metrics for indicating architectural technical debt, in: QoSA, 2014.
[P99] M. DAmbros, M. Lanza, R. Robbes, On the relationship between change coupling and software defects, in: WCRE, 2009.
[P100] D. Sas, P. Avgeriou, I. Pigazzini, F.A. Fontana, On the relation between architecture smells and source code changes, in: JSEP, 2022.
[P101] F.A. Fontana, P. Avgeriou, I. Pigazzini, R. Roveda, A study on architectural smells prediction, in: SEAA, 2019.
[P102] M. Machado, R. Choren, Improving the detection of evolutionary coupling: An approach considering sliding verification, in: SAC, 2018.
[P103] A. Zakurdaeva, M. Weiss, S. Muegge, Detecting architectural integrity violation patterns using machine learning, in: SAC, 2020.
[P104] M. Ali, H. Mushtaq, M.B. Rasheed, A. Baqir, T. Alquthami, Mining software architecture knowledge: Classifying stack overflow posts using machine learning, in: CPE, 2021.
[P105] L. Xiao, M.J. Pennock, J.L.F.P. Cardoso, X. Wang, A case study on modularity violations in cyber-physical systems, in: SYS, 2020.
[P106] T. Wang, B. Li, Analyzing software architecture evolvability based on multiple architectural attributes measurements, in: QRS, 2019.
[P107] G. Mazlami, J. Cito, P. Leitner, Extraction of microservices from monolithic software architectures, in: ICWS, 2017.
[P108] D. Cui, J. Guo, T. Liu, Q. Zheng, An empirical study of architectural changes in code commits, Internetware (2020).
[P109] A. Sejfia, A pilot study on architecture and vulnerabilities: Lessons learned, in: ECASE, 2019.
[P110] J.A. Diaz-Pace, A. Tommasel, I. Pigazzini, F.A. Fontana, Sen4Smells: A tool for ranking sensitive smells for an architecture debt index, in: ARGENCON, 2020.
[P111] D. Sas, P. Avgeriou, R. Kruizinga, R. Scheedler, Exploring the relation between co-changes and architectural smells, in: SNCS, 2021.
[P112] K.Z. Sultana, Z. Codabux, B. Williams, Examining the relationship of code and architectural smells with software vulnerabilities, in: APSEC, 2020.
[P113] G. Rodriguez, J.A. Diaz-Pace, L. Berdun, S. Misra, Deriving architectural responsibilities from textual requirements, in: ICIIA, 2021.
[P114] M. Bhat, K. Shumaiev, A. Biesdorf, U. Hohenstein, M. Hassel, F. Matthes, An ontology-based approach for software architecture recommendations, in: AMCIS, 2017.
[P115] K. Gomes, L. Teixeira, T. Alves, M. Ribeiro, R. Gheyi, Characterizing safe and partially safe evolution scenarios in product lines: An empirical study, in: VAMOS, 2019.
[P116] R. Minelli, M. Lanza, Software analytics for mobile applications - insights lessons learned, in: CSMR, 2013.
[P117] L. Wang, X. Kong, J. Wang, B. Li, An incremental software architecture recovery technique driven by code changes, in: FITEE, 2022.
[P118] C. Barbosa, A. Uchoa, D. Coutinho, F. Falcao, H. Brito, G. Amaral, V. Soares, A. Garcia, B. Fonseca, M. Ribeiro, L. Sousa, Revealing the social aspects of design decay: A retrospective study of pull requests, in: SBES, 2020.
[P119] A. McNair, D.M. German, J. Weber-Jahnke, Visualizing software architecture evolution using change-sets, in: WCRE, 2007.
[P120] J. Ryoo, R. Kazman, P. Anand, Architectural analysis for security, in: MSP, 2015.
[P121] R. Schwanke, L. Xiao, Y. Cai, Measuring architecture quality by structure plus history analysis, in: ICSE, 2013.
[P122] M. Chaabane, I.B. Rodriguez, K. Drira, M. Jmaiel, Mining approach for software architectures description discovery, in: AICCSA, 2017.
[P123] D. Spinellis, P. Avgeriou, Evolution of the unix system architecture: An exploratory case study, in: TSE, 2021.
[P124] R. Marinescu, Detection strategies: Metrics-based rules for detecting design flaws, in: ICSM, 2004.
[P125] C. Amrit, J. van Hillegersberg, Coordination implications of software coupling in open source projects, in: OSS, 2010.
[P126] C. Izurieta, J.M. Bieman, How software designs decay: A pilot study of pattern evolution, in: ESEM, 2007.
[P127] S.A. Tonu, A. Ashkan, L. Tahvildari, Evaluating architectural stability using a metric-based approach, in: CSMR, 2006.
[P128] T. Wetzlmaier, C. Klammer, R. Ramler, Extracting dependencies from software changes: An industry experience report, in: IWSM, 2014.
[P129] K. Chinnappan, I. Malavolta, G.A. Lewis, M. Albonico, Architectural tactics for energy-aware robotics software: A preliminary study, in: ECSA, 2021.
[P130] J. Keim, S. Schulz, D. Fuch, C. Kocher, J. Speit, Trace link recovery for software architecture documentation, in: ECSA, 2021.
[P131] P.G.U. Zdun, Identifying domain-based cyclic dependencies in microservice APIs using source code detectors, in: ECSA, 2021.
[P132] S.S.C.P. Fabio Di Lauro, To deprecate or to simply drop operations an empirical study on the evolution of a large OpenAPI collection, in: ECSA, 2022.
[P133] E. Ntentos, U. Zdun, J.S.A. Brogi, Assessing architecture conformance to coupling-related infrastructure-as-code best practices: Metrics and case studies, in: ECSA, 2022.
[P134] R. Li, M. Soliman, P. Liang, P. Avgeriou, Symptoms of architecture erosion in code reviews: A study of two OpenStack projects, in: ICSA, 2022.
[P135] M. Muszynski, S. Lugtigheid, F. Castor, S. Brinkkemper, A study on the software architecture documentation practices and maturity in open-source software development, in: ICSA, 2022.
[P136] S.J. Warnett, U. Zdun, Architectural design decisions for machine learning deployment, in: ICSA, 2022.
[P137] W. Ding, P. Liang, A. Tang, H. van Vliet, How do open source communities document software architecture: An exploratory survey, in: ICECCS, 2014.
[P138] R. Kazman, D. Goldenson, I. Monarch, W. Nichols, Evaluating the effects of architectural documentation: A case study of a large scale open source project, in: TSE, 2016.
[P139] K. de Graaf, P. Liang, A. Tang, H. van Vliet, How organisation of architecture documentation affects architectural knowledge retrieval, in: SCICO, 2016.
[P140] T. Bi, W. Ding, P. Liang, A. Tang, Architecture information communication in two OSS projects: The why, who, when, and what, in: JSS, 2021.
[P141] J. Garcia, E. Kouroshfar, N. Ghorbani, S. Malek, Forecasting architectural decay from evolutionary history, in: TSE, 2022.
[P142] W. Oizumi, L. Sousa, A. Oliveira, L. Carvalho, A. Garcia, T. Colanzi, On the density and diversity of degradation symptoms in refactored classes: A multi-case study, in: ISSRE, 2019.
[P143] A. Uchoa, C. Barbosa, W. Oizumi, P. Blenilio, R. Lima, A. Garcia, How does modern code review impact software design degradation an in-depth empirical study, in: ICSME, 2020.
[P144] A. Mahadi, N.A.E.K. Tongay, Conclusion stability for natural language based mining of design discussions, in: ESEM, 2022.
[P145] L. Wijerathna, A. Aleti, T.B.A. Tang, Mining and relating design contexts and design patterns from stack overflow, in: ESEM, 2022.
[P146] U.A. Mannan, I. Ahmed, C. Jensen, A. Sarma, On the relationship between design discussions and design quality: a case study of apache projects, in: FSE, 2020.
[P147] R. Morales, S. McIntosh, F. Khomh, Do code review practices impact design quality a case study of the Qt, VTK, and ITK projects, in: SANER, 2015.
[P148] J. Samuel, J. Jaskolka, G.O.M. Yee, Leveraging external data sources to enhance secure system design, in: RDAAPS, 2021.
[P149] I.C. Irsan, T.T. Zhang, K. Ferdian, K. Kisub, Picaso: Enhancing API recommendations with relevant stack overflow posts, in: MSR, 2023.
[P150] J. Keim, S. Corallo, D. Fuchß, A. Koziolek, Detecting inconsistencies in software architecture documentation using traceability link recovery, in: ICSA, 2023.
[P151] H.C. Vazquez, J.A. Diaz-Pace, S.A. Vidal, C. Marcos, A recommender system for recovering relevant JavaScript packages from web repositories, in: ICSA, 2023.