UNIVERSITY OF MACEDONIA
POSTGRADUATE STUDIES PROGRAMME
DEPARTMENT OF APPLIED INFORMATICS

GUIDELINES ON THE SELECTION OF OPEN-SOURCE SOFTWARE COLLECTIONS IN AN EMPIRICAL STUDY

Diploma Thesis

of

Kazantzidis Dimitrios

Thessaloniki, October 2023

GUIDELINES ON THE SELECTION OF OPEN-SOURCE SOFTWARE COLLECTIONS IN AN EMPIRICAL STUDY

Kazantzidis Dimitrios

Bachelor's Degree in Mathematics, Ioannina, 2016

Diploma Thesis

submitted in partial fulfilment of the requirements of the

POSTGRADUATE DEGREE IN APPLIED INFORMATICS

Supervising Professor
Ονοματεπώνυμο Καθηγητή/τριας

Approved by the three-member committee on ηη/μμ/εεεε

Ονοματεπώνυμο 1	Ονοματεπώνυμο 2	Ονοματεπώνυμο 3
.....	.....	.....

Kazantzidis Dimitrios

.....

Summary

The purpose of this thesis is to conduct a systematic mapping study on empirical researches in the field of software technology and to record the open-source software that have been used by researchers to verify the methodology they proposed in their papers. To achieve this, a systematic literature review has been carried out on scientific articles and papers published during the last years in specific conferences and journals aiming at the collection of open-source software that have been used in empirical studies. The aim of this research is to create a set of data which may well be reused in future research and to categorize open-source software so that they are easily available for future use by researchers. This thesis is divided into the following main sections. The first section is the introduction of this thesis. The second section refers to relevant articles and papers on the topic discussed in this thesis. The third section deals with the process and methodology used to complete the thesis. The fourth section presents the results of the systematic mapping study in the software technology industry. The fifth section presents a tool that we created so future researchers can access our data and finally the sixth section is the summary of the paper, the conclusions that have been drawn and suggestions for further research on this topic are presented.

Keywords: empirical study, software technology, open-source software (OSS), systematic mapping study (SMS), mining software repository (MSR), free and open-source software (FOSS)

Abstract

Background: In the field of software engineering, there has been an increasing trend of using empirical studies, as a method of verifying the particular methodology proposed by the researcher. However, to verify the proposed methodology it is quite common to use various collections of open-source software to verify the proposed methodology. However, no attempt has been made to record open-source software (OSS) used and their categorization, which can be considered as important and quite useful for the researchers and their future empirical studies.

Objective: The aim of this research is to record the OSS that have been used in papers in recent years in specific journals, identify research goals that were set, find the most used projects and the criteria that led in that choice and provide the link to the data (if available). Finally, we created a webpage with all the information that we collected so it can be used in future research.

Methodology: In this thesis we will conduct a systematic mapping study according to international standards. Initially we will retrieve articles of the last few years from different sources (e.g., JSS, ESE, etc.). Then we will select the papers which are empirical studies and use open-source software. In total 1492 papers were studied and according to the selection criteria we extracted the data we want to analyze from a total of 394 papers. Finally, we created a webpage that future researchers can consult and find the appropriate open-source software (from the articles we collected previously) according to the goals they have defined in their work.

Results and conclusion: Our comprehensive analysis of over 1400 papers revealed that the most common project selection criteria encompassed size, language, and popularity of the projects. Frequently used subjects in these studies included prominent projects like Apache and Eclipse highlighting the importance of mature development processes and large contributor communities. We found that the main goals of Mining Software Repositories (MSR) studies were to understand software development practices, identify defect patterns and their fixes, and predict software quality and maintenance effort. We used our findings to create an informative webpage that synthesizes these results and provides a data-driven guide to selecting open-source software for empirical research. This work underscores the need for a nuanced approach

to project selection and opens avenues for further research into potential biases and the evolving nature of selection criteria.

Prologue - Acknowledgements

At this point I would like to thank professors Apostolos Ampatzoglou and Elvira-Maria Arvanitou for their valuable contribution to the completion of this thesis and the guidance they provided throughout its duration and until its completion. I would also like to thank Nikolaos Nikolaidis for his valuable help and time spent in setting up the web page and backend on the servers of the University of Macedonia.

Table of Content

Index of Images

Index of Tables

1 Introduction

1.1 Background Information

Software development is a complex and demanding process involving many variables, including technological developments, evolving user requirements and organizational constraints. As a result, both researchers and practitioners have recognized the need for empirical studies to gain insights into software engineering practices and improve software development outcomes.

Empirical research provides a systematic approach to investigate phenomena in software engineering, allowing researchers to understand, evaluate, and improve software development processes.

According to Wohlin et al. (2003) empirical studies fall under four categories: *controlled experiments, case studies, surveys, and post hoc analysis*.

Controlled experiments are carefully designed investigations where researchers supervise the study progression. Often conducted in laboratories, these experiments aim to compare techniques, methods, or processes. They typically involve planning the study design, conducting the operation with participant engagement, preparing measurement tools, and executing data analysis and result interpretation. However, they may lack a broad experimental range.

Case studies delve into specific phenomena or sets. They collect detailed information via multiple techniques and are particularly useful for monitoring software engineering tasks. Case studies involve defining objectives, preparing for data collection, gathering data, analyzing the information, and drawing conclusions (Runeson et al., 2009). While they can effectively evaluate phenomena, their results may not be universally applicable.

Surveys provide a snapshot of a present situation and are employed when a tool or technique is complete or nearing completion (Wohlin et al., 2003). They collect data through questionnaires or interviews from a representative population to understand various attributes or characteristics. Surveys typically involve identifying objectives and participants, selecting the survey type, designing questions, conducting a pilot study, distributing the survey, and analyzing results.

Post-mortem analysis reviews past studies, focusing on specific situations. This method, resembling a case study but differing in the chronological investigation, aims to

learn from past experiences. Two types exist: a general review collecting all available information and a specific review targeting particular activities.

These empirical study methods can be grouped into qualitative and quantitative research paradigms. Qualitative research examines phenomena within their natural environment, interpreting events based on human understanding. Conversely, quantitative research focuses on quantifying relationships or comparisons, aiming to identify cause-effect connections. Controlled experiments and case studies are common methods for this research, providing an opportunity for comparison and statistical analysis.

Open-source software (OSS) is deeply entrenched in the history of computing, with roots dating back to the computing community's early shared ethos in the 1960s and 70s (Levy, 1984). The free sharing of software and its source code was a common practice. However, the rising commercialization of software in the late 70s and early 80s led to companies choosing to protect their intellectual property and profits by keeping source code private (Wayner, 2000).

Distressed by this trend, Richard Stallman catalyzed the OSS movement in 1983. Through the establishment of the Free Software Foundation and the initiation of the GNU project, Stallman strived to create an entirely open and free operating system. He also proposed the notion of "copyleft," a distinct variation of copyright designed to ensure that modified program versions retain their open-source nature (Stallman, 1985).

The actual phrase "open source" came into being only in 1998, coined by Christine Peterson considering Netscape's decision to make their Navigator web browser's source code freely available. This moment served as a critical turning point for OSS, marking the first instance of open-source development receiving backing from a large-scale corporation (Raymond, 1999).

The end of the 20th century and the dawn of the 21st saw OSS momentum accelerating. Several substantial projects such as the Apache web server, the Linux kernel, and the MySQL database came to life during this period. This era also witnessed technology giants like IBM starting to place substantial investments in OSS development (Weber, 2004).

The advent of platforms such as GitHub in the mid to late 2000s brought about a significant shift in the OSS landscape, facilitating easier contribution from developers

worldwide. This platform evolution resulted in a more community-driven and diversified development within OSS (Dabbish et al., 2012).

In recent years, the use of open-source software (OSS) has received attention in empirical research studies in various disciplines. Open-source software refers to computer software that is distributed with its source code openly available for users to view, modify and distribute. The adoption of OSS in empirical research offers several advantages, such as cost-effectiveness, transparency, and the ability to leverage collective intelligence (Steinmacher et al., 2016). However, the wide variety of open-source software options available presents a unique challenge for researchers when selecting the most appropriate tools for their studies (Steinmacher et al., 2016).

However, the need to optimize results and find suitable tools has given rise to a new field, Mining Software Repositories (MSR).

Mining Software Repositories (MSR) is an emerging field of empirical software engineering that focuses on extracting valuable information from software repositories. Kalliamvakou et al. (2016) present the mining capabilities of GitHub, a popular platform for hosting open-source projects, with the goal of uncovering valuable information. Software repositories contain a wealth of information such as source code, version control history, bug reports, mailing list discussions, and more. By analyzing this data, researchers can gain a deeper understanding of software development processes, identify patterns, and improve software engineering practices.

1.2 Research Problem

The importance of choosing open-source software in empirical research cannot be ignored. The selection of appropriate software plays a key role in ensuring the validity, reproducibility and reliability of research findings (Steinmacher et al., 2016). Open-source software offers researchers the ability to examine the underlying code, ensuring transparency and facilitating the reproducibility of results, which are fundamental principles in empirical research.

And although empirical research methods have gained prominence in software engineering, the lack of standardization and consistency in the conduct and reporting of studies has created challenges. The absence of clear guidelines can cause inconsistencies in study designs, data collection, analysis, and reporting, which can undermine the

reliability and comparability of research findings. In addition, the complexity of some empirical studies, particularly those involving multiple related websites and open-source projects, presents unique challenges that require specific guidelines to ensure rigor and reliability.

Therefore, the development and adoption of well-defined guidelines for empirical research in software engineering is essential to ensure robustness, reproducibility, and reliability. Moreover, the existence of guidelines will facilitate the comparability of research findings and promote knowledge accumulation.

1.3 Research Goals

Despite the advantages of using open-source software, choosing the most suitable software for empirical research can be a difficult task. The sheer number of options available, combined with varying quality, compatibility, and support, presents a formidable challenge for researchers. In addition, the dynamic nature of open-source software development introduces further complexities as new tools emerge and existing ones evolve rapidly.

Therefore, the primary objective of this thesis is to propose a set of guidelines for conducting and reporting empirical research in software engineering with respect to the selection of open-source projects. The guidelines aim to provide researchers with a systematic framework for effectively designing and conducting their empirical studies on software engineering.

In the following, we aim to create a repository of all empirical studies that have been studied, present the open-source projects that have been used, the criteria based on which they were selected and finally the goals of each study. Our aim is to provide a catalogue of open-source projects in the form of a web page, which will be a valuable resource for future researchers.

Overall, this thesis seeks to contribute to the field of empirical research by providing researchers with a set of guidelines for selecting open-source software. By offering a structured approach to software selection, researchers can overcome the challenges posed by the plethora of available options and harness the potential of open-source software to enhance the validity and reproducibility of their empirical studies.

However, it is important to recognize that the guidelines presented in this thesis may not cover every possible scenario when it comes to empirical studies in the software engineering field. The proposed guidelines are intended to provide a fundamental framework for conducting empirical studies, but individual researchers may need to adapt and refine the guidelines to fit their specific research contexts.

1.4 Study Structure

The remainder of this thesis is organized as follows:

Chapter 2: In this chapter we discuss related work on mining software repositories and empirical studies in the field of software engineering.

Chapter 3: In this chapter, we present the systematic mapping model we followed for our data mining.

Chapter 4: While in chapter 4, we will present the results of our research.

Chapter 5: In this chapter, we will present the website we created with the aim of creating a directory of open-source projects, hoping to help future researchers.

Chapter 6: The final chapter will summarize the contributions of this thesis, discuss its limitations, and suggest possible directions for future research.

By developing and promoting standardized guidelines, this thesis aims to contribute to the advancement of empirical software engineering and facilitate the generation of high-quality and reliable empirical evidence in the field.

2 Related Work

Empirical methods are central to software engineering. They are not only essential for developing technical solutions, but also for understanding organizational issues, project management, and human behavior. Indeed, the use of empirical methods allows human behavior to be incorporated into the research approach, a critical aspect of a discipline such as software engineering. The use of empirical methods, which is common practice in many other disciplines, helps to understand the complex dynamics within software engineering processes, such as the interaction between team members, decision-making processes, and the impact of management practices on software development outcomes. (Wohlin et al., 2003)

An empirical study conducted interviews with developers from 16 Norwegian software companies that integrate Open-Source Software (OSS) components into their systems. The study found that the selection of OSS components is situational in nature, with project-specific characteristics significantly constraining the outcome of the selection, and that developers use a 'first fit' rather than a 'best fit' approach when selecting OSS components. This may explain the limited adoption of normative selection approaches and generic evaluation schemes. The findings motivate a shift from developing such methods and schemas to understanding the situational nature of software selection. (Oyvind Hauge et al., 2009)

In another study, researchers analyzed up to 21 years of activity in 1314 individual FOSS projects and 1.4 billion lines of code. The study found that there is less activity now than there was a decade ago, especially in large and well-established FOSS organizations. The findings suggest that as technologies and business strategies around FOSS mature, the role of large formal FOSS organizations as intermediaries between developers is diminishing. (Chełkowski T et al., 2021)

In the context of Open-Source Software (OSS), Mining Software Repositories (MSR) has emerged as an important area of research. Software repositories contain a wealth of data, including the history of software changes throughout its evolution. By effectively mining this data, researchers and practitioners can extract valuable information and draw meaningful conclusions about the history or current state of the software. MSR approaches have been used with various goals in mind, such as understanding defects, analyzing developer contribution and behavior, and gaining

insight into software evolution. However, despite the wide application of MSR, there are still gaps in the goals, focus, and types of data sources used in MSR. For example, code comments are often under-utilized to identify code smells, refactoring opportunities, and software quality issues. (Mário André de F. Farias et al., 2016).

The literature review in "A Mining Software Repository Extended Cookbook: Lessons learned from a literature review" provides a comprehensive overview of the MSR field. The review analyzed 276 primary studies from the proceedings of the Working Conference on Mining Software Repositories (MSR). The findings suggest that the MSR field is gradually maturing, as evidenced by the increasing use of software artifacts, the shift towards more empirical studies using data from software repositories, and a greater focus on the context in which these studies are conducted. In addition, the review highlights the need for future research to pay greater attention to the validity of findings and the needs of practitioners. (D'Angelo R. Barros et al., 2021)

Another study, "A survey and taxonomy of approaches for mining software repositories in the context of software evolution", provides a comprehensive overview of the techniques and tools used in MSR. The study categorizes these techniques into three main groups: change-based, human-based, and defect prediction techniques. This categorization provides valuable insights into the range of approaches available for MSR and the different contexts in which they can be applied. (Kagdi et al., 2007)

Both empirical research methods and MSR are critical components of software engineering research. Their combined use enables a deeper understanding of software development practices, developer behavior, and software evolution. Despite this, the literature suggests that there are still areas that require further exploration and study, particularly in relation to the aims, focus, and data sources used in MSR, as well as the context in which empirical studies are conducted. As the field of MSR continues to mature, it is anticipated that future research will aim to fill these gaps and explore new areas of application.

It's also worth noting the ethical implications of mining software repositories. Researchers should consider privacy and confidentiality issues when extracting and using data from these repositories. In addition, the increasing use of machine learning techniques in MSR presents both opportunities and challenges. These techniques can help to automate and improve the accuracy of analysis, but they also raise questions about the interpretability and fairness of the results.

Finally, the interplay between MSR and other areas of software engineering, such as software maintenance and evolution, software quality assurance, and software analytics, deserves attention. Understanding how MSR can contribute to these areas, and vice versa, can promote a more integrated and comprehensive approach to software engineering research and practice.

3 Methodology

In this chapter, we present the study design that was used. The research design is crucial in conducting empirical research and plays an important role in ensuring the validity and reliability of the results. To guide our study design, we follow the guidelines suggested by Petersen et al. (2008) on conducting mapping studies.

3.1 Study Objectives

The goal of the thesis, articulated using the Goal-Question-Metrics format (Basili et al., 1994) is:

To *analyze* existing empirical research studies in the field of software engineering related to the use of open-source software *for the purpose of* characterization and evaluation, *with respect to*: (a) the goals of each primary study; (b) the selection criteria for the open-source projects; and (c) the project selection, from the point of view of researchers.

3.2 Research Questions

Research questions guide the entire research process and provide a clear focus for the study. Based on the goal, we define the following research questions:

RQ1: What are the goals of MSR studies that use collections of projects mined from open repositories?

RQ1 delves into the primary goals of MSR (Mining Software Repositories) studies that utilize collections of works sourced from open repositories. This sheds light on their significance in enhancing software development practices and facilitating empirical research. The essential aims are derived from the research inquiries posed in the primary studies, such as testing, version control, and refactoring.

RQ2: What are the most common project selection criteria?

RQ2 examines the prevailing factors utilized for the selection of open-source projects, emphasizing their crucial role in guaranteeing project success. To delve deeper into this inquiry, we investigated to identify the most frequently encountered criteria for selecting open-source software projects, based on the objectives established in the primary studies.

RQ3: What are the most used projects as subjects?

RQ3 focuses on the frequently chosen projects that serve as subjects in open-source research, emphasizing their significance in gaining insights into software development and uncovering valuable information. Additionally, as part of our further investigation, we examined the most prevalent open-source projects based on the objectives outlined in the primary studies.

3.3 Bibliography research

According to the study of (Wong et al. 2021) we came up with the journals on which this thesis will be based. Having decided to search literature only from reputable journals over a two-year period, a manual literature review had then been conducted.

Initially the search strategy included the combination of keywords related to open-source software and empirical research (refer to the following frame). The initial search resulted in many papers (1492 papers). Below in table 3.1 you can see the number of papers studied per journal and the steps that were needed until we resulted in the papers that met all the criteria we had set.

{«open source» || «open-source» || «open projects»} && {«empirical study» || «experimental study» || «case study»}

3.4 Study selection criteria

To ensure the relevance and reliability of the papers selected for our study, we defined specific inclusion criteria (IC) and exclusion criteria (EC). The criteria for the selection of papers are as follows:

Inclusion Criteria of our mapping study are:

IC1. The paper must be an empirical study. (e.g., case study, research, experiment)

IC2. The research must have used open-source software to draw conclusions.

IC3. The paper must have been published within the last 2 years (2021 and 2022).

The exclusion Criteria of our mapping study are:

EC1. The language of the paper should be English.

EC2. The study is an editorial, keynote, biography, opinion, tutorial, workshop summary report, progress report, poster, or panel.

Table 3.1: Number of journals by stage

Venue	Total number of journals (1st round)	Total number of journals (2nd round)	Journals meeting all the criteria
JSS	343	69	60
IEEE-TSE	364	138	115
ESE	189	66	50
IST	319	100	80
EMSE	134	48	40
TOSEM	143	54	49
Total	1492	475	394

3.5 Study selection process

The study selection process involved multiple stages to ensure the relevance and quality of the papers and a simplified version of the voting method, proposed by Farhoodi et al., 2013, was used. First, we screened the papers based on their titles and abstracts. Papers that did not meet the inclusion criteria were excluded. In the second stage, we performed a full-text review of the remaining papers and assigned an inclusion value (vote) on a 4-point Likert scale (4: strong inclusion, 1: strong exclusion) - with a maximum score of 8 points. Studies with values greater than 6 were included in the study. Again, papers that did not meet the inclusion criteria were excluded.

Any disagreements regarding the inclusion of papers were resolved through discussion and consensus among the research team. The above description of the process is evident if we also look at Figure 3.1.

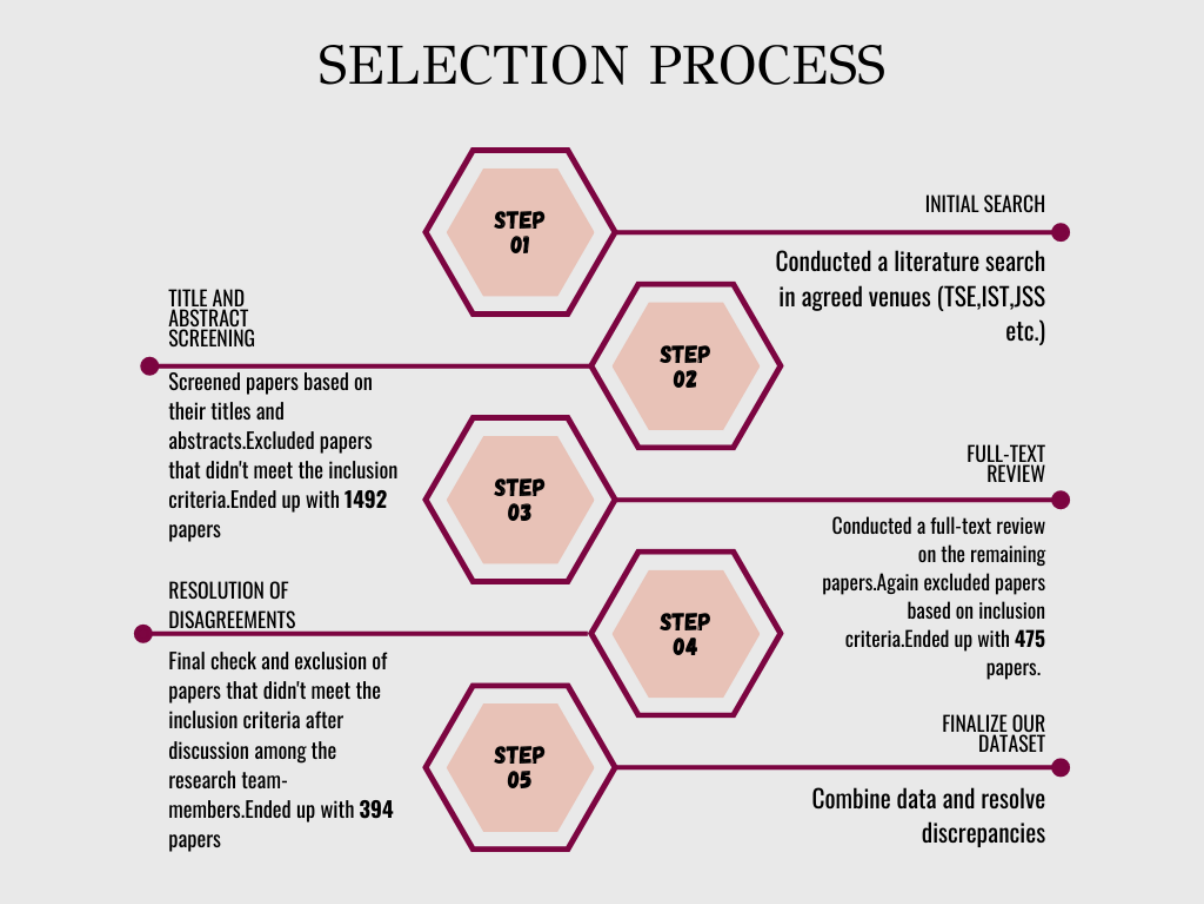

Figure 3.1: Selection process

3.6 Data analysis

For the selected papers, we extracted the relevant data using a predefined data mining template. The proposed procedure is largely based on synthesis and meta-analysis methods applied in the field of software engineering, as presented by Cruzes et al., 2011, dos Santos et al., 2020 and Kitchenham et al, 2020. Initially, we recorded all research questions from articles we studied in the corresponding research (e.g., "How frequent are code smells in Android applications?", "To what extent do corrective actions (refactoring) applied to smelly classes remove code smells?"). Here it should be emphasized that the research questions were recorded exactly as they appear in the studied articles, without any interference from us. In case an article has no research questions formulated, the field remains empty. At this point it is important to emphasize that in this work based on the systematic mapping study process, no quality assessment has been performed. For example, the use of DARE would have excluded studies without

research questions (Kitchenham et al., 2004). Next, we applied thematic analysis aiming to unify the research questions. To achieve this, we applied the Open Card Sorting methodology as proposed by Spencer (Spencer 2009). By studying the research questions, we extracted high-level objectives in the form of super-categories (for example, from the objective 'versions' we created the high-level objective 'version control'). Next, we identified high-level objectives with similar concepts which we unified (e.g., we unified the concepts "bug", "defect" and "fault"). Finally, we assigned a name to the unified categories (in the above example we gave the name "bug/fault/defect"). This methodology was also applied to other fields that were collected (selection criteria and open-source project name) and was particularly important since in open-source project names the list of names for the same software was very extensive (e.g., for Apache-camel there were references such as camel, camel-1.4, Apache/camel, camel-core etc.).

Therefore, for all the projects that participated in our mapping study we collected the following variables:

- [V1] **Title**: Title of the article
- [V2] **Author(s)**: Authors list
- [V3] **Year**: Year of publication of article
- [V4] **Type of Paper**: Article type
- [V5] **Publication Venue**: Name of the journal published.
- [V6] **Goal(s)**: The goals of each article (unified by the research questions)
- [V7] **Open-source Project(s)**: List of open-source software projects
- [V8] **Selection Criteria**: List of selection criteria for open-source projects
- [V9] **Open Dataset (Y/N)**: Data collection availability

From the very beginning of the research, the way in which the data would be collected and stored was decided between the researchers. For example, it had been decided that in case of multiple projects, goals, and selection criteria the separation of them would be done using the hash symbol (#). The purpose of this decision was to ensure consistency and to simplify the analysis of the data.

The variables [V1] - [V4] were collected primarily for their use in the website that we built and will be discussed in the next chapter. The remaining variables were used to answer the research questions. Table 3.2 shows the mapping between the research questions, the selected variables, and the data analysis methodologies.

To minimize any threat to the validity of the data collected, the supervising professor involved in the research always checked the data. Any ambiguities in the data were reviewed and resolved.

Table 3.2: Matching questions, variables, and data analysis methods

Research Question	Variable	Analysis method
RQ1	[V6], [V8]	Frequency tables, Crosstab Tables
RQ2	[V6], [V7]	Frequency tables, Crosstab Tables
RQ3	[V6]	Frequency tables

4 Results

4.1 Introduction

This chapter presents the results of the empirical study conducted to investigate the guidelines on the selection of open-source software collections in empirical studies.

In **4.2** we will discuss the goals of MSR studies that use open-source projects **(RQ1)**, next in **4.3** we will present the most common project selection criteria **(RQ2)** and finally in **4.4** the most commonly used open-source projects **(RQ3)**.

4.2 Goals of empirical studies (RQ1)

Table 4.1 shows the frequency of the occurrence of the goals in the empirical studies that were studied. We will report on the 5 most common targets that appeared in our data.

- *"Bugs/Defects/Faults"*: with the highest frequency of 84, this indicates that addressing and minimizing bugs, defects and errors remains a primary concern in software engineering. It underlines the importance of quality assurance and testing processes.
- *"Testing"*: with a frequency of 70, testing is a critical aspect of software engineering. It includes various testing methodologies, such as unit testing, integration testing and system testing, which are necessary to ensure software quality.
- *"Security"*: security, with a frequency of 37, is an important concern in software engineering. It emphasizes the need to address vulnerabilities and protect against potential threats such as unauthorized access and data breach.
- *"Localization"*: Localization, with a frequency of 28, indicates the importance of adapting software for specific languages, regions, and cultural contexts. This objective is particularly important for global software products.
- *"Vulnerabilities"*: a frequency of 26 indicates the focus on identifying and addressing vulnerabilities in software systems. This includes potential vulnerabilities that could be exploited to compromise system security.

The data extracted reflects the diverse and multifaceted nature of software engineering. They encompass a wide range of goals and focus areas, including bug fixing, testing, security, tracing, program debugging, version control, reconfiguration, requirements, and more. These objectives highlight the challenges and complexity involved in developing software systems of high quality, reliability, and safety.

The high frequency of objectives related to bugs, testing, security, and localization underscores their importance in software development. The importance of identifying and resolving defects, ensuring robust testing procedures, addressing security vulnerabilities and adapting software for different languages and regions is emphasized.

In addition, objectives such as program patching, version control and re-engineering point to ongoing efforts to improve code quality, maintainability, and collaboration within software projects. These objectives reflect the continuous evolution and improvement of software engineering practices as it is particularly important for industries to reduce maintenance costs which amount to 75% of total software development costs. (H. van Vliet, 1993)

In addition, the presence of objectives related to requirements, business models and documentation highlights the importance of aligning software development with customer needs, business objectives and effective communication.

Overall, the dataset demonstrates the broad scope of software engineering and the various dimensions that practitioners and researchers need to consider. It reinforces the need for an integrated approach to software development, which includes testing, security, localization, code quality, requirements management, and other critical aspects to ensure successful software systems.

Table 4.1: Frequency of goals

Goal	Freq.
Bugs / Defects / Faults	84
Testing	70
Security	37
Localization	28
Vulnerabilities	26
Program Repair	20

Goal	Freq.
Version Control	18
Refactoring	18
Features / Requirements / Business Models	18
Bad Smells	17
Code/Test Generation	15
Dependency Analysis	15
Performance / Resource Management / Time Behaviour	15
Human Factors	14
Code Review	14
Application Programming Interfaces (APIs)	14
Technical Debt	13
Software Change	12
Documentation	12
Quality Metrics	11
Architecture	11
Static Analysis	11
Data / Information	10
Management	8
Cost Analysis	8
Code Clones	8
Execution Traces	8
Software Libraries	7
Traceability	7
Logs	7
Debugging / Bug Fixing	6
Comments	6
Software Product Lines (SPL) / Reconfigurable Systems	6
Software Design	5
Maintenance	5
GUI	4
Search Based Software Engineering (SBSE)	4

Goal	Freq.
Software Quality Assurance Processes	4
Change Impact Analysis (CIA)	5
Software Patterns	4
Component Based Software Engineering (CBSE)	4
Software Crashes	4
Effort Estimation	4
Complexity	4
Network	4
Software Build	3
User Reviews	3
Privacy	3
Modularity	3
Code Transformation / Compilers	3
CI/CD	3
Team Management	3
Service Oriented Architectures (SOA)	3
User Experience	3
Aesthetics	2
Slicing	2
Agile Methodologies	2
Reuse	2
Dynamic Analysis	2
Testability	2
Program Comprehension	2
Exception Handling	2
Parallelization	2
Cloud-Based Software	2
Business Parameters	2
Domain Specific Languages (DSL)	1
Software Analytics	1
Software Anomalies	1

Goal	Freq.
Deployment	1
Code Annotations	1
Application Domain Analysis	1
Reliability	1
User Behaviour	1

4.3 Criteria for selecting open-source projects (RQ2)

Table 4.2 presents the criteria for project selection, along with their respective frequencies, as derived from our mapping study. The findings reveal that researchers commonly consider certain criteria when selecting projects. Among them, the use of *"Java"* as a programming language ranks the highest with a frequency of 78 (10.17%). This is followed by the criterion of *"Widely Used/Popular"* with a frequency of 67 (8.74%). The choice of Java as a language is in line with expectations due to the plethora of tools available for Java code analysis. The emphasis on popularity stems from the researchers' desire to select projects that are familiar to readers and provide assurance that the projects are actively developed and embraced by the community. On the other hand, *"Inactive Projects"* and projects with *"Random Selection"* were found to be used less frequently as project selection criteria, with a frequency of 4 times (0.52%). It is important to note that the term 'random selection' should not be confused with the concept of diversity, which involves a systematic (rather than random) selection of projects based on factors such as size and history.

When examining programming languages, *"Java"* emerged as the most frequently cited language with a frequency of 78 (10.17%). This was followed by the *"C-Family (C/C++/C#)"* with a frequency of (2.74%), *"Python"* with a frequency of 7 and *"JS/JavaScript"* with a frequency of 5. In total, programming languages accounted for 15.91% of the total.

Criteria related to project management features accounted for a total frequency of 14.73%. These criteria are particularly important in studies that aim to extract automated information using tools already used by the project development team. In particular, the criterion *"Version Control Information"* had a frequency of 43 (5.61%), while the

criterion *"Development Community Information"* appeared with a frequency of 22 (2.87%). The *"Issue Tracker Information"* appeared 13 times (1.69%).

Criteria related to project size and activity were also considered important. Project size serves as a critical criterion for empirical studies, ensuring the exclusion of very small projects that may not be comparable to industrial projects. The level of activity is important for articles studying software development, requiring a sufficient number of software versions for statistical analysis. *"Size Information (In Other Metric)"* was the most frequently mentioned criterion, appearing 41 times (5.35%). Specific criteria such as *"Size Information (In Loc)"* and *"Size Information (In Modules)"* appeared in 29 and 18 studies, respectively. Information on *"Project Age/Maturity"* appeared in 39 studies (5.08%), followed by *"Intensity Of Development Activity"* with 28 studies. In contrast, *"Inactive Projects"* had the lowest frequency with 4 occurrences (0.52%), as it does not prove to be a useful criterion for selecting open-source projects. This group represented the highest overall frequency, with a percentage of 23.99%.

A selection criterion based on previous literature had an overall frequency of 15.78%. Sub-criteria such as *"Widely used/Popular"* and *"Used In Previous Studies"* had frequencies of 67 (8.74%) and 54 (7.04%), respectively. This criterion aims to replicate empirical studies and allows comparisons of results.

In addition, criteria such as *"Various Domains"* had a frequency of 39 (5.08%), followed by *"Diversity In Size"* with a frequency of 25. *"Random Selection"* had a frequency of 4 (0.52%). The diversity group, in contrast to the literature-based group, had the lowest overall frequency of 5.60%. Nevertheless, it is considered vital for generalizability/external validity as it helps to mitigate the influence of confounding factors.

In terms of technologies, the most frequent criterion was *"Use Of Best Practices"* with a frequency of 54 (7.04%), followed by *"Specific Application Domain or Technology"* with a frequency of 45 (5.87%). The criterion related to *"Feature/Functionality/Requirements Criteria"* had the lowest frequency among the technologies as it appeared only 11 times (1.43%). The overall frequency for the technology-related criteria was 16.04%. Researchers take these criteria into account when targeting specific technologies (e.g., design standards, code smells) or application domains (e.g., web, services, cloud).

Finally, criteria related to testability, such as *"Tested Systems"* and *"Buildable Systems"*, had a frequency of 7.95%. These criteria find relevance in studies related to software testing or when using data collection tools that require code execution.

Table 4.2: Criteria for selecting open-source projects

Selected Criteria	Freq.
Java	78
Widely Used / Popular	67
Use Of Best Practices	56
Used In Previous Studies	54
Specific Application Domain or Technology	45
Tested Systems	43
Version Control Information	43
Size Information (In Other Metric)	41
Project Age / Maturity	39
Various Domains	39
Size Information (In Loc)	29
Intensity Of Development Activity	28
Diversity In Size	25
Development Community Information	22
C-Family (C/C++/C#)	21
Bug Tracker Information	20
Buildable Systems	18
Size Information (In Modules)	18
Hosted In Git/GitHub	15
Issue Tracker Information	13
Other Programming Language	11
Service-Oriented Software	11
Feature / Functionality / Requirements Criteria	11
Python	7
JS/JavaScript	5

Inactive Projects	4
Random Selection	4

Below in Table 4.3 we present the cross-tabulations between the selection criteria and goals. This table is particularly important to future researchers as it provides a clear mapping and alignment between the research goals and the selection criteria. It promotes consistency, standardization, and integrated analysis of the research framework, facilitating communication and iterative improvements. This structured approach enhances the quality of research and leads to impactful results. As can be seen from the table below it is evident that if one aims to study *"Bugs//Defects/Faults"* the most common criteria for selecting open-source projects in previous work are that the projects are in Java, are tested and have been used in previous studies (frequency of occurrence 17, 15 and 16 respectively).

Table 4.3: Cross-tabulations between selection criteria and goals

Final Goal	Criteria																									
	Bug Tracker Information	Buildable Systems	C-Family (C/C++/C#)	Development Community Information	Diversity In Size	Feature / Functionality / Requirements Criteria	Hosted In Git/GitHub	Inactive Projects	Intensity Of Development Activity	Issue Tracker Information	Java	JS/JavaScript	Other Programming Language	Project Age / Maturity	Python	Random Selection	Service-Oriented Software	Size (In Loc)	Size (In Modules)	Size (In Other Metric)	Specific Application Domain Or Technology	Tested Systems	Use Of Best Practices	Used In Previous Studies	Various Domains	Version Control Information	Widely Used / Popular
Bugs / Defects / Faults	14	1	6	4	6	1	3		7	6	17	1		10	2			6	6	12	6	15	4	16	8	9	12
Testing	1	5	7	4	7	2	5		1		9		2	7		1	1	8	2	4	12	21	5	13	7	2	9
Security	3	2	4		2			1	3		6		1	2	1			1		4	6	1	9	5	1	4	5
Localization	5	1	2	1	3	1	1		3	2	5			1				2	1	2	2	7	1	5	4		4
Vulnerabilities	2	2	3						1		5		1	1	1						3	1	8	3	1	2	4
Program Repair	2	3			2				5		4		1	2			1	1	1	1	4	2	3	2	1		2
Version Control	1		1	2	2	1	2	1	3	2	7			2	1					2	3	1	3	3		7	2
Refactoring				1	1				2		6	2		2				1	1	2			5	1		4	5

Final Goal	Criteria																								
	Bug Tracker Information	Buildable Systems	C-Family (C/C++/C#)	Development Community Information	Diversity In Size	Feature / Functionality / Requirements Criteria	Hosted In Git/GitHub	Inactive Projects	Intensity Of Development Activity	Issue Tracker Information	Java	JS/JavaScript	Other Programming Language	Project Age / Maturity	Python	Random Selection	Service-Oriented Software	Size (In Loc)	Size (In Modules)	Size (In Other Metric)	Specific Application Domain Or Technology	Tested Systems	Use Of Best Practices	Used In Previous Studies	Various Domains	Version Control Information	Widely Used / Popular
Features / Req. / Business models					3	2				1	3			1		1		3		3	1	4	7	3	4	1	2
Bad Smells		2	2	1	2		2		2		5			2				3	1		1		3	3	2		3
Code/Test Generation	1	1					2				4		1	1			1	1	2		2	2			2	1	3
Dependency Analysis	1	1			1	1		1	1		6		1		1		1	1		2	3	2	5		1	3	4
Performance / Resource Management / Time Behaviour		2	1			1	1				1	1		1		1	1	1	1	1	4	3	3	1	2	1	3
Human Factors	2			4	1		1		1	1				3				1	1	1	1		3		2	3	3
Code Review				1				1	3		1			1					1	2	1		5	2			2

Final Goal	Criteria																								
	Bug Tracker Information	Buildable Systems	C-Family (C/C++/C#)	Development Community Information	Diversity In Size	Feature / Functionality / Requirements Criteria	Hosted In Git/GitHub	Inactive Projects	Intensity Of Development Activity	Issue Tracker Information	Java	JS/JavaScript	Other Programming Language	Project Age / Maturity	Python	Random Selection	Service-Oriented Software	Size (In Loc)	Size (In Modules)	Size (In Other Metric)	Specific Application Domain Or Technology	Tested Systems	Use Of Best Practices	Used In Previous Studies	Various Domains	Version Control Information	Widely Used / Popular
Application Programming Interfaces (APIs)		1		1		1			2	1	5		1	1			5	3			1	1	4	2	1	1	4
Technical Debt		1	1	2				1	1	3	3			3	1			2	2	1	3		2		1	4	1
Software Change		1			3	1			2		3		2	2						2	1	1	2		3	2	3
Documentation	2			2					1	2	3		1	2				1	1	2			1	1	2		1
Quality Metrics	1	1		1	2	1	2		2	1	5			2					1		1	2		3	1	1	2
Architecture		1			1						3						1		1	1	3		1	1	1		1
Static Analysis		1	1			1	1		1	1	3			2						1	1	1	3	1		2	
Data / Information	1								1	2	3			1	1			2					1	4		2	2
Management	1		1				1	1	2	1	1		1	3					1	1	1		2		1	4	3

Final Goal	Criteria																									
	Bug Tracker Information	Buildable Systems	C-Family (C/C++/C#)	Development Community Information	Diversity In Size	Feature / Functionality / Requirements Criteria	Hosted In Git/GitHub	Inactive Projects	Intensity Of Development Activity	Issue Tracker Information	Java	JS/JavaScript	Other Programming Language	Project Age / Maturity	Python	Random Selection	Service-Oriented Software	Size (In Loc)	Size (In Modules)	Size (In Other Metric)	Specific Application Domain Or Technology	Tested Systems	Use Of Best Practices	Used In Previous Studies	Various Domains	Version Control Information	Widely Used / Popular
Cost Analysis			1		2				1		2			1				1			1	1		2	4		1
Code Clones				1	1						3		1	1						1		1	4	1	2	1	2
Execution Traces			1						1						1	1				1	1			1	1		2
Software Libraries				1		1		1	2					2		1				2	3		1			2	1
Traceability				1	1						2							1		1	2		2		1	1	1
Logs	1			1							2			2				1		4			3	3	2		1
Debugging / Bug Fixing	1		1						1		1		1	1												2	
Comments	1			2	1						2		1	1					1				1		1	1	1
SPL / Reconfigurable Systems					1	1								1						1				1	1		1

Final Goal	Criteria																										
	Bug Tracker Information	Buildable Systems	C-Family (C/C++/C#)	Development Community Information	Diversity In Size	Feature / Functionality / Requirements Criteria	Hosted In Git/GitHub	Inactive Projects	Intensity Of Development Activity	Issue Tracker Information	Java	JS/JavaScript	Other Programming Language	Project Age / Maturity	Python	Random Selection	Service-Oriented Software	Size (In Loc)	Size (In Modules)	Size (In Other Metric)	Specific Application Domain Or Technology	Tested Systems	Use Of Best Practices	Used In Previous Studies	Various Domains	Version Control Information	Widely Used / Popular
Software Design			1		2				2		3														3		
Maintenance					1						2			1				1			1		1	2	2		
GUI														1			1				2	1	1	1			
Search Based Software Engineering											2						1										2
Software Quality Assurance Processes		2							1				1	2					1	1		1	1	1	1	2	
Change Impact Analysis (Cia)					1				1		2		1							1		1	2		3	1	1
Software Patterns		1			2						3									1			1	1	1		

Final Goal	Criteria																									
	Bug Tracker Information	Buildable Systems	C-Family (C/C++/C#)	Development Community Information	Diversity In Size	Feature / Functionality / Requirements Criteria	Hosted In Git/GitHub	Inactive Projects	Intensity Of Development Activity	Issue Tracker Information	Java	JS/JavaScript	Other Programming Language	Project Age / Maturity	Python	Random Selection	Service-Oriented Software	Size (In Loc)	Size (In Modules)	Size (In Other Metric)	Specific Application Domain Or Technology	Tested Systems	Use Of Best Practices	Used In Previous Studies	Various Domains	Version Control Information	Widely Used / Popular
Component Based Software Engineering																			1	1	1		1	1			1
Software Crashes	1	1														1		1		1		2					
Effort Estimation						1			1		1									1	1		1	1			
Complexity													1	1						1			1				1
Network			1		1						1			1			1			1	3				1		2
Software Build		2									1	1		1							1	1		1			
User Reviews	1				1						1							1		1	1		2		1		
Privacy																					1		2				
Modularity														1												1	1

Final Goal	Criteria																									
	Bug Tracker Information	Buildable Systems	C-Family (C/C++/C#)	Development Community Information	Diversity In Size	Feature / Functionality / Requirements Criteria	Hosted In Git/GitHub	Inactive Projects	Intensity Of Development Activity	Issue Tracker Information	Java	JS/JavaScript	Other Programming Language	Project Age / Maturity	Python	Random Selection	Service-Oriented Software	Size (In Loc)	Size (In Modules)	Size (In Other Metric)	Specific Application Domain Or Technology	Tested Systems	Use Of Best Practices	Used In Previous Studies	Various Domains	Version Control Information	Widely Used / Popular
Code Transformation / Compilers			1								1													1			
CI/CD							1		1		2			1						1				2		2	
Team Management				1	1									1						1					1	1	
Service Oriented Architectures																	1		1	1	1		1	1			
User Experience																					2		1		1	1	1
Aesthetics											1						1										1
Slicing			1															1				1					
Agile Methodologies									1												1		1				1

Final Goal	Criteria																										
	Bug Tracker Information	Buildable Systems	C-Family (C/C++/C#)	Development Community Information	Diversity In Size	Feature / Functionality / Requirements Criteria	Hosted In Git/GitHub	Inactive Projects	Intensity Of Development Activity	Issue Tracker Information	Java	JS/JavaScript	Other Programming Language	Project Age / Maturity	Python	Random Selection	Service-Oriented Software	Size (In Loc)	Size (In Modules)	Size (In Other Metric)	Specific Application Domain Or Technology	Tested Systems	Use Of Best Practices	Used In Previous Studies	Various Domains	Version Control Information	Widely Used / Popular
Re-use		1		1							1			1	1												2
Dynamic Analysis																						1	1				
Testability																	1	1		1							
Program Comprehension											1							1					1				
Exception Handling											1							1	1	1							
Parallelization			1															1				1					
Cloud-Based Software																	1				1					1	
Business Parameters				1							1								1	1					1	1	

Final Goal	Criteria																										
	Bug Tracker Information	Buildable Systems	C-Family (C/C++/C#)	Development Community Information	Diversity In Size	Feature / Functionality / Requirements Criteria	Hosted In Git/GitHub	Inactive Projects	Intensity Of Development Activity	Issue Tracker Information	Java	JS/JavaScript	Other Programming Language	Project Age / Maturity	Python	Random Selection	Service-Oriented Software	Size (In Loc)	Size (In Modules)	Size (In Other Metric)	Specific Application Domain Or Technology	Tested Systems	Use Of Best Practices	Used In Previous Studies	Various Domains	Version Control Information	Widely Used / Popular
Domain Specific Languages																											
Software Analytics																											
Software Anomalies																											1
Deployment																	1									1	
Code Annotations							1				1									1							
Application Domain Analysis				1				1																			
Reliability																											
User Behaviour															1						1						1

4.4 Widely used open-source software (RQ3)

According to Table 4.4, the most used projects in the empirical studies examined were apache-camel and apache-commons-lang, with frequencies of 48 (2.74%) and 43 (2.74%) respectively. In contrast, there were over 1000 projects that were studied only once; to save space, in the table we decided to present projects that were used in at least 4 articles. The top 10 projects with the highest frequency are Apache Camel, Apache Commons Lang, Apache Commons Math, Apache Ant, Apache Lucene, Apache Log4j, jEdit, JFreeChart, Spring Framework and Apache HBase. These projects cover various domains such as integration, programming tasks, math and statistics, manufacturing automation, search functions, logging, word processing, charting, business application development, and NoSQL database management.

Table 4.4: Open-source software

Project	Freq	Project	Freq
apache-camel	48	weka	6
apache-commons-lang	43	curl	6
apache-commons-math	39	apache-commons configuration	6
apache-ant	35	apache-commons-net	6
apache-lucene	33	apache-kylin	6
apache-log4j	28	apache-avro	6
jedit	27	jenkins	6
jfreechart	25	netty	6
spring framework	24	apache-beam	6
apache-hbase	24	apache-flume	6
apache-cassandra	24	jetty	6
apache-xalan	23	petclinic	5
apache-hive	23	totinfo	5
apache-hadoop	23	replace	5
apache-commons-io	21	libtiff	5

Project	Freq	Project	Freq
apache-xerces	21	apache-spark	5
apache-commons-closure	21	antennapod	5
apache-wicket	19	couchbase	5
apache-poi	19	bugzilla	5
eclipse-jdt	19	columba	5
eclipse-core	19	postgresql	5
apache-ivy	18	equinox	5
apache-commons-collections	17	apache-deltaspike	5
apache-derby	16	apache-giraph	5
apache-activemq	16	apache-jspwiki	5
apache-tomcat	16	apache-knox	5
github organization	16	apache-nutch	5
apache-commons-mockito	16	apache-opennlp	5
joda-time	16	apache-santuario	5
apache-zookeeper	16	jabref	5
apache-velocity	15	netflix organization	5
chart	14	apache-httpcomponents	5
apache-flink	14	apache-struts	5
elasticsearch	13	hsqldb	5
google-guava	13	apache-jena	5
apache-synapse	13	pmd	5
apache-commons-codec	13	coreutils	5
apache-storm	12	wireshark	5
pde	12	libreoffice	5
argouml	12	tensorflow	5
apache-jmeter	12	apache-phoenix	5
apache-groovy	12	gcc	5
junit	11	apache-hdfs	5
apache-mahout	11	wordpress	5
apache-jackrabbit	10	notepad	5

Project	Freq	Project	Freq
gzip	10	chromium	4
qt	10	apache-qpid	4
google-gson	10	sed	4
apache-cxf	10	eq	4
hibernate	10	space	4
apache-maven	10	ansible	4
apache-kafka	10	geronimo	4
jackson-core	10	apache-mesos	4
apache-ambari	9	sentry	4
openstack	9	apache-zeppelin	4
reactivex-rxjava	9	jgit	4
apache-mylyn	9	connectbot	4
apache-cayenne	9	javaparser-organization	4
zxing	9	apache-jxpath	4
alibaba-fastjson	9	okhttp	4
apache-commons-bcel	9	busybox	4
mozilla-organization	9	nova	4
ffmpeg	9	apache-commons-digester	4
apache-jruby	9	apache-commons-vfs	4
apache-organization	9	apache-lens	4
time	8	apache-manifoldcf	4
checkstyle	8	tika	4
apache-commons-cli	8	apache-tez	4
jsoup	8	gitlab	4
linux	8	rhino	4
apache-calcite	8	zipkin	4
apache-commons-compress	8	matplotlib	4
apache-accumulo	8	react	4
apache-dubbo	8	apache-drill	4
django	8	apache-jclouds	4
pandas	7	colt	4

Project	Freq	Project	Freq
k-9 mail	7	eclipse-emf	4
apache-openjpa	7	fresco	4
apache-pig	7	php	4
jhotdraw	7	wget	4
android	7	apache-druid	4
apache-ignite	7	asterisk	4
apache-commons-beanutils	7	promise	4
apache-commons-dbcp	7	apache-aries	4
apache-commons-validator	7	keras	4
apache-pdfbox	7	gerrit	4
openssl	7	firefox	4
python	7	vlc	4
swt	6	qemu	4
eclipse	6	deeplearning4j	4
apache-karaf	6	aspectj	4
apache-thrift	6	quantum	4
apache-flex	6	apache-commons-csv	4
grep	6	apache-bookkeeper	4
squirrel	6	printtoken	4
freemind	6	tcas	4

Below in Table 4.5 we present the cross-tabulations frequencies between goals and open-source projects. Frequency cross-tabulations between goals and projects are useful for researchers as they help to identify project-target correlation, aim research efforts at high-frequency cross-tabulations for greater impact, explore unexplored areas, facilitate collaboration and networking, perform comparative analysis, and support evidence-based decision making.

Table 4.5: Correlation between goals and open-source projects

Final Goal	Projects	Freq.
Bugs / Defects / Faults	apache-camel	29

Final Goal	Projects	Freq.
	apache-lucene	19
	apache-commons-lang	19
	apache-commons-math	19
	apache-xalan	17
	apache-poi	16
	apache-ivy	16
	apache-log4j	15
	eclipse-jdt	15
	apache-ant	14
	jedit	14
	apache-velocity	14
	apache-xerces	13
	apache-commons-closure	13
	pde	12
	apache-hbase	12
	apache-hive	12
	apache-synapse	12
	apache-activemq	11
	apache-derby	10
	apache-tomcat	9
	apache-commons-collections	9
	chart	8
	apache-commons-mockito	8
	apache-mylyn	7
	mozilla-organization	7
	apache-commons-codec	7
	apache-commons-io	7
	apache-wicket	7
	time	6
	apache-commons-compress	6
	apache-zookeeper	6

Final Goal	Projects	Freq.
	apache-groovy	6
	apache-storm	6
	jackson-core	6
	swt	5
	zxing	5
	joda-time	5
	jfreechart	5
	postgresql	5
	equinox	5
	apache-calcite	5
	apache-commons-dbcp	5
	apache-kylin	5
	apache-jruby	5
	eclipse-core	5
	apache-hadoop	5
	eclipse	4
	apache-ambari	4
	apache-cayenne	4
	eq	4
	apache-commons-bcel	4
	apache-commons-beanutils	4
	apache-commons configuration	4
	apache-commons-net	4
	apache-commons-validator	4
	apache-commons-vfs	4
	apache-mahout	4
	apache-avro	4
	apache-tez	4
	apache-commons-cli	4
	apache-flink	4
	apache-ignite	4

Final Goal	Projects	Freq.
	aspectj	4
	quantum	4
	gzip	4
	apache-openjpa	3
	lc	3
	ml	3
	safe	3
	bugzilla	3
	apache-archiva	3
	apache-commons-digester	3
	apache-commons-jcs	3
	apache-commons-jexl	3
	apache-deltaspike	3
	apache-giraph	3
	apache-jspwiki	3
	apache-knox	3
	apache-lens	3
	apache-nutch	3
	apache-parquet	3
	apache-santuario	3
	apache-accumulo	3
	apache-beam	3
	apache-httpcomponents	3
	apache-jxpath	3
	apache-cassandra	3
	apache-cxf	3
	apache-kafka	3
	apache-commons-csv	3
	elasticsearch	3
	geronimo	3
	printtoken	3

Final Goal	Projects	Freq.
	tcas	3
	totinfo	3
	apache-flex	3
	apache-aries	3
	apache-organization	3
	roslyn	3
Localization	apache-commons-lang	14
	apache-commons-math	14
	apache-commons-closure	12
	chart	8
	apache-commons-mockito	8
	time	6
	jackson-core	6
	apache-hive	5
	apache-camel	4
	joda-time	4
	jfreechart	4
	apache-commons-compress	4
	apache-commons-collections	4
	apache-hbase	4
	apache-commons-codec	3
	swt	3
	aspectj	3
	eclipse-jdt	3
	apache-commons-csv	3
	gzip	3
	printtoken	3
	tcas	3
	totinfo	3
Testing	apache-commons-lang	13
	apache-commons-math	12

Final Goal	Projects	Freq.
	joda-time	7
	gzip	5
	grep	5
	apache-commons-io	5
	jfreechart	5
	apache-hadoop	5
	google-guava	5
	apache-flex	4
	sed	4
	replace	4
	apache-commons-closure	4
	apache-commons-collections	4
	alibaba-fastjson	4
	google-gson	4
	apache-ant	4
	apache-cassandra	4
	apache-flink	4
	apache-hbase	4
	petclinic	4
	apache-jmeter	4
	apache-log4j	4
	totinfo	3
	make	3
	weka	3
	apache-camel	3
	apache-hive	3
	apache-karaf	3
	restcountries	3
	apache-commons-codec	3
	colt	3
Code Review	qt	8

Final Goal	Projects	Freq.
	openstack	5
	eclipse-core	5
	couchbase	4
	android	3
Program Repair	apache-commons-closure	7
	apache-commons-lang	7
	apache-commons-math	7
	chart	4
	time	3
	apache-commons-mockito	3
	libtiff	3
	jfreechart	3
	joda-time	3
	apache-camel	3
Security	github organization	6
	ffmpeg	5
	coreutils	3
	linux	3
	wireshark	3
Logs	apache-hadoop	6
	apache-zookeeper	6
	apache-hdfs	3
	apache-activemq	3
Statistical Analysis	apache-cassandra	5
	apache-ant	3
Bad Smells	jfreechart	5
	ganttproject	4
	apache-ant	4
	apache-xerces	4
	apache-cassandra	4
	checkstyle	3

Final Goal	Projects	Freq.
	jasperreports	3
	apache-lucene	3
	squirrel	3
	eclipse-core	3
	junit	3
	apache-camel	3
	apache-hive	3
Version Control	apache-hadoop	5
	elasticsearch	4
	apache-activemq	3
	apache-camel	3
	apache-derby	3
	apache-hbase	3
	apache-openjpa	3
	spring framework	3
Vulnerabilities	github organization	5
	ffmpeg	4
	coreutils	3
Quality Metrics	apache-commons-lang	4
	joda-time	4
	eclipse-jdt	4
	apache-lucene	4
	apache-cassandra	4
	apache-hbase	4
	apache-commons-math	3
	apache-commons-closure	3
	jfreechart	3
	apache-commons-mockito	3
	checkstyle	3
	apache-commons-io	3
	junit	3

Final Goal	Projects	Freq.
	pde	3
	apache-ant	3
	apache-camel	3
	apache-hive	3
	apache-wicket	3
	netty	3
	apache-xerces	3
Technical Dept	quantum	4
	apache-beam	3
	apache-dubbo	3
	apache-pdfbox	3
	apache-ant	3
	argouml	3
	columba	3
	eclipse-emf	3
	hibernate	3
	jedit	3
	jfreechart	3
	apache-jmeter	3
	apache-jruby	3
	squirrel	3
Code/Test Generation	freemind	4
	weka	3
Dependency Analysis	github organization	4
	apache-thrift	3
Comments	elasticsearch	3
	google-guava	3
Application Programming Interfaces	apache-log4j	3
Documentation	hibernate	3
Effort Estimation	spring framework	3

Final Goal	Projects	Freq.
Human Factors	apache-lucene	3

As we can observe from the table above, most open-source projects belong to the Apache "family". Also, in almost every goal, if not all, researchers use Apache software. The above table is particularly useful to new researchers, giving them the appropriate guidance in selecting software according to the objectives they have set in their research.

Finally, in Figure 4.1 we can see the tendency of researchers to make the results of their research available to the public. Of the total of 394 empirical studies, 121 did not have their data available, while correspondingly 273 did.

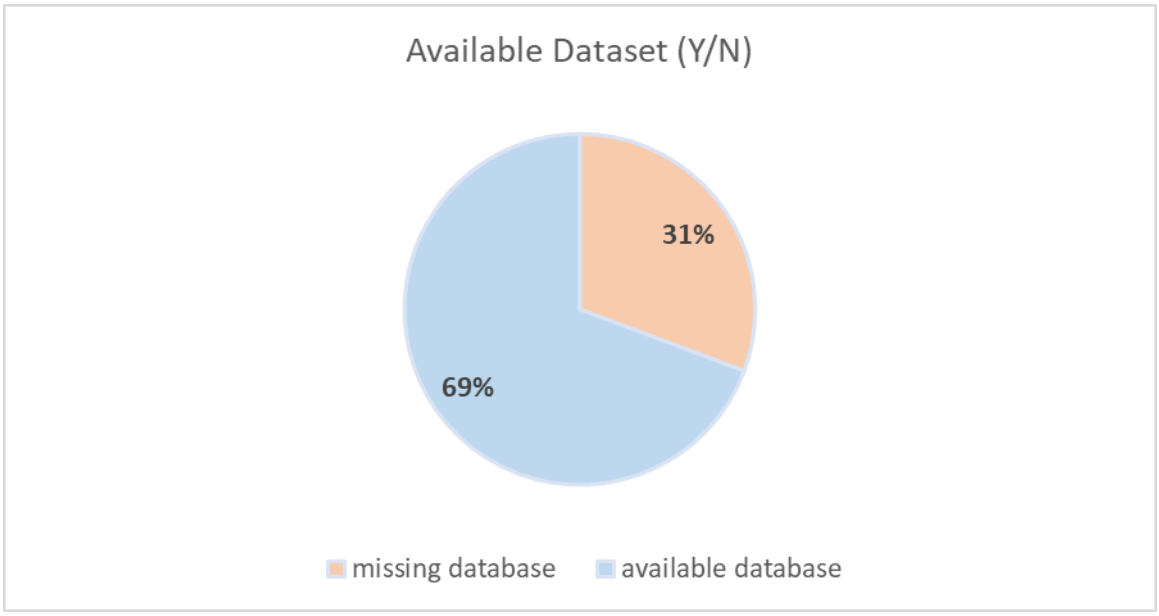

Figure 4.1: Available dataset

5 Website Presentation

5.1 Introduction

In this chapter, we will give a detailed description of the website developed for our thesis. The website serves as the main interface for users to interact with the system and access various functions. We will briefly mention the front-end technologies used. In addition, we will delve into the back-end technologies used, such as .NET 6 to create a REST API with basic authentication. The API is connected to a PostgreSQL database and the entire backend is executed through a Docker container.

5.2 Website Presentation

In the following link we can see the website that we have built, http://195.251.210.147:3030/ .

Essentially, the website is an empirical research database, and the platform is designed to facilitate future researchers. The home page presents an extensive collection of empirical studies, each distinguished by their title. By simply clicking on the accompanying icon, you can unlock additional information pertaining to each entry.

Recognizing the crucial importance of an efficient search engine, we have included a search area at the top of the page. In this designated area, you can enter specific titles to quickly locate the desired entries. For a more sophisticated exploration, the advanced search function allows you to drill down into papers based on open-source project names, researcher goals, and the selection criteria of open-source projects. The "Top Projects", "Top Goals" and "Top Criteria" buttons serve as inputs to the most notable and established entries in each respective category.

Finally, logged-in users have the additional possibility to contribute to our website, expanding our database by entering new empirical studies, thus enriching the scientific dialogue and promoting academic collaboration. Below we have some illustrative images from browsing the website.

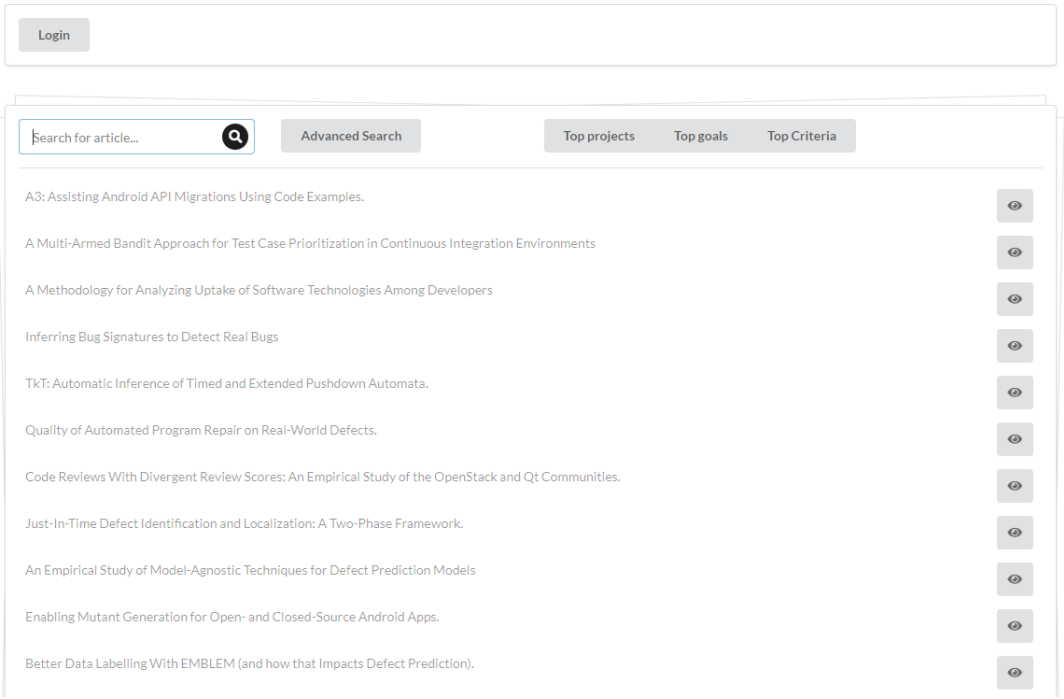

Figure 5.1: Home page

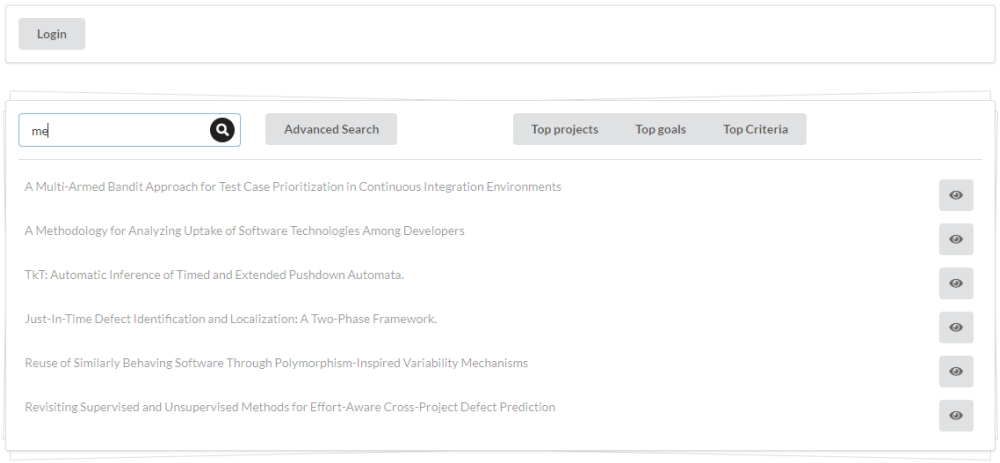

Figure 5.2: Advanced search feature

Figure 5.3: Information from an empirical study

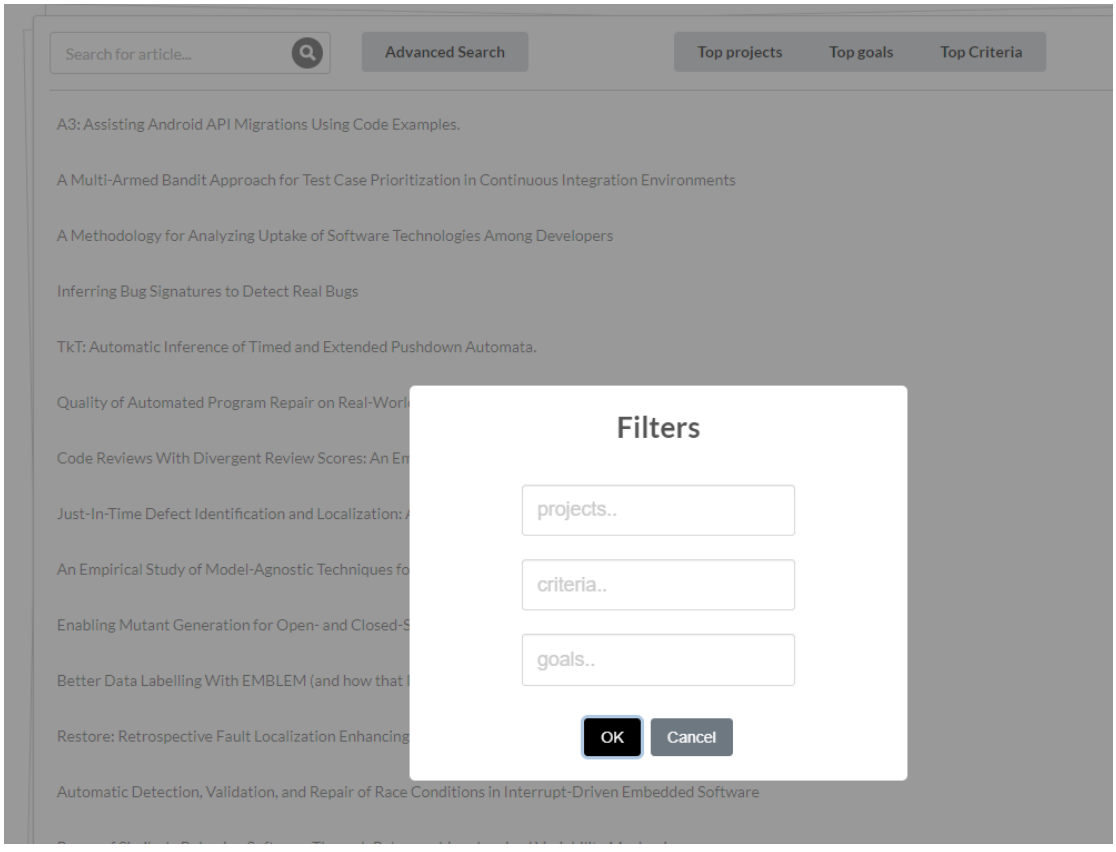

Figure 5.4: Search based on open-source project, selection criterion and goal

5.3 Website Development

Frontend development

In order for the reader to have a complete picture of the website we created; we will make a reference to the technologies used. More information on the Frontend part can be found in the thesis of my colleague, Anna Zivoni (Zivoni, 2023). Some of the technologies used are:

1. jQuery: a fast and concise JavaScript library, was used to simplify the process of DOM handling and event handling. Its extensive feature set facilitated dynamic and interactive user experiences.
2. React Fragment: a feature introduced in React 16, was used to group multiple elements without adding unnecessary markup. It allowed us to create a cohesive user interface while keeping the HTML structure clean and clear.
3. Babel: a JavaScript compiler, was used to convert the modern ECMAScript syntax into browser-compatible versions. This allowed us to write code using the latest JavaScript features while ensuring compatibility between browsers.
4. Semantic UI: a user interface framework, provided a comprehensive set of ready-to-use elements and styles. It allowed us to create an aesthetically pleasing and responsive design with minimal effort.
5. SweetAlert Popup: a JavaScript library, was used to display attractive and customizable popup messages to users. It improved the user experience by providing visually appealing and informative notifications.

Backend development

a) REST API

To develop the backend, we used .NET 6, a flexible and powerful development framework, to create a REST API. The REST API acts as a communication bridge between the frontend and the database, allowing data retrieval, manipulation, and storage.

RESTful APIs are built upon the architectural style introduced by Roy Fielding in his doctoral dissertation. (R.T.Fielding,2000) This style emphasizes scalability, statelessness, and a client-server interaction model. The principles of REST guide us in

designing APIs that are both robust and easy to understand. Below are some key principles that have influenced the design of our REST API:

Stateless Communication

Each API request from a client to the server must contain all the information needed to understand and process the request. This means that sessions are not stored on the server between requests, making the system more scalable and manageable.

Client-Server Architecture

In a RESTful system, the client and server are separate entities that interact over HTTP. The client handles the user interface and user experience, while the server manages the data and the backend logic. This separation allows each to evolve independently of the other.

Resource-Based

RESTful APIs operate on the concept of "resources," which are essentially objects or data entities that can be manipulated using standard HTTP methods like GET, POST, PUT, and DELETE. Resources are identified by URLs, making it easy to perform operations on them.

State Representation

When a client interacts with a resource, they are manipulating its state. The resource's current state is represented when it is fetched (typically, in a JSON or XML format). This state can be manipulated by the client, and changes can be stored back on the server.

Uniform Interface

RESTful APIs have a uniform and consistent interface, which simplifies interactions and enhances usability. Operations are standardized through HTTP methods:

GET: Retrieve a resource

POST: Create a new resource

PUT: Update an existing resource

DELETE: Remove a resource

Layered System

REST allows for a layered system where each layer has a specific role and responsibility. For instance, one layer might handle caching, another might deal with authentication, and yet another might handle business logic. This separation of concerns simplifies maintenance and scalability.

Below in Figure 5.5 we can see the methods and ways in which the user can communicate with our database. The name of the methods is such that anyone using this API will be able to understand what data will be returned from the database. The way that these methods work can be seen in Figure 5.6. For example, we chose to demonstrate the GET method "*/get/data/by/project/{project}*". To work, this method needs to connect with the database using Npgsql (Npgsql is an open-source package for accessing PostgreSQL database server which you can download to your project through NuGet). Npgsql allows you to execute SQL statements directly from your C# project and returns the content that you requested as a stream of rows. Then we handle that data and present it in the form of a JSON file. As an illustration in Figure 5.7 we have some dummy data from the first steps of creating the backend.

To ensure secure access to the system, we implemented basic authentication in the REST API. Basic authentication requires users to provide their login credentials (username and password) for authentication before accessing sensitive data. This authentication mechanism helps protect sensitive data and ensures that only authorized users can interact with the system.

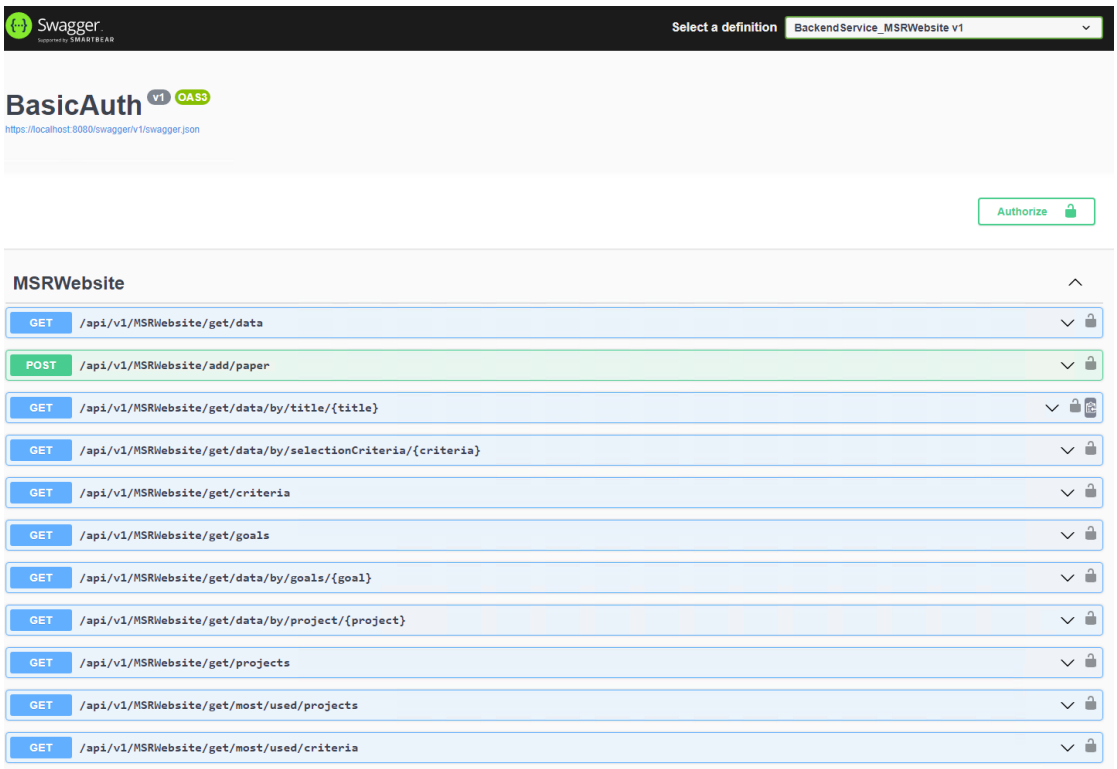

Figure 5.5: Web service illustration through swagger

```
public async Task<Dictionary<string, List<Dictionary<string, object>>>> SelectPapersByCriteriaAsync(string connection, string criteria)
{
    var con = new NpgsqlConnection(connection);
    con.Open();

    var cmd = new NpgsqlCommand();
    cmd.Connection = con;

    cmd.CommandText = "select Paper.* from Paper " +
        "inner join Paper_Criteria " +
        "on Paper.id = Paper_Criteria.paper_id " +
        "inner join consolidatedcriteria " +
        "on Paper_Criteria.criteria_id = consolidatedcriteria.id " +
        $"where selection_criteria = '{criteria}'";

    await using var res = cmd.ExecuteReader();

    var data = new List<Dictionary<string, object>>();
    while (await res.ReadAsync())
    {
        var dict = new Dictionary<string, object>();
        for (int i = 0; i < res.FieldCount; i++)
        {
            dict.Add(res.GetName(i), res[i]);
        }
        data.Add(dict);
    }
    res.Close();
    cmd.Dispose();
    con.Close();

    if (data.Count != 0)
    {
        var returnData = new Dictionary<string, List<Dictionary<string, object>>>();
        returnData["data"] = data;
        return returnData;
    }
    else
    {
        return new Dictionary<string, List<Dictionary<string, object>>>();
    }
}
```

Figure 5.6: GET method "*/get/data/by/project/{project}*"

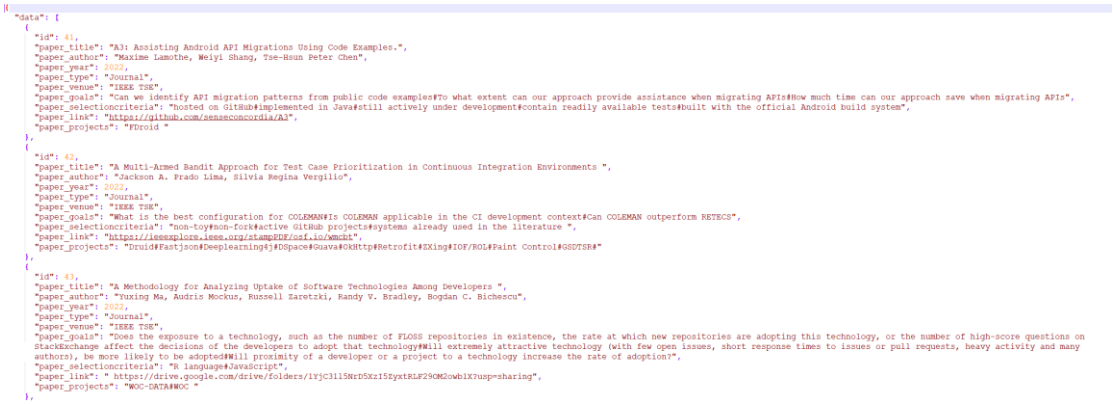

Figure 5.7: Dummy data

b) Database

We used PostgreSQL to store and manage the data of our system. PostgreSQL is an open-source relational database management system, known for its reliability, scalability and extensive feature set. Its support for complex data types, transactions, and concurrency control made it an appropriate choice for our project. We employed ACID

(Atomicity, Consistency, Isolation, Durability) properties to ensure data consistency and integrity. PostgreSQL's native support for transactions aids us in achieving this.

1.Database schematic and code execution program

The database schema was designed to represent our data and their relationships in the system. We created the tables as: "paper", "goals", "projects", "consolidatedcriteria", "paper_goals", "paper_project" and "paper_criteria". These tables represent the relationships between entities, whose relationships are many to many. The tables 'paper_goals', 'paper_project' and 'paper_criteria' act as link tables for linking the respective entities. This was done because as we can realize in research there can be more than one goal as well as open-source software used. Similarly, one open-source software can be used in more than one research. The schema of the database and the connection between the tables can be seen in figure 5.8.

To initialize the database, we developed a script to create the required tables and their associations. In addition, this script was also used to populate the tables with the data we collected and had stored in an Excel spreadsheet. Again, Npgsql was used for the creation and population of the tables. This approach allowed us to easily manage and import the original data into our database. The way this script (which is in .NET 6) works is:

- Connect to the database
- Create the tables (if they do not already exist)
- Read the excel file and create the data model
- Read data from the model, line by line, and insert the data into the "paper" table. At the end it returns the id.
- We split the projects, goals, research questions according to the rule we had defined with the hash symbol (#). If they are not already in the respective tables we insert them, otherwise we skip them. In any case we take back their id.
- We insert in the relation tables the pairs of ids that have been created.
- We proceed to the next line until all the data is inserted.

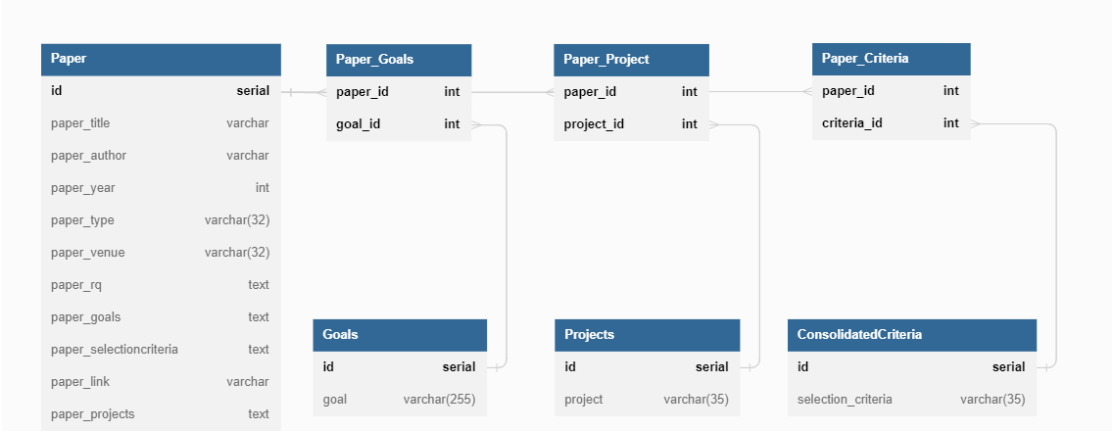

Figure 5.8: Database schema

c) Docker Containerization

To ensure deployment and easy portability of our backend, we embedded the entire application inside a Docker container. Docker provides a lightweight and isolated runtime environment, allowing us to package the backend, its dependencies, and necessary configuration in a single container. This approach simplifies deployment across different environments and ensures consistent behavior regardless of the underlying infrastructure. To deploy our backend, we needed to include 2 files, which will be explained below, *dockerfile* and *docker-compose.yml*.

Dockerfile

The Dockerfile serves as a script containing a set of instructions to build a Docker image for our application. It specifies the operating system, installs necessary software, copies project files, and sets up the environment for our .NET 6 application. The image built from this Dockerfile is a snapshot that contains everything our application needs to run.

The Dockerfile in Figure 5.9 employs a multi-stage build process to optimize the size and configuration of the resulting Docker image. The first stage, named **build-env**, uses the .NET 6 SDK image to compile the application in a temporary container. It copies the source code, restores NuGet packages, and publishes the application to an **out** directory. The second stage uses a lighter ASP.NET 6 runtime image to create the final container. It copies the compiled application and additional files from the **build-env**

stage to the final image, setting **BackendService_MSRWebsite.dll** as the entry point for container execution.

```
FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build-env
WORKDIR /App

# Copy everything
COPY . ./
# Restore as distinct layers
RUN dotnet restore
# Build and publish a release
RUN dotnet publish -c Release -o out

# Build runtime image
FROM mcr.microsoft.com/dotnet/aspnet:6.0
WORKDIR /App
COPY --from=build-env /App/out/ .
COPY --from=build-env /App/ ./copy
ENTRYPOINT ["dotnet", "BackendService_MSRWebsite.dll"]
```

Figure 5.9: Dockerfile

Docker-compose.yml

The **docker-compose.yml** file is used to define and manage multi-container Docker applications. It provides an easy way to configure and run all the services, including databases, queues, and the application itself, that make up a complex application. With a single command, **docker-compose up**, you can spin up the entire stack.

```
version: "3.8"
services:
  db:
    container_name: pg_container_1
    image: postgres:14.7
    restart: always
    environment:
      POSTGRES_USER: ****
      POSTGRES_PASSWORD: ******
      POSTGRES_DB: test_db
    volumes:
      - ./postgresql_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  backend:
    container_name: backend_container_1
    build: .
    restart: always
    ports:
      - "8080:80"
```

Figure 5.10: Docker-compose.yml

In Figure 5.10, the docker-compose.yml file defines two services: backend for our backend service and db for our PostgreSQL database. The backend service builds an image using the Dockerfile in the current directory and maps port 8000 on the host to port 80 on the container. The db service uses a prebuilt PostgreSQL image and sets some environment variables for database configuration.

By using both a *Dockerfile* and a *docker-compose.yml* file, we gain the ability to easily build, ship, and run our application in a variety of environments with a single command, ensuring consistency across all stages of development and deployment.

6 Conclusions and Future Research

In this thesis, we attempted to answer several key research questions related to the selection of open-source software in empirical studies. Our research was guided by three research questions:

- What are the goals of Mining Software Repositories (MSR) studies that use collections of projects mined from open repositories?
- What are the most common project selection criteria?
- What are the most used projects as subjects?

During our study, we thoroughly reviewed 1492 academic papers and drew on this extensive literature corpus to isolate key trends and patterns in the selection of open-source software for empirical research.

Our analysis revealed several interesting trends. First, our analysis shed light on the primary goals of MSR studies using open repositories. We found that these studies typically aim to find bugs, defects, and faults, predict vulnerabilities and identify security issues.

Secondly, we found that project selection criteria often revolve around project size, language, and popularity. These criteria are chosen primarily because of their perceived influence on the robustness of empirical findings.

Finally, our study identified the most used projects in empirical research. We found that projects such as Apache and Eclipse are frequently used as objects in empirical studies due to their large code bases, mature development processes and active contributor communities. This finding reflects the importance of project size, maturity, and community involvement in project selection criteria.

Based on our findings, we developed a web site that serves as a resource for future researchers. This website presents a summary of our research findings and provides a data-driven guide for selecting open-source software in empirical studies.

While this thesis provides an important step forward in understanding how open-source software is selected for empirical research, it also highlights several areas that need further investigation.

Future research could, for example, explore the potential biases introduced by over-reliance on a small set of popular projects. It might be worth exploring whether less

popular but nevertheless important works could offer new and unique insights. Moreover, future work could also delve into the impact of different project selection criteria on the results of empirical studies. As our understanding of software development evolves, it is likely that project selection criteria will also need to evolve.

In conclusion, we believe that our research has made a significant contribution to the field of empirical software engineering. We hope that our findings and the resulting web site can serve as valuable resources for future researchers and help guide more robust, representative, and insightful empirical studies.

Bibliography

B.1 Citations

Wohlin, Claes, Martin Höst and Kennet Henningsson, “Empirical Research Methods in Software Engineering.”, Experimental Software Engineering Network, 2003.

Runeson, P., Höst, M. Guidelines for conducting and reporting case study research in software engineering. Empir Software Eng 14, 131–164 ,2009.

Levy, S. (1984). Hackers: Heroes of the Computer Revolution. Doubleday. ISBN 0-385-19195-2.

Wayner, P. (2000). Free for All: How LINUX and the Free Software Movement Undercut the High-Tech Titans. HarperCollins.

Stallman, R. (1985). The GNU Manifesto.

Raymond, E.S. (1999). The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly Media. ISBN 1-56592-724-9.

Weber, S. (2004). The Success of Open Source. Harvard University Press. ISBN 0-674-01292-5.

Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. (2012). Social coding in GitHub: transparency and collaboration in an open software repository. In Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work.

Mário André de F. Farias, Renato Novais, Methanias Colaço Júnior, Luís Paulo da Silva Carvalho, Manoel Mendonça, and Rodrigo Oliveira Spínola, “A systematic mapping study on mining software repositories.”, In Proceedings of the 31st Annual ACM Symposium on Applied Computing (SAC '16). Association for Computing Machinery, New York, NY, USA, 1472–1479, 2016

D'Angelo R. Barros, Daniel & Horita, Flavio & Wiese, Igor & Silva, Kanan, “A Mining Software Repository Extended Cookbook: Lessons learned from a literature review.”, 2021

Kagdi, Huzefa & Collard, Michael & Maletic, Jonathan, “A survey and taxonomy of approaches for mining software repositories in the context of software evolution.”, Journal of Software Maintenance. 19. 77-131, 2007

Oyvind Hauge, Thomas Osterlie, Carl-Fredrik Sorensen, and Marinela Gerea, “An empirical study on selection of Open-Source Software - Preliminary results.”, In Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open-Source Software Research and Development (FLOSS '09). IEEE Computer Society, USA, 42–47, 2009

Chełkowski T, Jemielniak D, Macikowski K., “Free and Open-Source Software organizations: A large-scale analysis of code, comments, and commits frequency.” PLoS ONE 16(9), 2021

Igor Steinmacher, Marco Aurelio Graciotto Silva, Marco Aurelio Gerosa, David F. Redmiles, “A systematic literature review on the barriers faced by newcomers to open-source software projects.”, Information and Software Technology, 59, 67-85, 2015

Kalliamvakou, Eirini & Gousios, Georgios & Blincoe, Kelly & Singer, Leif & German, Daniel & Damian, Daniela, “An in-depth study of promises and perils of mining GitHub.”, Empirical Software Engineering 21, 2035-2071 (pp. 92-101), 2016

Apostolos Ampatzoglou, Stamatia Bibi, Paris Avgeriou, Marijn Verbeek, Alexander Chatzigeorgiou, “Identifying, categorizing and mitigating threats to validity in software engineering secondary studies.”, Information and Software Technology,106 (pp. 201-230), 2016

Petersen, Kai & Feldt, Robert & Mujtaba, Shahid & Mattsson, Michael, “Systematic Mapping Studies in Software Engineering.”, Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering. 17, 2008

Basili, R., Caldiera, G. and Rombach, H.D., “The Goal Question Metric Approach.”, Encyclopedia of Software Engineering, 1, 528-532, 1994

W.Eric Wong, Nikolaos Mittas, Elvira Maria Arvanitou, Yihao Li, “A bibliometric assessment of software engineering themes, scholars and institutions” (2013–2020), Journal of Systems and Software,Volume 180, 2021

Farhoodi, R., Garousi, V., Pfahl, D. and Sillito, J., "Development of Scientific Software: A Systematic Mapping, a Bibliometrics Study, and a Paper Repository", International Journal of Software Engineering and Knowledge Engineering, 23 (4), 2013

D. S. Cruzes and T. Dybå, "Research synthesis in software engineering: A tertiary study", Information and Software Technology, 53(5), pp. 440-455, 2011

P. Santos and G. H. Travassos, “Research Synthesis in Software Engineering”, Contemporary Empirical Methods in Software Engineering, pp. 443-474, 2020

B. Kitchenham, L. Madeyski, and P. Brereton, “Meta-analysis for families of experiments in software engineering: a systematic review and reproducibility and validity assessment”, Empirical Software Engineering, 25, pp. 353–401, 2020

B. Kitchenham, T. Dyba, and M. Jorgensen, “Evidence-based software engineering”, 26th International Conference on Software Engineering, Edinburgh, UK, 28 May 2004

H. van Vliet, "Software Engineering: Principles and Practice (3rd Edition)",Wiley & Sons, Chichester,England, 1993

R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, Dept. of Information and Computer Science, University of California, Irvine, 2000

B.2 Systematic mapping study references

[1] Aatira Anum Ahmad, Abdul Rafae Noor, Hashim Sharif, Usama Hameed, Shoaib Asif, Mubashir Anwar, Ashish Gehani, Fareed Zaffar, Junaid Haroon Siddiqui ,"Trimmer: An Automated System for Configuration-Based Software Debloating. ", IEEE Transactions on Software Engineering, 48 (09) pp, 3485-3505,2022

[2] Aayush Garg, Renzo Degiovanni, Matthieu Jimenez, Maxime Cordy, Mike Papadakis, Yves Le Traon ,"Learning from what we know: How to perform vulnerability prediction using noisy historical data" , Empirical Software Engineering , 27 (07) pp. 169 ,2022

[3] Abdul Razzaq, Anthony Ventresque, Rainer Koschke, Andrea De Lucia, Jim Buckley , "The Effect of Feature Characteristics on the Performance of Feature Location Techniques " , IEEE Transactions on Software Engineering, 48 (06) pp. 2066-2085 ,2022

[4] Abu Naser Masud ,"Efficient computation of minimal weak and strong control closure", Journal of Systems and Software, 184 (02) pp. 111140 ,2022

[5] Afsoon Afzal, Manish Motwani, Kathryn T. Stolee, Yuriy Brun, Claire Le Goues, "SOSRepair: Expressive Semantic Search for Real-World Program Repair", IEEE Transactions on Software Engineering, 47 (10) pp.2162-2181 ,2021

[6] Ahmed Zerouali, Tom Mens, Alexandre Decan, Coen De Roover,"On the impact of security vulnerabilities in the npm and RubyGems dependency networks.", Empirical Software Engineering , 27 (05) pp.107 ,2022

[7] Aleem Khalid Alvi, Mohammad Zulkernine, "A security pattern detection framework for building more secure software" , Journal of Systems and Software, 171 (01) pp. 110838 ,2021

[8] Alexandre Decan, Tom Mens , "What Do Package Dependencies Tell Us About Semantic Versioning?" , IEEE Transactions on Software Engineering, 47 (06) pp.1226-1240 ,2021

[9] Alexandre Perez, Rui Abreu, Arie van Deursen , "A Theoretical and Empirical Analysis of Program Spectra Diagnosability." , IEEE Transactions on Software Engineering, 47 (02) pp,412-431 ,2021

[10] Alexi Turcotte, Ellen Arteca, Ashish Mishra, Saba Alimadadi, Frank Tip, "Stubbifier: debloating dynamic server-side JavaScript applications" , Empirical Software Engineering , 27 (07) pp. 161 ,2022

[11] Aline Brito, Andre Hora, and Marco Tulio Valente, Characterizing refactoring graphs in Java and JavaScript projects, Empirical Software Engineering, 26(6), 2021

[12] Alireza Aghamohammadi, Seyed-Hassan Mirian-Hosseinabadi, and Sajad Jalali, "Statement frequency coverage: A code coverage criterion for assessing test suite effectiveness", Information and Software Technology, Volume 129, 2021

[13] Alvin Jian Jia Tan, Chun Yong Chong, Aldeida Aleti ,"E-SC4R: Explaining Software Clustering for Remodularisation", Journal of Systems and Software, 186 (04) pp. 111162 ,2022

[14] Amine Barrak, Ellis E. Eghan, Bram Adams, Foutse Khomh, "Why do builds fail? - A conceptual replication study" , Journal of Systems and Software, 177 (07) pp. 110939 ,2021

[15] Amjad AbuHassan, Mohammad Alshayeb, and Lahouari Ghouti, Prioritization of model smell refactoring using a covariance matrix-based adaptive evolution algorithm, Information and Software Technology, 146,2022

[16] Amjed Tahir, Kwabena E. Bennin, Xun Xiao, and Stephen G. MacDonell, Does class size matter? An in-depth assessment of the effect of class size in software defect prediction, Empirical Software Engineering, 26(5), 2021

[17] Amritanshu Agrawal, Xueqi Yang, Rishabh Agrawal, Rahul Yedida, Xipeng Shen, Tim Menzies ,"Simpler Hyperparameter Optimization for Software Analytics: Why, How, When? ", IEEE Transactions on Software Engineering, 48 (08) pp, 2939-2954 ,2022

[18] An Ran Chen, Tse-Hsun (Peter) Chen, and Shaowei Wang, Demystifying the challenges and benefits of analyzing user-reported logs in bug reports, Empirical Software Engineering, 26 (1), 2021

[19] An Ran Chen, Tse-Hsun Chen, Shaowei Wang ,"Pathidea: Improving Information Retrieval-Based Bug Localization by Re-Constructing Execution Paths Using Logs " , IEEE Transactions on Software Engineering, 48 (08) pp,2905-2919 ,2022

[20] André de S. Landi, Daniel San Martín, Bruno Marinho Santos, Warteruzannan Soyer Cunha, Rafael S. Durelli, Valter Vieira de Camargo,"Architectural conformance checking for KDM-represented systems." , 183 (01) pp. 111116 ,2022

[21] Andrea Arcuri and Juan P. Galeotti. 2021. Enhancing Search-based Testing with Testability Transformations for Existing APIs. ACM Trans. Softw. Eng. Methodol.

[22] Andrea Di Sorbo, and Sebastiano Panichella, Exposed! A case study on the vulnerability-proneness of Google Play Apps, Empirical Software Engineering, 26(4), 2021

[23] Andrea Romdhana, Alessio Merlo, Mariano Ceccato, and Paolo Tonella. 2022. Deep Reinforcement Learning for Black-box Testing of Android Apps. ACM Trans. Softw. Eng. Methodol.

[24] Andreas Dann, Henrik Plate, Ben Hermann, Serena Elisa Ponta, Eric Bodden, “Identifying Challenges for OSS Vulnerability Scanners - A Study & Test Suite “ , IEEE Transactions on Software Engineering, 48 (09) pp, 3613-3625 ,2022

[25] Ankur Tagra, Haoxiang Zhang, Gopi Krishnan Rajbahadur, Ahmed E. Hassan ,”Revisiting reopened bugs in open source software systems.”, Empirical Software Engineering , 27 (04) pp. 92 ,2022

[26] Annibale Panichella, "A Systematic Comparison of search-Based approaches for LDA hyperparameter tuning", Information and Software Technology, Volume 130, 2021

[27] Annibale Panichella, Sebastiano Panichella, Gordon Fraser, Anand Ashok Sawant, Vincent J. Hellendoorn ,”Test smells 20 years later: detectability, validity, and reliability.”,Empirical Software Engineering , 27 (07) pp. 170 ,2022

[28] Antonia Bertolino, Breno Miranda, Roberto Pietrantuono, Stefano Russo ,”Adaptive Test Case Allocation, Selection and Generation Using Coverage Spectrum and Operational Profile.”, IEEE Transactions on Software Engineering, 47 (05) pp. 881-898 ,2021

[29] Antonios Gkortzis, Daniel Feitosa, Diomidis Spinellis , “Software reuse cuts both ways: An empirical analysis of its relationship with security vulnerabilities” , Journal of Systems and Software, 172 (02) pp. 110653 ,2021

[30] Aoi Takahashi, Natthawute Sae-Lim, Shinpei Hayashi, Motoshi Saeki, “An extensive study on smell-aware bug localization” , Journal of Systems and Software, 178 (08) pp. 110986 ,2021

[31] Arianna Blasi, Alessandra Gorla, Michael D. Ernst, Mauro Pezzè, Antonio Carzaniga,”MeMo: Automatically identifying metamorphic relations in Javadoc comments for test automation” ,Journal of Systems and Software, 181 (11) pp. 111041 ,2021

[32] Arianna Blasi, Nataliia Stulova, Alessandra Gorla, Oscar Nierstrasz,"RepliComment: Identifying clones in code comments" ,Journal of Systems and Software, 182 (12) pp. 111069 ,2021

[33] Arif Nurwidyantoro, Mojtaba Shahin, Michel R.V. Chaudron, Waqar Hussain, Rifat Shams, Harsha Perera, Gillian Oliver, and Jon Whittle, Human values in software development artefacts: A case study on issue discussions in three Android applications, Information and Software Technology,141,2022

[34] Arthur Kamienski, and Cor-Paul Bezemer, An empirical study of Q&A websites for game developers, Empirical Software Engineering, 26(6), 2021

[35] Arthur Vitui, and Tse-Hsun (Peter) Chen, MLASP: Machine learning assisted capacity planning, Empirical Software Engineering, 26(5), 2021

[36] Baicai Sun, Dunwei Gong, Tian Tian, Xiangjuan Yao , "Integrating an Ensemble Surrogate Model's Estimation into Test Data Generation. " , IEEE Transactions on Software Engineering, 48 (04) pp.1336-1350 ,2022

[37] Bailey Vandehei, Daniel Alencar Da Costa, and Davide Falessi. 2021. Leveraging the Defects Life Cycle to Label Affected Versions and Defective Classes. ACM Trans. Softw. Eng. Methodol.

[38] Béla Vancsics, Ferenc Horváth, Attila Szatmári, Árpád Beszédes" ,Fault localization using function call frequencies." , Journal of Systems and Software, 193 (11) pp. 111429 ,2022

[39] Benjamin Loriot, Fernanda Madeiral, Martin Monperrus, "Styler: learning formatting conventions to repair Checkstyle violations.", Empirical Software Engineering , 27 (06) pp. 149 ,2022

[40] Beyza Eken, Ayse Tosun, "Investigating the performance of personalized models for software defect prediction" , Journal of Systems and Software, 181 (11) pp. 111038 ,2021

[41] Biruk Asmare Muse, Csaba Nagy, Anthony Cleve, Foutse Khomh, Giuliano Antoniol ,"FIXME: synchronize with database! An empirical study of data access self-admitted technical debt." , Empirical Software Engineering , 27 (06) pp. 130 ,2022

[42] Bo Lin, Shangwen Wang, Ming Wen, and Xiaoguang Mao. 2022. Context-Aware Code Change Embedding for Better Patch Correctness Assessment. ACM Trans. Softw. Eng. Methodol.

[43] Bodin Chinthanet, Raula Gaikovina Kula, Shane McIntosh, Takashi Ishio, Akinori Ihara, and Kenichi Matsumoto, Lags in the release, adoption, and propagation of npm vulnerability fixes, Empirical Software Engineering, 26(3), 2021

[44] Boqin Qin, Tengfei Tu, Ziheng Liu, Tingting Yu, Linhai Song , "Algorithmic Profiling for Real-World Complexity Problems " , IEEE Transactions on Software Engineering, 48 (07) pp, 2680-2694 ,2022

[45] Brent van Bladel, Serge Demeyer ,"A comparative study of test code clones and production code clones" , Journal of Systems and Software, 176 (06) pp. 110940 ,2021

[46] Camilo Escobar-Velásquez, Alejandro Mazuera-Rozo, Claudia Bedoya, Michael Osorio-Riaño, Mario Linares-Vásquez, Gabriele Bavota ,"Studying eventual connectivity issues in Android apps" , Empirical Software Engineering , 27 (01) pp. 22 ,2022

[47] Camilo Escobar-Velásquez, Mario Linares-Vásquez, Gabriele Bavota, Michele Tufano, Kevin Moran, Massimiliano Di Penta, Christopher Vendome, Carlos Bernal-Cárdenas, Denys Poshyvanyk, "Enabling Mutant Generation for Open- and Closed-Source Android Apps. ", IEEE Transactions on Software Engineering, 48 (02) pp.186-208 ,2022

[48] Catarina Costa, Jair Figueiredo, João Felipe Pimentel, Anita Sarma, Leonardo Murta ,"Recommending Participants for Collaborative Merge Sessions" , IEEE Transactions on Software Engineering, 47 (06) pp. 1198-1210 ,2021

[49] César Soto-Valero, Nicolas Harrand, Martin Monperrus, and Benoit Baudry, A comprehensive study of bloated dependencies in the Maven ecosystem, Empirical Software Engineering, 26(3), 2021

[50] Cezar Sas, Andrea Capiluppi ,"Antipatterns in software classification taxonomies" , Journal of Systems and Software, 190 (08) pp. 111343 ,2022

[51] Chaima Abid, Marouane Kessentini, Vahid Alizadeh, Mouna Dhaouadi, Rick Kazman , "How Does Refactoring Impact Security When Improving Quality? A Security-Aware Refactoring Approach " , IEEE Transactions on Software Engineering, 48 (03) pp. 864-878 ,2022

[52] Chang-Ai Sun, Baoli Liu, An Fu, Yiqiang Liu, Huai Liu ,"Path-directed source test case generation and prioritization in metamorphic testing" , Journal of Systems and Software, 183 (01) pp. 111091 ,2022

[53] Changqing Wei, Xiangjuan Yao, Dunwei Gong,and Huai Liu, Spectral clustering based mutant reduction for mutation testing, Information and Software Technology, Volume 132, 2021

[54] Chao Liu, Xin Xia, David Lo, Zhiwe Liu, Ahmed E. Hassan, and Shanping Li. 2021. CodeMatcher: Searching Code Based on Sequential Semantics of Important Query Words. ACM Trans. Softw. Eng. Methodol.

[55] Chao Ni, Xin Xia, David Lo, Xiang Chen, Qing Gu , “Revisiting Supervised and Unsupervised Methods for Effort-Aware Cross-Project Defect Prediction “, IEEE Transactions on Software Engineering, 48 (03) pp. 786-802,2022

[56] Chi Chen, Xin Peng, Bihuan Chen, Jun Sun, Zhenchang Xing, Xin Wang, Wenyun Zhao, “"More Than Deep Learning": post-processing for API sequence recommendation.” , Empirical Software Engineering , 27 (01) pp. 15 ,2022

[57] Christoph Laaber, Harald C. Gall, and Philipp Leitner, Applying test case prioritization to software microbenchmarks, Empirical Software Engineering, 26(6), 2021

[58] Christophe Rezk, Yasutaka Kamei, Shane McIntosh,”The Ghost Commit Problem When Identifying Fix-Inducing Changes: An Empirical Study of Apache Projects “, IEEE Transactions on Software Engineering, 48 (09) pp, 3297-3309,2022

[59] Chunying Zhou, Peng He, Cheng Zeng, and Ju Ma, Software defect prediction with semantic and structural information of codes based on Graph Neural Networks, Information and Software Technology, 152,2022

[60] Cuiyun Gao, Jichuan Zeng, Federica Sarro, David Lo, Irwin King, Michael and R. Lyu, Do users care about ad’s performance costs? Exploring the effects of the performance costs of in-app ads on user experience, Information and Software Technology, Volume 132, 2021

[61] Daming Zou, Jingjing Liang, Yingfei Xiong, Michael D. Ernst, Lu Zhang , “An Empirical Study of Fault Localization Families and Their Combinations” , IEEE Transactions on Software Engineering, 47 (02) pp, 332-347 ,2021

[62] Danilo Dominguez Perez, Wei Le ,”Specifying Callback Control Flow of Mobile Apps Using Finite Automata.” , IEEE Transactions on Software Engineering, 47 (02) pp, 379-392 ,2021

[63] Danilo Silva, João Paulo da Silva, Gustavo Jansen de Souza Santos, Ricardo Terra, Marco Túlio Valente ,"RefDiff 2.0: A Multi-Language Refactoring Detection Tool." , IEEE Transactions on Software Engineering, 47 (12) pp. 2786-2802 ,2021

[64] Daoyuan Wu, Debin Gao, and David Lo, Scalable online vetting of Android apps for measuring declared SDK versions and their consistency with API calls, Empirical Software Engineering, 26(1), 2021

[65] David Binkley, Leon Moonen, and Sibren Isaacman, Featherweight assisted vulnerability discovery, Information and Software Technology, 146,2022

[66] Davide Falessi, Aalok Ahluwalia, and Massimiliano DI Penta. 2021. The Impact of Dormant Defects on Defect Prediction: A Study of 19 Apache Projects. ACM Trans. Softw. Eng. Methodol.

[67] Dayi Lin, Chakkrit Tantithamthavorn, Ahmed E. Hassan ,"The Impact of Data Merging on the Interpretation of Cross-Project Just-In-Time Defect Models " , IEEE Transactions on Software Engineering, 48 (08) pp, 2969-2986,2022

[68] Debolina Ghosh,and Jagannath Singh, Spectrum-based multi-fault localization using Chaotic Genetic Algorithm, Information and Software Technology, Volume 133, 2021

[69] Devika Sondhi, Mayank Jobanputra, Divya Rani, Salil Purandare, Sakshi Sharma, Rahul Purandare ,"Mining Similar Methods for Test Adaptation. " , IEEE Transactions on Software Engineering, 48 (07) pp. 2262-2276 ,2022

[70] Dhia Elhaq Rzig, Foyzul Hassan, and Marouane Kessentini, An empirical study on ML DevOps adoption trends, efforts, and benefits analysis, Information and Software Technology,152,2022

[71] Diego Costa, Cor-Paul Bezemer, Philipp Leitner, Artur Andrzejak ,"What's Wrong with My Benchmark Results? Studying Bad Practices in JMH Benchmarks." , IEEE Transactions on Software Engineering, 47 (07) pp. 1452-1467 ,2021

[72] Diego Elias Costa, Suhaib Mujahid, Rabe Abdalkareem, Emad Shihab ,"Breaking Type Safety in Go: An Empirical Study on the Usage of the unsafe Package " , IEEE Transactions on Software Engineering, 48 (07) pp, 2277-2294 ,2022

[73] Dilini Rajapaksha, Chakkrit Tantithamthavorn, Jirayus Jiarpakdee, Christoph Bergmeir, John Grundy, Wray L. Buntine,"SQAPlanner: Generating Data-Informed Software Quality Improvement Plans. " , IEEE Transactions on Software Engineering, 48 (08) pp,2814-2835,2022

[74] Diomidis Spinellis, Paris Avgeriou, “Evolution of the Unix System Architecture: An Exploratory Case Study.”, IEEE Transactions on Software Engineering, 47 (06) pp. 1134-1163 ,2021

[75] Dipesh Pradhan, Shuai Wang, Shaukat Ali, Tao Yue, Marius Liaaen ,”CBGA-ES+: A Cluster-Based Genetic Algorithm with Non-Dominated Elitist Selection for Supporting Multi-Objective Test Optimization”, IEEE Transactions on Software Engineering, 47 (01) pp, 86-107 ,2021

[76] Dong Wang, Tao Xiao, Patanamon Thongtanunam, Raula Gaikovina Kula, and Kenichi Matsumoto, Understanding shared links and their intentions to meet information needs in modern code review, Empirical Software Engineering, 26(5), 2021

[77] Dong Jae Kim, Tse-Hsun (Peter) Chen, and Jinqiu Yang, The secret life of test smells - an empirical study on test smell evolution and maintenance, Empirical Software Engineering, 26(5), 2021

[78] Dong Wang, Raula Gaikovina Kula, Takashi Ishio, and Kenichi Matsumoto, Automatic patch linkage detection in code review using textual content and file location features, Information and Software Technology,139,2021

[79] Donghoon Jeon, Minseok Jeon,and Hakjoo Oh, A practical algorithm for learning disjunctive abstraction heuristics in static program analysis, Information and Software Technology, Volume 135, 2021

[80] Donghwan Shin, Domenico Bianculli, Lionel C. Briand , “PRINS: scalable model inference for component-based system logs”, Empirical Software Engineering , 27 (04) pp. 87 ,2022

[81] Dongliang Mu, Yunlan Du, Jianhao Xu, Jun Xu, Xinyu Xing, Bing Mao, Peng Liu ,”POMP++: Facilitating Postmortem Program Diagnosis with Value-Set Analysis.” , IEEE Transactions on Software Engineering, 47 (09) pp. 1929-1942,2021

[82] Eliane Maria De Bortoli Fávero, Dalcimar Casanova, and Andrey Ricardo Pimentel, SE3M: A model for software effort estimation using pre-trained embedding models, Information and Software Technology,147,2022

[83] Elijah Zolduoarrati, and Sherlock A. Licorish, On the value of encouraging gender tolerance and inclusiveness in software engineering communities, Information and Software Technology,139,2021

[84] Eman Abdullah AlOmar, Mohamed Wiem Mkaouer, Ali Ouni ,"Toward the automatic classification of Self-Affirmed Refactoring" , Journal of Systems and Software, 171 (01) pp. 110821 ,2021

[85] Emre Doğan, and Eray Tüzün, Towards a taxonomy of code review smells, Information and Software Technology,142,2022

[86] Emre Sülün, Eray Tüzün,and Uğur Doğrusöz, "RSTrace+: Reviewer suggestion using software artifact traceability graphs", Information and Software Technology, Volume 130, 2021

[87] Enrico Fregnan, Fernando Petrulio, Alberto Bacchelli ,"The evolution of the code during review: an investigation on review changes" , Empirical Software Engineering , 27 (07) pp. 177 ,2022

[88] Ezekiel O. Soremekun, Esteban Pavese, Nikolas Havrikov, Lars Grunske, Andreas Zeller , "Inputs From Hell" , IEEE Transactions on Software Engineering, 48 (04) pp. 1138-1153,2022

[89] Ezekiel Soremekun, Lukas Kirschner, Marcel Böhme, and Andreas Zeller, Locating faults with program slicing: an empirical analysis, Empirical Software Engineering, 26(3), 2021

[90] Fabiano Pecorelli, Fabio Palomba, and Andrea De Lucia, The Relation of Test-Related Factors to Software Quality: A Case Study on Apache Systems,Empirical Software Engineering,26(2), 2021

[91] Fabiano Pecorelli, Savanna Lujan, Valentina Lenarduzzi, Fabio Palomba, Andrea De Lucia, "On the adequacy of static analysis warnings with respect to code smell prediction." , Empirical Software Engineering , 27 (03) pp. 64 ,2022

[92] Fábio de Almeida Farzat, Márcio de Oliveira Barros, Guilherme H. Travassos ."Evolving JavaScript Code to Reduce Load Time" , IEEE Transactions on Software Engineering, 47 (08) pp. 1544-1558 ,2021

[93] Fabio Palomba, Damian Andrew Tamburri ,"Predicting the emergence of community smells using socio-technical metrics: A machine-learning approach" ,Journal of Systems and Software, 171 (01) pp. 110847 ,2021

[94] Fabio Palomba, Damian Andrew Tamburri, Francesca Arcelli Fontana, Rocco Oliveto, Andy Zaidman, Alexander Serebrenik, "Beyond Technical Aspects: How Do Community Smells Influence the Intensity of Code Smells?", IEEE Transactions on Software Engineering, 47 (01) pp, 108-129 ,2021

[95] Fabrizio Pastore, Daniela Micucci, Michell Guzmán, Leonardo Mariani,"TkT: Automatic Inference of Timed and Extended Pushdown Automata. ",IEEE Transactions on Software Engineering, 48 (02) pp. 617-636 ,2022

[96] Fanlong Zhang,and Siau-cheng Khoo, An empirical study on clone consistency prediction based on machine learning, Information and Software Technology,136,2021

[97] Faria Huq, Masum Hasan, Md Mahim Anjum Haque, Sazan Mahbub, Anindya Iqbal, and Toufique Ahmed, Review4Repair: Code review aided automatic program repairing, Information and Software Technology, 143,2022

[98] Ferenc Horváth, Árpád Beszédes, Béla Vancsics, Gergõ Balogh, László Vidács, Tibor Gyimóthy , "Using contextual knowledge in interactive fault localization" , Empirical Software Engineering , 27 (06) pp.150 ,2022

[99] Filipe R. Cogo, Gustavo A. Oliva, Cor-Paul Bezemer, and Ahmed E. Hassan, An empirical study of same-day releases of popular packages in the npm ecosystem, Empirical Software Engineering, 26(5), 2021

[100] Fiorella Zampetti, Saghan Mudbhari, Venera Arnaoudova, Massimiliano Di Penta, Sebastiano Panichella, Giuliano Antoniol, "Using code reviews to automatically configure static analysis tools.", Empirical Software Engineering , 27 (01) pp. 28 ,2022

[101] Flavio Corradini, Fabrizio Fornari, Andrea Polini, Barbara Re, Francesco Tiezzi, Andrea Vandin, "A formal approach for the analysis of BPMN collaboration models.", Journal of Systems and Software, 180 (10) pp. 111007 ,2021

[102] Francesco Lomio, Emanuele Iannone, Andrea De Lucia, Fabio Palomba, Valentina Lenarduzzi, "Just-in-time software vulnerability detection: Are we there yet", Journal of Systems and Software, 188 (06) pp. 111283 ,2022

[103] Francesco Lomio, Sergio Moreschini, Valentina Lenarduzzi ,"A machine and deep learning analysis among SonarQube rules, product, and process metrics for fault prediction." , Empirical Software Engineering , 27 (07) pp. 189 ,2022

[104] Francis Palma, Tobias Olsson, Anna Wingkvist, Javier Gonzalez-Huerta , "Assessing the linguistic quality of REST APIs for IoT applications" ,Journal of Systems and Software, 191 (09) pp. 111369 ,2022

[105] Francisco Handrick da Costa, Ismael Medeiros, Thales Menezes, João Victor da Silva, Ingrid Lorraine da Silva, Rodrigo Bonifácio, Krishna Narasimhan, Márcio Ribeiro, "Exploring the use of static and dynamic analysis to improve the performance of the

mining sandbox approach for android malware identification" , Journal of Systems and Software, 183 (01) pp. 111092 ,2022

[106] Frolin S. Ocariza Jr. ,"On the Effectiveness of Bisection in Performance Regression Localization.", Empirical Software Engineering , 27 (04) pp. 95 ,2022

[107] Gabriela Karoline Michelon, David Obermann, Wesley K. G. Assunção, Lukas Linsbauer, Paul Grünbacher, Stefan Fischer, Roberto E. Lopez-Herrejon, Alexander Egyed, "Evolving software system families in space and time with feature revisions", Empirical Software Engineering , 27 (05) pp. 112 ,2022

[108] Gede Artha Azriadi Prana, Abhishek Sharma, Lwin Khin AU - Shar, Darius Foo, Andrew E. Santosa, AsankhayaSharma, and David Lo, Out of sight, out of mind? How vulnerable dependencies affect open-source projects, Empirical Software Engineering, 26(4), 2021

[109] George Digkas, Alexander Chatzigeorgiou, Apostolos Ampatzoglou, Paris Avgeriou , "Can Clean New Code Reduce Technical Debt Density? " , IEEE Transactions on Software Engineering, 48 (05) pp. 1705-1721 ,2022

[110] Gerardo Canfora, Andrea Di Sorbo, Sara Forootani, Matias Martinez, and Corrado A. Visaggio, Patchworking: Exploring the code changes induced by vulnerability fixing activities, Information and Software Technology, 142,2022

[111] Giovani Guizzo, Federica Sarro, Jens Krinke, Silvia R. Vergilio ,"Sentinel: A Hyper-Heuristic for the Generation of Mutant Reduction Strategies " ,IEEE Transactions on Software Engineering, 48 (03) pp. 803-818 ,2022

[112] Giovanni Grano, Christoph Laaber, Annibale Panichella, Sebastiano Panichella, "Testing with Fewer Resources: An Adaptive Approach to Performance-Aware Test Case Generation", IEEE Transactions on Software Engineering, 47 (11) pp. 2332-2347,2021

[113] Giovanni Grano, Fabio Palomba, Harald C. Gall , "Lightweight Assessment of Test-Case Effectiveness Using Source-Code-Quality Indicators." ,IEEE Transactions on Software Engineering, 47 (04) pp. 758-774 ,2021

[114] Giovanni Liva, Muhammad Taimoor Khan, Martin Pinzger, Francesco Spegni, Luca Spalazzi ,"Automatic Repair of Timestamp Comparisons.", IEEE Transactions on Software Engineering, 47 (11) pp.2369-2381 ,2021

[115] Gopi Krishnan Rajbahadur, Shaowei Wang, Gustavo Ansaldi Oliva, Yasutaka Kamei, Ahmed E. Hassan, "The Impact of Feature Importance Methods on the

Interpretation of Defect Classifiers. “ , IEEE Transactions on Software Engineering, 48 (07) pp. 2245-2261 ,2022

[116] Goran Piskachev, Johannes Späth, Ingo Budde, Eric Bodden ,”Fluently specifying taint-flow queries with fluentTQL”, Empirical Software Engineering , 27 (05) pp. 104 ,2022

[117] Guanhua Wang, Sudipta Chattopadhyay, Ivan Gotovchits, Tulika Mitra, Abhik Roychoudhury , “oo7: Low-Overhead Defense Against Spectre Attacks via Program Analysis.” , IEEE Transactions on Software Engineering, 47 (11) pp. 2504-2519 ,2021

[118] Guoli Cheng, Shi Ying, Bingming Wang, “Tuning configuration of apache spark on public clouds by combining multi-objective optimization and performance prediction model “, Journal of Systems and Software, 180 (10) pp. 111028 ,2021

[119] Gustav Bergstr, Arif Nurwidyantoro, Michel R. V. Chaudronöm, Fadhl Hujainah, Truong Ho-Quang, Rodi Jolak, Satrio Adi Rukmono “,Evaluating the layout quality of UML class diagrams using machine learning.”, Journal of Systems and Software, 192 (10) pp. 111413 ,2022

[120] Ha Thanh Le, Lwin Khin Shar, Domenico Bianculli, Lionel Claude Briand, Cu Duy Nguyen ,”Automated reverse engineering of role-based access control policies of web applications”, Journal of Systems and Software, 184 (02) pp. 111109 ,2022

[121] Hadhemi Jebnoun, Md. Saidur Rahman, Foutse Khomh, Biruk Asmare Muse, “Clones in deep learning code: what, where, and why?”, Empirical Software Engineering , 27 (04) pp. 84 ,2022

[122] Hadi Jahanshahi, Mucahit Cevik, José Navas-Sú, Ayse Basar, Antonio González Torres ,”Wayback Machine: A tool to capture the evolutionary behavior of the bug reports and their triage process in open-source software systems” ,Journal of Systems and Software, 189 (07) pp. 111308 ,2022

[123] Haijun Wang, Yun Lin, Zijiang Yang, Jun Sun, Yang Liu, Jin Song Dong, Qinghua Zheng, Ting Liu ,”Explaining Regressions via Alignment Slicing and Mending”. IEEE Transactions on Software Engineering, 47 (11) pp. 2421-2437 ,2021

[124] Hamid Bagheri, Jianghao Wang, Jarod Aerts, Negar Ghorbani, and Sam Malek, Flair: efficient analysis of Android inter-component vulnerabilities in response to incremental changes, Empirical Software Engineering, 26(3), 2021

[125] Hamidreza Ahmadi, Mehrdad Ashtiani, Mohammad Abdollahi Azgomi, and Raana Saheb-Nassagh, A DQN-based agent for automatic software refactoring, Information and Software Technology,147,2022

[126] Hamzeh Eyal Salman, Feature-based insight for forks in social coding platforms, Information and Software Technology,140,2021

[127] Hanefi Mercan, Atakan Aytar, Giray Coskun, Dilara Müstecep, Gülsüm Uzer, Cemal Yilmaz ,"CIT-daily: A combinatorial interaction testing-based daily build process",Journal of Systems and Software, 190 (08) pp. 111353 ,2022

[128] Hanyu Pei, Beibei Yin, Min Xie,and Kai-Yuan Cai, Dynamic random testing with test case clustering and distance-based parameter adjustment, Information and Software Technology, Volume 131, 2021

[129] Hao Zhong, Xiaoyin Wang, Hong Mei ,"Inferring Bug Signatures to Detect Real Bugs" ,IEEE Transactions on Software Engineering, 48 (02) pp. 571-584 ,2022

[130] Haonan Tong, Bin Liu, Shihai Wang, "Kernel Spectral Embedding Transfer Ensemble for Heterogeneous Defect Prediction." , IEEE Transactions on Software Engineering, 47 (09) pp. 1886-1906 ,2021

[131] Haonan Tong, Wei Lu, Weiwei Xing, Bin Liu, and Shihai Wang, SHSE: A subspace hybrid sampling ensemble method for software defect number prediction, Information and Software Technology,142,2022

[132] Haonan Zhang, Yiming Tang, Maxime Lamothe, Heng Li, Weiyi Shang, "Studying logging practice in test code", Empirical Software Engineering , 27 (04) pp. 83 ,2022

[133] Haowen Chen, Xiao-Yuan Jing, Yuming Zhou, Bing Li, and Baowen Xu, Aligned metric representation based balanced multiset ensemble learning for heterogeneous defect prediction, Information and Software Technology,147,2022

[134] Haowen Chen, Xiao-Yuan Jing, Zhiqiang Li, Di Wu, Yi Peng, Zhiguo Huang , "An Empirical Study on Heterogeneous Defect Prediction Approaches" , IEEE Transactions on Software Engineering, 47 (12) pp. 2803-2822 ,2021

[135] Hayyan Hasan, Behrouz Tork Ladani, and Bahman Zamani, MEGDroid: A model-driven event generation framework for dynamic android malware analysis, Information and Software Technology, Volume 135, 2021

[136] He Ye, Jian Gu, Matias Martinez, Thomas Durieux, Martin Monperru, “Automated Classification of Overfitting Patches With Statically Extracted Code Features “ , IEEE Transactions on Software Engineering, 48 (08) pp, 2920-2938 ,2022

[137] He Ye, Matias Martinez, and Martin Monperrus, Automated patch assessment for program repair at scale, Empirical Software Engineering,26(2), 2021

[138] Héctor D. Menéndez, David Clark ,”Hashing Fuzzing: Introducing Input Diversity to Improve Crash Detection. “, IEEE Transactions on Software Engineering, 48 (09) pp, 3540-3553 ,2022

[139] Héctor D. Menéndez, Gunel Jahangirova, Federica Sarro, Paolo Tonella, and David Clark. 2021. Diversifying Focused Testing for Unit Testing. ACM Trans. Softw. Eng. Methodol.

[140] Hetong Dai, Heng Li, Che-Shao Chen, Weiyi Shang, Tse-Hsun Chen, “Logram: Efficient Log Parsing Using nn-Gram Dictionaries.” , IEEE Transactions on Software Engineering, 48 (03) pp. 879-892 ,2022

[141] Hoa Khanh Dam, Truyen Tran, Trang Pham, Shien Wee Ng, John Grundy, Aditya Ghose ,”Automatic Feature Learning for Predicting Vulnerable Software Components.”, IEEE Transactions on Software Engineering, 47 (01) pp, 67-85 ,2021

[142] Hong Jin Kang, David Lo , “Active Learning of Discriminative Subgraph Patterns for API Misuse Detection. “ IEEE Transactions on Software Engineering, 48 (08) pp, 2761-2783 ,2022

[143] Huangzhao Zhang, Zhiyi Fu, Ge Li, Lei Ma, Zhehao Zhao, Hua’an Yang, Yizhe Sun, Yang Liu, and Zhi Jin. 2022. Towards Robustness of Deep Program Processing Models—Detection, Estimation, and Enhancement. ACM Trans. Softw. Eng. Methodol.

[144] Hui Liu, Jiahao Jin, Zhifeng Xu, Yanzhen Zou, Yifan Bu, Lu Zhang ,”Deep Learning Based Code Smell Detection”, IEEE Transactions on Software Engineering, 47 (09) pp. 1811-1837 ,2021

[145] Hui Liu, Mingzhu Shen, Jiaqi Zhu, Nan Niu, Ge Li, Lu Zhang ,”Deep Learning Based Program Generation From Requirements Text: Are We There Yet? “, IEEE Transactions on Software Engineering, 48 (04) pp. 1268-1289 ,2022

[146] Huy Tu, Tim Menzies, “DebtFree: minimizing labeling cost in self-admitted technical debt identification using semi-supervised learning”, Empirical Software Engineering , 27 (04) pp. 80 ,2022

[147] Huy Tu, Zhe Yu, Tim Menzies , “Better Data Labelling With EMBLEM (and how that Impacts Defect Prediction). “ , IEEE Transactions on Software Engineering, 48 (02) pp.278-294 ,2022

[148] Ilaria Pigazzini, Francesca Arcelli Fontana, Bartosz Walter, “A study on correlations between architectural smells and design patterns”, Journal of Systems and Software, 178 (08) pp. 110984 ,2021

[149] Information and Software Technology,151,2022

[150] Islem Saidani, Ali Ouni, Mohamed Wiem Mkaouer, and Fabio Palomba, On the impact of Continuous Integration on refactoring practice: An exploratory study on TravisTorrent, Information and Software Technology,138,2021

[151] Ivan Pashchenko, Henrik Plate, Serena Elisa Ponta, Antonino Sabetta, Fabio Massacci, “Vuln4Real: A Methodology for Counting Actually Vulnerable Dependencies “ , IEEE Transactions on Software Engineering, 48 (05) pp. 1592-1609 ,2022

[152] Jackson A. Prado Lima, Silvia Regina Vergilio ,”A Multi-Armed Bandit Approach for Test Case Prioritization in Continuous Integration Environments “ , IEEE Transactions on Software Engineering, 48 (02) pp. 453-465 ,2022

[153] Jahanshahi, Hadi, and Mucahit Cevik. S-DABT: Schedule and Dependency-Aware Bug Triage in Open-Source Bug Tracking Systems,Information and Software Technology ,151, 2022

[154] James Callan, Oliver Krauss, Justyna Petke, Federica Sarro ,”How do Android developers improve non-functional properties of software?” ,Empirical Software Engineering , 27 (05) pp. 113 ,2022

[155] Javaria Imtiaz, Muhammad Zohaib Iqbal, Muhammad Uzair Khan , “An automated model-based approach to repair test suites of evolving web applications” , Journal of Systems and Software, 171 (01) pp. 110841 ,2021

[156] Jeongju Sohn, Shin Yoo, “Empirical Evaluation of Fault Localisation Using Code and Change Metrics.”, IEEE Transactions on Software Engineering, 47 (08) pp. 1605-1625 ,2021

[157] Jhih-Sin Lin, Chin-Yu Huang, Chih-Chiang Fang , “Analysis and assessment of software reliability modeling with preemptive priority queueing policy”, Journal of Systems and Software, 187 (05) pp. 111249 ,2022

[158] Jhonny Mertz, Ingrid Nunes, “Tigris: A DSL and framework for monitoring software systems at runtime.” , Journal of Systems and Software, 177 (07) pp. 110963 ,2021

[159] Jiakun Liu, Qiao Huang, Xin Xia, Emad Shihab, David Lo,and Shanping Li.An exploratory study on the introduction and removal of different types of technical debt in deep learning frameworks,Empirical Software Engineering ,26(2),2021

[160] Jiaming Ye, Mingliang Ma, Yun Lin, Lei Ma, Yinxing Xue, Jianjun Zhao , “Vulpedia: Detecting vulnerable ethereum smart contracts via abstracted vulnerability signatures.”, Journal of Systems and Software, 192 (10) pp. 111410 ,2022

[161] Jian Gao, Yu Jiang, Zhe Liu, Xin Yang, Cong Wang, Xun Jiao, Zijiang Yang, Jiaguang Sun ,”Semantic Learning and Emulation Based Cross-Platform Binary Vulnerability Seeker.”, IEEE Transactions on Software Engineering, 47 (11) pp. 2575-2589 ,2021

[162] Jianyi Zhou, Junjie Chen, and Dan Hao. 2021. Parallel Test Prioritization. ACM Trans. Softw. Eng. Methodol.

[163] Jiaojiao Bai, Jingdong Jia, and Luiz Fernando Capretz, A three-stage transfer learning framework for multi-source cross-project software defect prediction, Information and Software Technology,150,2022

[164] Jiaojiao Yu, Kunsong Zhao, Jin Liu, Xiao Liu, Zhou Xu, Xin Wang ,”Exploiting gated graph neural network for detecting and explaining self-admitted technical debts”, Journal of Systems and Software, 187 (05) pp. 111219 ,2022

[165] Jie Tan, Daniel Feitosa, and Paris Avgeriou, Does it matter who pays back Technical Debt? An empirical study of self-fixed TD, Information and Software Technology, 143,2022

[166] Jinfu Chen, Weiyi Shang, Emad Shihab , “PerfJIT: Test-Level Just-in-Time Prediction for Performance Regression Introducing Commits “ , IEEE Transactions on Software Engineering, 48 (05) pp.1529-1544 ,2022

[167] Jing Jiang, Qiudi Wu, Jin Cao, Xin Xia, and Li Zhang, "Recommending tags for pull requests in GitHub", Information and Software Technology , 129, 2021

[168] Jirat Pasuksmit, Patanamon Thongtanunam, Shanika Karunasekera ,”Story points changes in agile iterative development.” , Empirical Software Engineering , 27 (06) pp. 156 ,2022

[169] Jirayus Jiarpakdee, Chakkrit Kla Tantithamthavorn, Hoa Khanh Dam, John C. Grundy , “An Empirical Study of Model-Agnostic Techniques for Defect Prediction Models “ , IEEE Transactions on Software Engineering, 48 (02) pp. 166-185 ,2022

[170] Jirayus Jiarpakdee, Chakkrit Tantithamthavorn, Ahmed E. Hassan , “The Impact of Correlated Metrics on the Interpretation of Defect Models” , IEEE Transactions on Software Engineering, 47 (02) pp, 320-331 ,2021

[171] Joseph Hejderup, Georgios Gousios ,”Can we trust tests to automate dependency updates? A case study of Java Projects”, Journal of Systems and Software, 183 (01) pp. 111097 ,2022

[172] Joshua Garcia, Ehsan Kouroshfar, Negar Ghorbani, Sam Malek ,”Forecasting Architectural Decay From Evolutionary History. “ , IEEE Transactions on Software Engineering, 48 (07) pp,2439-2454 ,2022

[173] Juan Manuel Florez, Laura Moreno, Zenong Zhang, Shiyi Wei, Andrian Marcus ,”An empirical study of data constraint implementations in Java” , Empirical Software Engineering , 27 (05) pp. 119 ,2022

[174] Jun Wang, Xiaofang Zhang, Lin Chen, and Xiaoyuan Xie, Personalizing label prediction for GitHub issues, Information and Software Technology, 145,2022

[175] K. Ayberk Tecimer, Eray Tüzün, Cansu Moran, and Hakan Erdogmus, Cleaning ground truth data in software task assignment, Information and Software Technology, 149,2022

[176] Kaiyuan Yang, Junfeng Wang, Zhiyang Fang, Peng Wu, and Zihua Song, Enhancing software modularization via semantic outliers filtration and label propagation, Information and Software Technology, 145,2022

[177] Katerina Paltoglou, Vassilis E. Zafeiris, N. A. Diamantidis, Emmanouel A. Giakoumakis, “Automated refactoring of legacy JavaScript code to ES6 modules.” , Journal of Systems and Software, 181 (11) pp. 111049 ,2021

[178] Kewen Peng, Tim Menzies , “Defect Reduction Planning (Using TimeLIME). “ , IEEE Transactions on Software Engineering, 48 (07) pp, 2510-2525,2022

[179] Khairul Islam, Toufique Ahmed, Rifat Shahriyar, Anindya Iqbal, and Gias Uddin, Early prediction for merged vs abandoned code changes in modern code reviews, Information and Software Technology, 142,2022

[180] Khaled Sellami, Ali Ouni, Mohamed Aymen Saied, Salah Bouktif, and Mohamed Wiem Mkaouer, Improving microservices extraction using evolutionary search, Information and Software Technology,151,2022

[181] Khalid Alkharabsheh, Sadi Alawadi, Victor R. Kebande, Yania Crespo, Manuel Fernández-Delgado, and José A. Taboada, A comparison of machine learning algorithms on design smell detection using balanced and imbalanced dataset: A study of God class, Information and Software Technology, 143,2022

[182] Khashayar Etemadi, Niloofar Tarighat, Siddharth Yadav, Matias Martinez, Martin Monperrus ,"Estimating the potential of program repair search spaces with commit analysis.".Journal of Systems and Software, 188 (06) pp. 111263 ,2022

[183] Krishna Patel, Robert M. Hierons, and David Clark, An information theoretic notion of software testability, Information and Software Technology, 143,2022

[184] Kui Liu, Dongsun Kim, Tegawendé F. Bissyandé, Shin Yoo, Yves Le Traon, "Mining Fix Patterns for FindBugs Violations." , IEEE Transactions on Software Engineering, 47 (01) pp, 165-188 ,2021

[185] Kui Liu, Li Li, Anil Koyuncu, Dongsun Kim, Zhe Liu, Jacques Klein, Tegawendé F. Bissyandé , "A critical review on the evaluation of automated program repair systems" , Journal of Systems and Software, 171 (01) pp. 110817 ,2021

[186] Kunsong Zhao, Zhou Xu, Meng Yan, Tao Zhang, Dan Yang, and Wei Li, A comprehensive investigation of the impact of feature selection techniques on crashing fault residence prediction models, Information and Software Technology,139,2021

[187] Laerte Xavier, João Eduardo Montandon, Fabio Ferreira, Rodrigo Brito, Marco Túlio Valente, "On the documentation of self-admitted technical debt in issues.", Empirical Software Engineering , 27 (07) pp. 163 ,2022

[188] Lam Nguyen Tung, Hoang-Viet Tran, Khoi Nguyen Le, and Pham Ngoc Hung, An automated test data generation method for void pointers and function pointers in C/C++ libraries and embedded projects, Information and Software Technology, 145,2022

[189] Le Yu, Xiapu Luo, Jiachi Chen, Hao Zhou, Tao Zhang, Henry Chang, Hareton K. N. Leung, "PPChecker: Towards Accessing the Trustworthiness of Android Apps' Privacy Policies.", IEEE Transactions on Software Engineering, 47 (02) pp,221-242 ,2021

[190] Lin Jiang, Hui Liu, He Jiang, Lu Zhang, Hong Mei ,"Heuristic and Neural Network Based Prediction of Project-Specific API Member Access " , IEEE Transactions on Software Engineering, 48 (04) pp. 1249-1267,2022

[191] Liu Wang, Ren He, Haoyu Wang, Pengcheng Xia, Yuanchun Li, Lei Wu, Yajin Zhou, Xiapu Luo, Yulei Sui, Yao Guo, and Guoai Xu, Beyond the virus: a first look at coronavirus-themed Android malware, Empirical Software Engineering, 26(4), 2021

[192] Liushan Chen, Yu Pei, Carlo A. Furia , "Contract-Based Program Repair Without The Contracts: An Extended Study" , IEEE Transactions on Software Engineering, 47 (12) pp. 2841-2857,2021

[193] Long Zhang, Brice Morin, Philipp Haller, Benoit Baudry, Martin Monperrus ,"A Chaos Engineering System for Live Analysis and Falsification of Exception-Handling in the JVM." ,IEEE Transactions on Software Engineering, 47 (11) pp. 2534-2548 ,2021

[194] Lu Xiao, Yuanfang Cai, Rick Kazman, Ran Mo, Qiong Feng , "Detecting the Locations and Predicting the Maintenance Costs of Compound Architectural Debts " , IEEE Transactions on Software Engineering, 48 (09) pp, 3686-3715 ,2022

[195] Luan P. Lima, Lincoln S. Rocha, Carla I. M.Bezerra, and Matheus Paixao, Assessing exception handling testing practices in open-source libraries, Empirical Software Engineering, 26(5), 2021

[196] Luca Traini, Daniele Di Pompeo, Michele Tucci, Bin Lin, Simone Scalabrino, Gabriele Bavota, Michele Lanza, Rocco Oliveto, and Vittorio Cortellessa. 2021. How Software Refactoring Impacts Execution Time. ACM Trans. Softw. Eng. Methodol.

[197] Luis Cruz, Rui Abreu , "On the Energy Footprint of Mobile Testing Frameworks" , IEEE Transactions on Software Engineering, 47 (10) pp. 2260-2271 ,2021

[198] Luiz Alberto Ferreira Gomes, Ricardo da Silva Torres, and Mario Lúcio Côrtes, On the prediction of long-lived bugs: An analysis and comparative study using FLOSS projects, Information and Software Technology, Volume 132, 2021

[199] Luiz Laerte Nunes da Silva, Troy Costa Kohwalter, Alexandre Plastino, and Leonardo Gresta Paulino Murta, Sequential coding patterns: How to use them effectively in code recommendation, Information and Software Technology, 140,2021

[200] M. Nosrati, H. Haghighi, and M. Vahidi Asl, Test data generation using genetic programming, Information and Software Technology, Volume 130, 2021

[201] Majid Hatamian, Samuel Wairimu, Nurul Momen, and Lothar Fritsch, A privacy and security analysis of early-deployed COVID-19 contact tracing Android apps, Empirical Software Engineering, 26(3), 2021

[202] Maleknaz Nayebi, Guenther Ruhe, Thomas Zimmermann ,"Mining Treatment-Outcome Constructs from Sequential Software Engineering Data", IEEE Transactions on Software Engineering, 47 (02) pp, 393-411 ,2021

[203] Malinda Dilhara, Ameya Ketkar, and Danny Dig. 2021. Understanding Software-2.0: A Study of Machine Learning Library Usage and Evolution. ACM Trans. Softw. Eng. Methodol.

[204] Man Zhang, Bogdan Marculescu, and Andrea Arcuri, Resource and dependency based test case generation for RESTful Web services, Empirical Software Engineering, 26(4), 2021

[205] Manish Motwani, Mauricio Soto, Yuriy Brun, René Just, Claire Le Goues, "Quality of Automated Program Repair on Real-World Defects." , IEEE Transactions on Software Engineering, 48 (02) pp. 637-661 ,2022

[206] Manuel Ohrndorf, Christopher Pietsch, Udo Kelter, Lars Grunske, and Timo Kehrer. 2021. History-based Model Repair Recommendations. ACM Trans. Softw. Eng. Methodol.

[207] Marc-André Laverdière, Karl Julien,and Ettore Merlo, RBAC protection-impacting changes identification: A case study of the security evolution of two PHP applications, Information and Software Technology,139,2021

[208] Maria Kechagia, Sergey Mechtaev, Federica Sarro, Mark Harman ,"Evaluating Automatic Program Repair Capabilities to Repair API Misuses. " , IEEE Transactions on Software Engineering, 48 (07) pp, 2658-2679,2022

[209] Maria Ulan, Welf Löwe, Morgan Ericsson, and Anna Wingkvist, Weighted software metrics aggregation and its application to defect prediction, Empirical Software Engineering, 26(5), 2021

[210] Marianna Di Gregorio, Dario Di Nucci, Fabio Palomba, Giuliana Vitiello ,"The making of accessible Android applications: an empirical study on the state of the practice" , Empirical Software Engineering , 27 (06) pp. 145 ,2022

[211] Mario Janke, Patrick Mäder ,"Graph Based Mining of Code Change Patterns From Version Control Commits " , IEEE Transactions on Software Engineering, 48 (03) pp. 848-863 ,2022

[212] Martina Iammarino, Fiorella Zampetti, Lerina Aversano, Massimiliano Di Penta, “An empirical study on the co-occurrence between refactoring actions and Self-Admitted Technical Debt removal” , Journal of Systems and Software, 178 (08) pp. 110976 ,2021

[213] Maryam Raiyat Aliabadi, Mojtaba Vahidi-Asl, Ramak Ghavamizadeh , “ARTINALI++: Multi-dimensional Specification Mining for Complex Cyber-Physical System Security”, Journal of Systems and Software, 180 (10) pp. 111016 ,2021

[214] Masanari Kondo, Yutaro Kashiwa, Yasutaka Kamei, Osamu Mizuno , “An empirical study of issue-link algorithms: which issue-link algorithms should we use?” , Empirical Software Engineering , 27 (06) pp. 136 ,2022

[215] Mateus Lopes, André C. Hora ,”How and why we end up with complex methods: a multi-language study” , Empirical Software Engineering , 27 (05) pp. 115 ,2022

[216] Matheus Paixão, Jens Krinke, DongGyun Han, Chaiyong Ragkhitwetsagul, Mark Harman ,”The Impact of Code Review on Architectural Changes.” , IEEE Transactions on Software Engineering, 47 (05) pp. 1041-1059 ,2021

[217] Mathias Hedenborg, Jonas Lundberg, Welf Löwe, “Memory efficient context-sensitive program analysis”, Journal of Systems and Software, 177 (07) pp. 110952 ,2021

[218] Mathieu Nassif, Alexa Hernandez, Ashvitha Sridharan, Martin P. Robillard ,”Generating Unit Tests for Documentation “, IEEE Transactions on Software Engineering, 48 (09) pp, 3268-3279,2022

[219] Matteo Biagiola and Paolo Tonella. 2022. Testing the Plasticity of Reinforcement Learning-based Systems. ACM Trans. Softw. Eng. Methodol.

[220] Maxime Lamothe, Heng Li, Weiyi Shang, “Assisting Example-Based API Misuse Detection via Complementary Artificial Examples “, IEEE Transactions on Software Engineering, 48 (09) pp, 3410-3422,2022

[221] Md Hasan Ibrahim, Mohammed Sayagh, and Ahmed E. Hassan, A study of how Docker Compose is used to compose multi-component systems, Empirical Software Engineering, 26(6), 2021

[222] Md. Imran Alam, Raju Halder, Jorge Sousa Pinto, “A deductive reasoning approach for database applications using verification conditions” , Journal of Systems and Software, 175 (05) pp. 110903 ,2021

[223] Md. Nadim, Manishankar Mondal, Chanchal K. Roy, Kevin A. Schneider ,"Evaluating the performance of clone detection tools in detecting cloned co-change candidates." ,Journal of Systems and Software, 187 (05) pp. 111229 ,2022

[224] Mehrdad Abdi, Henrique Rocha, Serge Demeyer, Alexandre Bergel ,"Small-Amp: Test amplification in a dynamically typed language" , Empirical Software Engineering , 27 (06) pp. 128 ,2022

[225] Meng Yan, Xin Xia, Yuanrui Fan, Ahmed E. Hassan, David Lo, Shanping Li , "Just-In-Time Defect Identification and Localization: A Two-Phase Framework." , IEEE Transactions on Software Engineering, 48 (02) pp. 82-101 ,2022

[226] Mengshi Zhang, Yaoxian Li, Xia Li, Lingchao Chen, Yuqun Zhang, Lingming Zhang, Sarfraz Khurshid, "An Empirical Study of Boosting Spectrum-Based Fault Localization via PageRank", IEEE Transactions on Software Engineering, 47 (06) pp. 1089-1113 ,2021

[227] Miao Zhang, Jacky Wai Keung, Yan Xiao, and Md Alamgir Kabir, "Evaluating the effects of similar-class combination on class integration test order generation", Information and Software Technology,129,2021

[228] Min Kyung Shin, Sudipto Ghosh, Leo R. Vijayasarathy ,"An empirical comparison of four Java-based regression test selection techniques.", Journal of Systems and Software, 186 (04) pp. 111174 ,2022

[229] Ming Wen, Junjie Chen, Yongqiang Tian, Rongxin Wu, Dan Hao, Shi Han, Shing-Chi Cheung, "Historical Spectrum Based Fault Localization" , IEEE Transactions on Software Engineering, 47 (11) pp. 2348-2368 ,2021

[230] Minxue Pan, Tongtong Xu, Yu Pei, Zhong Li, Tian Zhang, Xuandong Li , "GUI-Guided Test Script Repair for Mobile Apps. " , IEEE Transactions on Software Engineering, 48 (03) pp.910-929 ,2022

[231] Mitchell Joblin and Sven Apel. 2022. How Do Successful and Failed Projects Differ? A Socio-Technical Analysis. ACM Trans. Softw. Eng. Methodol.

[232] Mohamed Lamine Kerdoudi, Tewfik Ziadi, Chouki Tibermacine, Salah Sadou, "A novel approach for Software Architecture Product Line Engineering",Journal of Systems and Software, 186 (04) pp. 111191 ,2022

[233] Mohammad Javad Beheshtian, Amir Hossein Bavand, Peter C. Rigby , "Software Batch Testing to Save Build Test Resources and to Reduce Feedback Time. " , IEEE Transactions on Software Engineering, 48 (08) pp,2784-2801 ,2022

[234] Mohammad Masudur Rahman, Foutse Khomh, Marco Castelluccio ,"Works for Me! Cannot Reproduce - A Large Scale Empirical Study of Non-reproducible Bugs" , Empirical Software Engineering , 27 (05) pp. 111 ,2022

[235] Mohammad Masudur Rahman, Foutse Khomh, Shamima Yeasmin, and Chanchal K.Roy, The forgotten role of search queries in IR-based bug localization: an empirical study, Empirical Software Engineering, 26(6), 2021

[236] Mohammed Sayagh, Ahmed E. Hassan,"ConfigMiner: Identifying the Appropriate Configuration Options for Config-Related User Questions by Mining Online Forums" , IEEE Transactions on Software Engineering, 47 (12) pp. 2907-2918,2021

[237] Mojtaba Bagherzadeh, Nafiseh Kahani, Lionel C. Briand ,"Reinforcement Learning for Test Case Prioritization ", IEEE Transactions on Software Engineering, 48 (08) pp, 2836-2856,2022

[238] Morena Barboni, Andrea Morichetta, Andrea Polini ,"SuMo: A mutation testing approach and tool for the Ethereum blockchain" ,Journal of Systems and Software, 193 (11) pp. 111445 ,2022

[239] Moses Openja, Mohammad Mehdi Morovati, Le An, Foutse Khomh, Mouna Abidi, "Technical debts and faults in open-source quantum software systems: An empirical study",Journal of Systems and Software, 193 (11) pp. 111458 ,2022

[240] Mouna Abidi, Md Saidur Rahman, Moses Openja, and Foutse Khomh. 2021. Are Multi-Language Design Smells Fault-Prone? An Empirical Study. ACM Trans. Softw. Eng. Methodol.

[241] Navid Teymourian, Habib Izadkhah, Ayaz Isazadeh ,"A Fast Clustering Algorithm for Modularization of Large-Scale Software Systems. " , IEEE Transactions on Software Engineering, 48 (04) pp. 1451-1462 ,2022

[242] Nemania Borovits, Indika Kumara, Dario Di Nucci, Parvathy Krishnan, Stefano Dalla Palma, Fabio Palomba, Damian A. Tamburri, Willem-Jan van den Heuvel ,"FindICI: Using machine learning to detect linguistic inconsistencies between code and natural language descriptions in infrastructure-as-code", Empirical Software Engineering , 27 (07) pp. 178 ,2022

[243] Nezih Sunman, Yigit Soydan, Hasan Sözer ,"Automated Web application testing driven by pre-recorded test case" ,Journal of Systems and Software, 193 (11) pp. 111441 ,2022

[244] Nicolas Harrand, Amine Benelallam, César Soto-Valero, François Bettega, Olivier Barais, Benoit Baudry, “API beauty is in the eye of the clients: 2.2 million Maven dependencies reveal the spectrum of client-API usages” ,Journal of Systems and Software, 184 (02) pp. 111134 ,2022

[245] Nima Miryeganeh, Sepehr Hashtroudi, Hadi Hemmati ,”GloBug: Using global data in Fault Localization.” ,Journal of Systems and Software, 177 (07) pp. 110961 ,2021

[246] Olivier Nourry, Yutaro Kashiwa, Yasutaka Kamei, and Naoyasu Ubayashi, Does shortening the release cycle affect refactoring activities: A case study of the JDT Core, Platform SWT, and UI projects, Information and Software Technology,139,2021

[247] Osamah AlDhafer, Irfan Ahmad, and Sajjad Mahmood, An end-to-end deep learning system for requirements classification using recurrent neural networks, Information and Software Technology, 147,2022

[248] Oumayma Hamdi, Ali Ouni, Mel Ó Cinnéide, and Mohamed Wiem Mkaouer, A longitudinal study of the impact of refactoring in android applications, Information and Software Technology, 140,2021

[249] Patrick Keller, Abdoul Kader Kaboré, Laura Plein, Jacques Klein, Yves Le Traon, and Tegawendé F. Bissyandé. 2021. What You See is What it Means! Semantic Representation Learning of Code based on Visualization and Transfer Learning. ACM Trans. Softw. Eng. Methodol.

[250] Pattara Leelaprute, and Sousuke Amasaki, A comparative study on vectorization methods for non-functional requirements classification, Information and Software Technology,150,2022

[251] Paul Muntean, Martin Monperrus, Hao Sun, Jens Grossklags, Claudia Eckert , “ IntRepair: Informed Repairing of Integer Overflows” , IEEE Transactions on Software Engineering, 47 (10) pp.2225-2241 ,2021

[252] Paul Temple, Mathieu Acher, Jean-Marc Jézéquel ,”Empirical Assessment of Multimorphic Testing.” , IEEE Transactions on Software Engineering, 47 (07) pp. 1511-1527 ,2021

[253] Peipei Wang, Chris Brown, Jamie A. Jennings, Kathryn T. Stolee, “Demystifying regular expression bugs.”, Empirical Software Engineering , 27 (01) pp. 21 ,2022

[254] Peng Zhang, Yang Wang, Xutong Liu, Yanhui Li, Yibiao Yang, Ziyuan Wang, Xiaoyu Zhou, Lin Chen, and Yuming Zhou. 2022. Mutant Reduction Evaluation: What is There and What is Missing? ACM Trans. Softw. Eng. Methodol.

[255] Peng Zhang, Yanhui Li, Wanwangying Ma, Yibiao Yang, Lin Chen, Hongmin Lu, Yuming Zhou, Baowen Xu , “CBUA: A Probabilistic, Predictive, and Practical Approach for Evaluating Test Suite Effectiveness “ , IEEE Transactions on Software Engineering, 48 (03) pp. 1067-1096,2022

[256] Phuong T. Nguyen, Juri Di Rocco, Claudio Di Sipio, Davide Di Ruscio, Massimiliano Di Penta ,”Recommending API Function Calls and Code Snippets to Support Software Development “ , IEEE Transactions on Software Engineering, 48 (07) pp, 2417-2438,2022

[257] Pooja Rani, Sebastiano Panichella, Manuel Leuenberger, Andrea Di Sorbo, Oscar Nierstrasz, “How to identify class comment types? A multi-language approach for class comment classification”, Journal of Systems and Software, 181 (11) pp. 111047 ,2021

[258] Pooja Rani, Sebastiano Panichella, Manuel Leuenberger, Mohammad Ghafari, and Oscar Nierstrasz, What do class comments tell us? An investigation of comment evolution and practices in Pharo Smalltalk, Empirical Software Engineering, 26(6), 2021

[259] Qiang Hu, Yuejun Guo, Maxime Cordy, Xiaofei Xie, Lei Ma, Mike Papadakis, and Yves Le Traon. 2022. An Empirical Study on Data Distribution-Aware Test Selection for Deep Learning Enhancement. ACM Trans. Softw. Eng. Methodol.

[260] Qianqian Zhu, Andy Zaidman, Annibale Panichella, “How to kill them all: An exploratory study on the impact of code observability on mutation testing” , Journal of Systems and Software, 173 (03) pp. 110864 ,2021

[261] Quang-Hung Luu, Man Fai Lau, Sebastian P. H. Ng, Tsong Yueh Chen, “Testing multiple linear regression systems with metamorphic testing”.Journal of Systems and Software, 182 (12) pp. 111062 ,2021

[262] Quanjun Zhang, Chunrong Fang, Weisong Sun, Shengcheng Yu, Yutao Xu, Yulei Liu , “Test case prioritization using partial attention” , Journal of Systems and Software, 192 (10) pp. 111419 ,2022

[263] Quanyi Zou, Lu Lu, Zhanyu Yang, Xiaowei Gu,and Shaojian Qiu, Joint feature representation learning and progressive distribution matching for cross-project defect prediction, Information and Software Technology,137,2021

[264] Qun Mao, Weiwei Wang, Feng You, Ruilian Zhao, Zheng Li, “User behavior pattern mining and reuse across similar Android apps” , Journal of Systems and Software, 183 (01) pp. 111085 ,2022

[265] Rabe Abdalkareem, Suhaib Mujahid, Emad Shihab , “A Machine Learning Approach to Improve the Detection of CI Skip Commits” , IEEE Transactions on Software Engineering, 47 (12) pp. 2740-2754 ,2021

[266] Rafael Caballero, Enrique Martin-Martin, Adrián Riesco,and Salvador Tamarit, "A unified framework for declarative debugging and testing", Information and Software Technology,129,2021

[267] Rafi Almhana, Marouane Kessentini,and Wiem Mkaouer, Method-level bug localization using hybrid multi-objective search, Information and Software Technology, Volume 131, 2021

[268] Rahul Krishna, Chong Tang, Kevin J. Sullivan, Baishakhi Ray , “ConEx: Efficient Exploration of Big-Data System Configurations for Better Performance “ , IEEE Transactions on Software Engineering, 48 (03) pp. 893-909 ,2022

[269] Rahul Krishna, Vivek Nair, Pooyan Jamshidi, Tim Menzies, “Whence to Learn? Transferring Knowledge in Configurable Systems Using BEETLE” , IEEE Transactions on Software Engineering, 47 (12) pp. 2956-2972 ,2021

[270] Rahul Yedida, Tim Menzies , “On the Value of Oversampling for Deep Learning in Software Defect Prediction “ , IEEE Transactions on Software Engineering, 48 (08) pp, 3103-3116 ,2022

[271] Ran Mo, Shaozhi Wei, Qiong Feng, Zengyang Li, An exploratory study of bug prediction at the method level, Information and Software Technology, 144,2022

[272] Ran Mo, Yuanfang Cai, Rick Kazman, Lu Xiao, Qiong Feng ,”Architecture Anti-Patterns: Automatically Detectable Violations of Design Principles”, IEEE Transactions on Software Engineering, 47 (05) pp. 1008-1028 ,2021

[273] Ratnadira Widyasari, Gede Artha Azriadi Prana, Stefanus Agus Haryono, Shaowei Wang, David Lo ,”Real world projects, real faults: evaluating spectrum based fault localization techniques on Python projects” , Empirical Software Engineering , 27 (06) pp. 147 ,2022

[274] Rezvan Mahdavi-Hezaveh, Nirav Ajmeri, and Laurie Williams, Feature toggles as code: Heuristics and metrics for structuring feature toggles, Information and Software Technology, 145,2022

[275] Riccardo Coppola, Luca Ardito, Marco Torchiano, Emil Alégroth ,"Translation from layout-based to visual android test scripts: An empirical evaluation.", Journal of Systems and Software, 171 (01) pp. 110845 ,2021

[276] Richárd Szalay, Ábel Sinkovics, Zoltán Porkoláb, "Practical heuristics to improve precision for erroneous function argument swapping detection in C and C++." ,Journal of Systems and Software, 181 (11) pp. 111048 ,2021

[277] Ridhi Jain, Rahul Purandare, and Subodh Sharma. 2022. BiRD: Race Detection in Software Binaries under Relaxed Memory Models. ACM Trans. Softw. Eng. Methodol.

[278] Ridwan Salihin Shariffdeen, Shin Hwei Tan, Mingyuan Gao, Abhik Roychoudhury. 2021. Automated Patch Transplantation. ACM Trans. Softw. Eng. Methodol.

[279] Robert White, Jens Krinke ,"TCTracer: Establishing test-to-code traceability links using dynamic and static techniques." , Empirical Software Engineering , 27 (03) pp. 67 ,2022

[280] Roberto Natella , "StateAFL: Greybox fuzzing for stateful network servers" , Empirical Software Engineering , 27 (07) pp. 190 ,2022

[281] Rohit Gheyi, Márcio Ribeiro, Beatriz Souza, Marcio Guimarães, Leo Fernandes, Marcelo d'Amorim, Vander Alves, Leopoldo Teixeira, and Baldoino Fonseca, Identifying method-level mutation subsumption relations using Z3, Information and Software Technology, Volume 132, 2021

[282] Roland Kretschmer, Djamel Eddine Khelladi, Alexander Egyed , "Transforming abstract to concrete repairs with a generative approach of repair values" , Journal of Systems and Software, 175 (05) pp. 110889 ,2021

[283] Rômulo Manciola Meloca, Ingrid Nunes ,"A comparative study of application-level caching recommendations at the method level, Empirical Software Engineering , 27 (04) pp. 88 ,2022

[284] Ruben Heradio, David Fernández-Amorós, José A. Galindo, David Benavides, Don S. Batory ,"Uniform and scalable sampling of highly configurable systems.", Empirical Software Engineering , 27 (02) pp. 44 ,2022

[285] Sandra L. Ramírez-Mora, Hanna Oktaba, Helena Gómez-Adorno,and Gerardo Sierra, Exploring the communication functions of comments during bug fixing in Open Source Software projects, Information and Software Technology,136,2021

[286] Sarra Habchi, Naouel Moha, Romain Rouvoy, “Android code smells: From introduction to refactoring.” , Journal of Systems and Software, 177 (07) pp. 110964 ,2021

[287] Saurabh Malgaonkar, Sherlock A. Licorish, and Bastin Tony Roy Savarimuthu, Prioritizing user concerns in app reviews – A study of requests for new features, enhancements and bug fixes, Information and Software Technology,144,2022

[288] Sebastiano Panichella, Gerardo Canfora, and Andrea Di Sorbo, “Won’t We Fix this Issue?” Qualitative characterization and automated identification of wontfix issues on GitHub, Information and Software Technology,139,2021

[289] Seonah Lee, Rongxin Wu, Shing-Chi Cheung, Sungwon Kang , “Automatic Detection and Update Suggestion for Outdated API Names in Documentation.” ,IEEE Transactions on Software Engineering, 47 (04) pp,653-675 ,2021

[290] Shaiful Alam Chowdhury, Reid Holmes, Andy Zaidman, Rick Kazman , “Revisiting the debate: Are code metrics useful for measuring maintenance effort?” , Empirical Software Engineering , 27 (06) pp. 158 ,2022

[291] Shangqing Liu, Cuiyun Gao, Sen Chen, Lun Yiu Nie, Yang Liu , “ATOM: Commit Message Generation Based on Abstract Syntax Tree and Hybrid Ranking.” , IEEE Transactions on Software Engineering, 48 (05) pp. 1800-1817 ,2022

[292] Shayan Zamani, and Hadi Hemmati, A pragmatic approach for hyper-parameter tuning in search-based test case generation, Empirical Software Engineering, 26(6), 2021

[293] Shiran Liu, Zhaoqiang Guo, Yanhui Li, Hongmin Lu, Lin Chen, Lei Xu, Yuming Zhou, and Baowen Xu, Prioritizing code documentation effort: Can we do it simpler but better? , Information and Software Technology,140,2021

[294] Shivashree Vysali, Shane McIntosh, Bram Adams ,”Quantifying, Characterizing, and Mitigating Flakily Covered Program Elements” , IEEE Transactions on Software Engineering, 48 (03) pp.1018-1029 ,2022

[295] Shiwen Yu, Ting Wang, Ji Wang ,”Data Augmentation by Program Transformation” ,Journal of Systems and Software, 190 (08) pp. 111304 ,2022

[296] Shiyue Rong, Weisheng Wang, Umme Ayda Mannan, Eduardo Santana de Almeida, Shurui Zhou, and Iftekhar Ahmed, An empirical study of emoji use in software development communication, Information and Software Technology,148,2022

[297] Shouvick Mondal, Rupesh Nasre, “Hansie: Hybrid and consensus regression test prioritization” , Journal of Systems and Software, 172 (02) pp. 110850 ,2021

[298] Shujuan Jiang, Miao Zhang, Yanmei Zhang, Rongcun Wang, Qiao Yu, Jacky Wai Keung, “An Integration Test Order Strategy to Consider Control Coupling.” , IEEE Transactions on Software Engineering, 47 (07) pp. 1350-1367 ,2021

[299] Shuo Feng, Jacky Keung, Peichang Zhang, Yan Xiao, and Miao Zhang, The impact of the distance metric and measure on SMOTE-based techniques in software defect prediction, Information and Software Technology,142,2022

[300] Shuo Feng, Jacky Keung, Xiao Yu, Yan Xiao, and Miao Zhang, Investigation on the stability of SMOTE-based oversampling techniques in software defect prediction, Information and Software Technology,139,2021

[301] Shuo Feng, Jacky Keung, Xiao Yu, Yan Xiao, Kwabena Ebo Bennin, Md Alamgir Kabir,and Miao Zhang, "COSTE: Complexity-based OverSampling TEchnique to alleviate the class imbalance problem in software defect prediction", Information and Software Technology,129,2021

[302] Sicong Cao, Xiaobing Sun, Lili Bo, Ying Wei,and Bin Li, BGNN4VD: Constructing Bidirectional Graph Neural-Network for Vulnerability Detection, Information and Software Technology,136,2021

[303] Simone Scalabrino, Antonio Mastropaolo, Gabriele Bavota, and Rocco Oliveto. 2021. An Adaptive Search Budget Allocation Approach for Search-Based Test Case Generation. ACM Trans. Softw. Eng. Methodol.

[304] Sofia Reis, Rui Abreu, and Luis Cruz, Fixing vulnerabilities potentially hinders maintainability, Empirical Software Engineering, 26(6), 2021

[305] Sofien Boutaib, Maha Elarbi, Slim Bechikh, Fabio Palomba, Lamjed Ben Said , “Handling uncertainty in SBSE: a possibilistic evolutionary approach for code smells detection” , Empirical Software Engineering , 27 (06) pp. 124 ,2022

[306] Solm Salimi, Mehdi Kharrazi ,”VulSlicer: Vulnerability detection through code slicing” ,Journal of Systems and Software, 193 (11) pp. 111450 ,2022

[307] Song Wang, Chetan Bansal,and Nachiappan Nagappan, "Large-scale intent analysis for identifying large-review-effort code changes", Information and Software Technology,130,2021

[308] Sooyoung Cha, Seongjoon Hong, Jiseong Bak, Jingyoung Kim, Junhee Lee, Hakjoo Oh , “Enhancing Dynamic Symbolic Execution by Automatically Learning

Search Heuristics “, IEEE Transactions on Software Engineering, 48 (09) pp, 3640-3663 ,2022

[309] Sophia Quach, Maxime Lamothe, Yasutaka Kamei, and Weiyi Shang, An empirical study on the use of SZZ for identifying inducing changes of non-functional bugs, Empirical Software Engineering, 26(4), 2021

[310] Stefanus A. Haryono, Ferdian Thung, David Lo, Lingxiao Jiang, Julia Lawall, Hong Jin Kang, Lucas Serrano, Gilles Muller, “AndroEvolve: automated Android API update with data flow analysis and variable denormalization.”, Empirical Software Engineering , 27 (03) pp. 73 ,2022

[311] Steffen Herbold, Alexander Trautsch, Fabian Trautsch, Benjamin Ledel ,”Problems with SZZ and features: An empirical study of the state of practice of defect prediction data collection.” , Empirical Software Engineering , 27 (02) pp. 42 ,2022

[312] Steffen Tunkel, Steffen Herbold,”Exploring the relationship between performance metrics and cost saving potential of defect prediction models.” , Empirical Software Engineering , 27 (07) pp. 182 ,2022

[313] Steven Locke, Heng Li, Tse-Hsun Peter Chen, Weiyi Shang, Wei Liu , “LogAssist: Assisting Log Analysis Through Log Summarization “ , IEEE Transactions on Software Engineering, 48 (09) pp, 3227-3241 ,2022

[314] Supatsara Wattanakriengkrai, Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Hideaki Hata, Kenichi Matsumoto ,”Predicting Defective Lines Using a Model-Agnostic Technique. “ , IEEE Transactions on Software Engineering, 48 (05) pp.1480-1496 ,2022

[315] Suppawong Tuarob, Noppadol Assavakamhaenghan, Waralee Tanaphantaruk, Ponlakit Suwanworaboon, Saeed-Ul Hassan, and Morakot Choetkiertikul, Automatic team recommendation for collaborative software development, Empirical Software Engineering, 26(4), 2021

[316] Suyu Ma, Zhenchang Xing, Chunyang Chen, Cheng Chen, Lizhen Qu, Guoqiang Li ,”Easy-to-Deploy API Extraction by Multi-Level Feature Embedding and Transfer Learning” , IEEE Transactions on Software Engineering, 47 (10) pp.2296-2311 ,2021

[317] Taher Ahmed Ghaleb, Daniel Alencar da Costa, Ying Zou, Ahmed E. Hassan , “Studying the Impact of Noises in Build Breakage Data” , IEEE Transactions on Software Engineering, 47 (09) pp. 1998-2011 ,2021

[318] Tao Zhang, Jiachi Chen, Xian Zhan, Xiapu Luo, David Lo, He Jiang, “Where2Change: Change Request Localization for App Reviews.”, IEEE Transactions on Software Engineering, 47 (11) pp. 2590-2616 ,2021

[319] Thazin Win Win Aung, Yao Wan, Huan Huo, Yulei Sui ,”Multi-triage: A multi-task learning framework for bug triage” , Journal of Systems and Software, 184 (02) pp. 111133 ,2022

[320] Thierry Titcheu Chekam, Mike Papadakis, Maxime Cordy, and Yves Le Traon. 2021. Killing Stubborn Mutants with Symbolic Execution. ACM Trans. Softw. Eng. Methodol.

[321] Thomas Bock, Angelika Schmid, and Sven Apel. 2021. Measuring and Modeling Group Dynamics in Open-Source Software Development: A Tensor Decomposition Approach. ACM Trans. Softw. Eng. Methodol.

[322] Tianpei Xia, Rui Shu, Xipeng Shen, Tim Menzies , “Sequential Model Optimization for Software Effort Estimation “, IEEE Transactions on Software Engineering, 48 (06) pp.1994-2009 ,2022

[323] Tobias Olsson, Morgan Ericsson, Anna Wingkvist, “To automatically map source code entities to architectural modules with Naive Bayes.”,Journal of Systems and Software, 183 (01) pp. 111095 ,2022

[324] Tongtong Xu, Liushan Chen, Yu Pei, Tian Zhang, Minxue Pan, Carlo A. Furia, “Restore: Retrospective Fault Localization Enhancing Automated Program Repair “ , IEEE Transactions on Software Engineering, 48 (02) pp.309-326 ,2022

[325] Toshiki Hirao, Shane McIntosh, Akinori Ihara, Kenichi Matsumoto ,”Code Reviews With Divergent Review Scores: An Empirical Study of the OpenStack and Qt Communities. “, IEEE Transactions on Software Engineering, 48 (02) pp. 69-81,2022

[326] Truong Ho-Quang, Arif Nurwidyantoro, Satrio Adi Rukmono, Michel R. V. Chaudron, Fabian Fröding, Duy Nguyen Ngoc, “Role stereotypes in software designs and their evolution” , Journal of Systems and Software, 189 (07) pp. 111296 ,2022

[327] Valentina Lenarduzzi, Vili Nikkola, Nyyti Saarimäki, Davide Taibi , “Does code quality affect pull request acceptance? An empirical study.” , Journal of Systems and Software, 171 (01) pp. 110806 ,2021

[328] Valeria Pontillo, Fabio Palomba, Filomena Ferrucci ,”Static test flakiness prediction: How Far Can We Go?”, Empirical Software Engineering , 27 (07) pp. 187 ,2022

[329] Van-Thuan Pham, Marcel Böhme, Andrew E. Santosa, Alexandru Razvan Caciulescu, Abhik Roychoudhury, “Smart Greybox Fuzzing” , IEEE Transactions on Software Engineering, 47 (09) pp. 1980-1997 ,2021

[330] Veronika Dashuber, and Michael Philippsen, Trace visualization within the Software City metaphor: Controlled experiments on program comprehension, Information and Software Technology,150,2022

[331] Vijay Walunj, Gharib Gharibi, Rakan Alanazi, Yugyung Lee ,”Defect prediction using deep learning with Network Portrait Divergence for software evolution”, Empirical Software Engineering , 27 (05) pp. 118,2022

[332] Vu Nguyen,and Bach Le, RLTCP: A reinforcement learning approach to prioritizing automated user interface tests, Information and Software Technology,136,2021

[333] Wei Zheng, Tianren Shen, Xiang Chen, Peiran Deng ,”Interpretability application of the Just-in-Time software defect prediction model” ,Journal of Systems and Software, 188 (06) pp. 111245 ,2022

[334] Weifeng Pan, Ming Hua, Carl K. Chang, Zijiang Yang, Dae-Kyoo Kim , “ElementRank: Ranking Java Software Classes and Packages using a Multilayer Complex Network-Based Approach” , IEEE Transactions on Software Engineering, 47 (10) pp. 2272-2295 ,2021

[335] Wolfgang Mauerer, Mitchell Joblin, Damian A. Tamburri, Carlos V. Paradis, Rick Kazman, Sven Apel, “In Search of Socio-Technical Congruence: A Large-Scale Longitudinal Study “, IEEE Transactions on Software Engineering, 48 (08) pp, 3159-3184 ,2022

[336] Wuxia Jin, Ting Liu, Yuanfang Cai, Rick Kazman, Ran Mo, Qinghua Zheng,”Service Candidate Identification from Monolithic Systems Based on Execution Traces”, IEEE Transactions on Software Engineering, 47 (05) pp. 987-1007,2021

[337] Xi Xiao, Yuqing Pan, Bin Zhang, Guangwu Hu, Qing Li,and Runiu Lu, ALBFL: A novel neural ranking model for software fault localization via combining static and dynamic features, Information and Software Technology,139,2021

[338] Xiajing Wang, Changzhen Hu, Rui Ma, Donghai Tian, and Jinyuan He, CMFuzz: context-aware adaptive mutation for fuzzers, Empirical Software Engineering, 26(1), 2021

[339] Xiang Chen, Yanzhou Mu, Ke Liu, Zhanqi Cui, and Chao Ni, “Revisiting heterogeneous defect prediction methods: How far are we?” , Information and Software Technology, Volume 130, 2021

[340] Xiang Gao, Bo Wang, Gregory J. Duck, Ruyi Ji, Yingfei Xiong, and Abhik Roychoudhury. 2021. Beyond Tests: Program Vulnerability Repair via Crash Constraint Extraction. ACM Trans. Softw. Eng. Methodol.

[341] Xiangjuan Yao, Gongjie Zhang, Feng Pan, Dunwei Gong, Changqing Wei , “Orderly Generation of Test Data via Sorting Mutant Branches Based on Their Dominance Degrees for Weak Mutation Testing “ , IEEE Transactions on Software Engineering, 48 (04) pp. 1169-1184,2022

[342] Xiangying Dang, Dunwei Gong, Xiangjuan Yao, Tian Tian, Huai Liu , “Enhancement of Mutation Testing via Fuzzy Clustering and Multi-Population Genetic Algorithm “ , IEEE Transactions on Software Engineering, 48 (06) pp. 2141-2156 ,2022

[343] Xiao Cheng, Haoyu Wang, Jiayi Hua, Guoai Xu, and Yulei Sui. 2021. DeepWukong: Statically Detecting Software Vulnerabilities Using Deep Graph Neural Network. ACM Trans. Softw. Eng. Methodol.

[344] Xiao Ling, Rishabh Agrawal, Tim Menzies ,”How Different is Test Case Prioritization for Open and Closed Source Projects? “ , IEEE Transactions on Software Engineering, 48 (07) pp, 2526-2540,2022

[345] Xiaobo Yan, Bin Liu, Shihai Wang , “A Test Restoration Method based on Genetic Algorithm for effective fault localization in multiple-fault programs”, Journal of Systems and Software, 172 (02) pp. 110861 ,2021

[346] Xiaofei Xie, Tianlin Li, Jian Wang, Lei Ma, Qing Guo, Felix Juefei-Xu, and Yang Liu. 2022. NPC: Neuron Path Coverage via Characterizing Decision Logic of Deep Neural Networks. ACM Trans. Softw. Eng. Methodol.

[347] Xiaofeng Han, Amjed Tahir, Peng Liang, Steve Counsell, Kelly Blincoe, Bing Li, Yajing Luo ,”Code smells detection via modern code review: a study of the OpenStack and Qt communities” , Empirical Software Engineering , 27 (06) pp.127 ,2022

[348] Xiaoqin Fu, Haipeng Cai, Wen Li, and Li Li. 2021. SEADS: Scalable and Cost-effective Dynamic Dependence Analysis of Distributed Systems via Reinforcement Learning. ACM Trans. Softw. Eng. Methodol.

[349] Xiaoxue Ma, Jacky Keung, Zhen Yang, Xiao Yu, Yishu Li, and Hao Zhang, CASMS: Combining clustering with attention semantic model for identifying security bug reports, Information and Software Technology,147,2022

[350] Xiaoxue Ma, Shangru Wu, Ernest Pobee, Xiupei Mei, Hao Zhang, Bo Jiang, and Wing-Kwong Chan. 2021. RegionTrack: A Trace-Based Sound and Complete Checker to Debug Transactional Atomicity Violations and Non-Serializable Traces. ACM Trans. Softw. Eng. Methodol.

[351] Xiaoxue Wu, Wei Zheng, Xiang Chen, Yu Zhao, Tingting Yu,and Dejun Mu, Improving high-impact bug report prediction with combination of interactive machine learning and active learning, Information and Software Technology, Volume 133, 2021

[352] Xiaoyu Sun, Li Li, Tegawendé F. Bissyandé, Jacques Klein, Damien Octeau, and John Grundy. 2021. Taming Reflection: An Essential Step Toward Whole-program Analysis of Android Apps. ACM Trans. Softw. Eng. Methodol.

[353] Xiuting Ge, Chunrong Fang, Meiyuan Qian, Yu Ge, and Mingshuang Qing, Locality-based security bug report identification via active learning, Information and Software Technology, 147,2022

[354] Xueqi Yang, Jianfeng Chen, Rahul Yedida, Zhe Yu, and Tim Menzies, Learning to recognize actionable static code warnings (is intrinsically easy), Empirical Software Engineering, 26(3), 2021

[355] Yan Xiaobo, Liu Bin, Wang Shihai, An Dong, Zhu Feng,and Yang Yelin, Efilter: An effective fault localization based on information entropy with unlabelled test cases, Information and Software Technology, Volume 134, 2021

[356] Yanjie Zhao, Li Li, Xiaoyu Sun, Pei Liu,and John Grundy, Icon2Code: Recommending code implementations for Android GUI components, Information and Software Technology,138,2021

[357] Yaqin Zhou, Jing Kai Siow, Chenyu Wang, Shangqing Liu, and Yang Liu. 2021. SPI: Automated Identification of Security Patches via Commits. ACM Trans. Softw. Eng. Methodol.

[358] Yida Tao, Shan Tang, Yepang Liu, Zhiwu Xu, and Shengchao Qin. 2021. Speeding Up Data Manipulation Tasks with Alternative Implementations: An Exploratory Study. ACM Trans. Softw. Eng. Methodol.

[359] Yikun Hu, Hui Wang, Yuanyuan Zhang, Bodong Li, Dawu Gu , “A Semantics-Based Hybrid Approach on Binary Code Similarity Comparison” , IEEE Transactions on Software Engineering, 47 (06) pp. 1241-1258 ,2021

[360] Yilin Yang, Tianxing He, Zhilong Xia, and Yang Feng, A comprehensive empirical study on bug characteristics of deep learning frameworks, Information and Software Technology,151,2022

[361] Yin Liu, Siddharth Dhar, Eli Tilevich, “Only pay for what you need: Detecting and removing unnecessary TEE-based code.” ,Journal of Systems and Software, 188 (06) pp. 111253 ,2022

[362] Yingfei Xiong and Bo Wang. 2022. L2S: A Framework for Synthesizing the Most Probable Program under a Specification. ACM Trans. Softw. Eng. Methodol.

[363] Yingzhe Lyu, Heng Li, Mohammed Sayagh, Zhen Ming (Jack) Jiang, and Ahmed E. Hassan. 2021. An Empirical Study of the Impact of Data Splitting Decisions on the Performance of AIOps Solutions. ACM Trans. Softw. Eng. Methodol.

[364] Youngjoo Ko, Bin Zhu, Jong Kim ,”Fuzzing with automatically controlled interleavings to detect concurrency bugs.” ,Journal of Systems and Software, 191 (09) pp. 111379 ,2022

[365] Yu Nong, Haipeng Cai, Pengfei Ye, Li Li,and Feng Chen, Evaluating and comparing memory error vulnerability detectors, Information and Software Technology,137,2021

[366] Yu Qu, Jianlei Chi,and Heng Yin, Leveraging developer information for efficient effort-aware bug prediction, Information and Software Technology, 137,2021

[367] Yu Qu, Qinghua Zheng, Jianlei Chi, Yangxu Jin, Ancheng He, Di Cui, Hengshan Zhang, Ting Liu, “Using K-core Decomposition on Class Dependency Networks to Improve Bug Prediction Model's Practical Performance” , IEEE Transactions on Software Engineering, 47 (02) pp,348-366 ,2021

[368] Yu Wang, Fengjuan Gao, Linzhang Wang, Tingting Yu, Jianhua Zhao, Xuandong Li , “Automatic Detection, Validation, and Repair of Race Conditions in Interrupt-Driven Embedded Software “,IEEE Transactions on Software Engineering, 48 (02) pp.346-363,2022

[369] Yu Zhou, Xinying Yang, Taolue Chen, Zhiqiu Huang, Xiaoxing Ma, Harald C. Gall ,”Boosting API Recommendation With Implicit Feedback “ ,IEEE Transactions on Software Engineering, 48 (06) pp. 2157-2172 ,2022

[370] Yu Zhou, Yanqi Su, Taolue Chen, Zhiqiu Huang, Harald C. Gall, Sebastiano Panichella, “User Review-Based Change File Localization for Mobile Applications.” , IEEE Transactions on Software Engineering, 47 (12) pp. 2755-2770 ,2021

[371] Yuan Huang, Jinyu Jiang, Xiapu Luo, Xiangping Chen, Zibin Zheng, Nan Jia, Gang Huang ,”Change-Patterns Mapping: A Boosting Way for Change Impact Analysis “, IEEE Transactions on Software Engineering, 48 (07) pp, 2376-2398 ,2022

[372] Yuan Huang, Xingjian Liang, Zhihao Chen, Nan Jia, Xiapu Luo, Xiangping Chen, Zibin Zheng, Xiaocong Zhou ,”Reviewing rounds prediction for code patches”, Empirical Software Engineering , 27 (01) pp. 7 ,2022

[373] Yuanrui Fan, Xin Xia, Daniel Alencar da Costa, David Lo, Ahmed E. Hassan, Shanping Li ,”The Impact of Mislabeled Changes by SZZ on Just-in-Time Defect Prediction”, IEEE Transactions on Software Engineering, 47 (08) pp. 1559-1586 ,2021

[374] Yusuf Sulistyo Nugroho, Syful Islam, Keitaro Nakasai, Ifraz Rehman, Hideaki Hata, Raula Gaikovina Kula, MeiyappanNagappan, and Kenichi Matsumoto, How are project-specific forums utilized? A study of participation, content, and sentiment in the Eclipse ecosystem, Empirical Software Engineering, 26(6), 2021

[375] Yutaro Kashiwa, Ryoma Nishikawa, Yasutaka Kamei, Masanari Kondo, Emad Shihab, Ryosuke Sato, and Naoyasu Ubayashi, An empirical study on self-admitted technical debt in modern code review, Information and Software Technology,146,2022

[376] Yutian Tang, Haoyu Wang, Xian Zhan, Xiapu Luo, Yajin Zhou, Hao Zhou, Qiben Yan, Yulei Sui, Jacky Keung ,”A Systematical Study on Application Performance Management Libraries for Apps. “, IEEE Transactions on Software Engineering, 48 (08) pp, 3044-3065,2022

[377] Yuxiang Gao, Yi Zhu, and Yu Zhao, Dealing with imbalanced data for interpretable defect prediction,

[378] Yuyang Liu, Ehsan Noei, and Kelly Lyons, How ReadMe files are structured in open source Java projects, Information and Software Technology, 148,2022

[379] Zehao Lin, Guodun Li, Jingfeng Zhang, Yue Deng, Xiangji Zeng, Yin Zhang, and Yao Wan. 2022. XCode: Towards Cross-Language Code Representation with Large-Scale Pre-Training. ACM Trans. Softw. Eng. Methodol.

[380] Zeinab Azadeh Kermansaravi, Md Saidur Rahman, Foutse Khomh, Fehmi Jaafar, and Yann-Gaël Guéhéneuc, Investigating design anti-pattern and design pattern

mutations and their change- and fault-proneness,Empirical Software Engineering,26 (1), 2021

[381] Zhaoqiang Guo, Shiran Liu, Jinping Liu, Yanhui Li, Lin Chen, Hongmin Lu, and Yuming Zhou. 2021. How Far Have We Progressed in Identifying Self-admitted Technical Debts? A Comprehensive Empirical Study. ACM Trans. Softw. Eng. Methodol.

[382] Zhengliang Li, Zhiwei Jiang, Xiang Chen, Kaibo Cao, and Qing Gu, "Laprob: A Label propagation-Based software bug localization method", Information and Software Technology, Volume 130, 2021

[383] Zhenhao Li, Tse-Hsun Chen, Jinqiu Yang, Weiyi Shang ,"Studying Duplicate Logging Statements and Their Relationships With Code Clones " , IEEE Transactions on Software Engineering, 48 (07) pp,2476-2494,2022

[384] Zhiding Li, Chenqi Shang, Jianjie Wu, and Yuan Li, Microservice extraction based on knowledge graph from monolithic applications, Information and Software Technology,150,2022

[385] Zhifei Chen, Wanwangying Ma, Lin Chen, and Wei Song, Collaboration in software ecosystems: A study of work groups in open environment, Information and Software Technology, 145,2022

[386] Zhipeng Gao, Xin Xia, David Lo, and John Grundy. 2021. Technical Q8A Site Answer Recommendation via Question Boosting. ACM Trans. Softw. Eng. Methodol.

[387] Zhongxin Liu, Xin Xia, David Lo, Zhenchang Xing, Ahmed E. Hassan, Shanping Li , "Which Variables Should I Log?" , IEEE Transactions on Software Engineering, 47 (09) pp. 2012-2031 ,2021

[388] Zhongxing Yu, Chenggang Bai, Lionel Seinturier, Martin Monperrus , "Characterizing the Usage, Evolution and Impact of Java Annotations in Practice", IEEE Transactions on Software Engineering, 47 (05) pp, 969-986 ,2021

[389] Zhou Xu, Li Li, Meng Yan, Jin Liu, Xiapu Luo, John Grundy, Yifeng Zhang, Xiaohong Zhang , "A comprehensive comparative study of clustering-based unsupervised defect prediction models." , Journal of Systems and Software, 172 (02) pp. 110862 ,2021

[390] Zhou Xu, Tao Zhang, Jacky Keung, Meng Yan, Xiapu Luo, Xiaohong Zhang, Ling Xu,and Yutian Tang, Feature selection and embedding based cross project

framework for identifying crashing fault residence, Information and Software Technology, Volume 131, 2021

[391] Zhuo Zhang, Yan Lei, Xiaoguang Mao, Meng Yan, Ling Xu,and Xiaohong Zhang, A study of effectiveness of deep learning in locating real faults, Information and Software Technology, Volume 131, 2021

[392] Zi Peng, Tse-Hsun Chen, Jinqiu Yang, “Revisiting Test Impact Analysis in Continuous Testing From the Perspective of Code Dependencies “, IEEE Transactions on Software Engineering, 48 (06) pp.1979-1993 ,2022

[393] Zijian Jiang, Hao Zhong, Na Meng, “Investigating and recommending co-changed entities for JavaScript programs” , Journal of Systems and Software, 180 (10) pp. 111027 ,2021

[394] Ziyi Zhou, Huiqun Yu, Guisheng Fan, Zijie Huang, and Xingguang Yang, Summarizing source code with hierarchical code representation, Information and Software Technology, 143,2022

[395] Zubair Khaliq, Sheikh Umar Farooq, and Dawood Ashraf Khan, A deep learning-based automated framework for functional User Interface testing, Information and Software Technology, 150,2022