30 Years of Software Refactoring Research: A Systematic Literature Review

Chaima Abid, Vahid Alizadeh,Marouane Kessentini, Thiago do Nascimento Ferreira and Danny Dig

Abstract—Due to the growing complexity of software systems, there has been a dramatic increase and industry demand for tools and techniques on software refactoring in the last ten years, defined traditionally as a set of program transformations intended to improve the system design while preserving the behavior. Refactoring studies are expanded beyond code-level restructuring to be applied at different levels (architecture, model, requirements, etc.), adopted in many domains beyond the object-oriented paradigm (cloud computing, mobile, web, etc.), used in industrial settings and considered objectives beyond improving the design to include other non-functional requirements (e.g., improve performance, security, etc.). Thus, challenges to be addressed by refactoring work are, nowadays, beyond code transformation to include, but not limited to, scheduling the opportune time to carry refactoring, recommendations of specific refactoring activities, detection of refactoring opportunities, and testing the correctness of applied refactorings. Therefore, the refactoring research efforts are fragmented over several research communities, various domains, and objectives. To structure the field and existing research results, this paper provides a systematic literature review and analyzes the results of 3183 research papers on refactoring covering the last three decades to offer the most scalable and comprehensive literature review of existing refactoring research studies. Based on this survey, we created a taxonomy to classify the existing research, identified research trends, and highlighted gaps in the literature and avenues for further research.

Index Terms—Refactoring, systematic literature review, program transformation, software quality.

1 Introduction

For decades, code refactoring has been applied in informal ways before it was introduced and properly defined in academic work. The first known use of the term *Refactoring* in the published literature was in an article written by William Opdyke and Ralph Johnson in September 1990 [1]. William Griswold's Ph.D. dissertation [2], published in 1991, is also one of the first major academic works on refactoring functional and procedural programs. The author defined a set of automatable transformations and described their impact on the code structure. One year later, William Opdyke also published his Ph.D. dissertation [3] on the Refactoring of object-oriented programs. In 1999, Martin Fowler published the first book about refactoring that has as title *Improving the Design of Existing Code* [4]. This book popularised the practice of code refactoring, set its fundamentals, and had a high impact on the world of software development. Martin Fowler defined Refactoring in his book as a sequence of small changes - called refactoring operations - made to the internal structure of the code without altering its external behavior. The goal of these refactoring operations is to improve the code readability and reusability as well as reduce its complexity and maintenance costs in the long run. Since then, a lot has changed in the software development world, but one thing has remained the same: The need for Refactoring.

• *Chaima Abid, Vahid Alizadeh, Marouane Kessentini, and Thiago do Nascimento Ferreira are with the department of Computer and Information Science, University of Michigan, Dearborn, MI, USA.*
Danny Dig is with the Computer Science department, University of Colorado, Boulder, CO, USA.
E-mail: marouane@umich.edu

Manuscript received on June 2020.

Nearly 30 years later, Refactoring has become a crucial part of software development practice, especially with the ever-changing landscape of IT and user requirements. It is a core element of agile methodologies, and most professional IDEs include refactoring tools. Recent studies show that restructuring software systems may reduce developers' time by over 60% [5]. Others demonstrate how Refactoring can help detect, fix, and reduce software bugs [6]. Companies are becoming more and more aware of the importance of Refactoring, and they encourage their developers to continuously refactor their code to set a clean foundation for future updates.

It might be difficult for a developer to be justified to spend time on improving a piece of code to have the same functionality. However, it can be seen as an investment for future developments. Specifically, Refactoring is a crucial task on software with longer lifespans with multiple developers need to read and understand the codes. Refactoring can improve both the quality of software and the productivity of its developers. Increasing the quality of the software is due to decreasing its complexity at design and source code level caused by refactoring, which is proved by many studies [7], [8]. The long-term effect of Refactoring is improving developers' productivity by increasing two crucial factors, understandability and maintainability of the codes, especially when a new developer joins an existing project. It is shown that Refactoring can help to detect, fix, and reduce software bugs and leading to software projects which are less likely to expose bug in development process [6]. Another study claims that there are some specific kinds of refactoring methods that are very probable to induce bug fixes [9].

1.1 Problem Description and Motivation

Refactoring is among the fastest-growing software engineering research areas, if not the fastest. Figure 1 shows the distribution of publications related to refactoring across the globe. Figure 2 reflects the number of publications in the top 10 most active countries in the field of Refactoring. The United States tops the list of countries with a total of 714 publications followed by Germany and Canada with a total of 317 and 248 publications, respectively. During the past 4 years, the number of published refactoring studies has increased with an average of 37% in all top 10 countries. This demonstrates a noticeable increase in interest/need in Refactoring.

Over 5584 authors from all over the world contributed to the field of Refactoring. We highlight the most active authors in Figure 3 and 4, based on both the number of publications and citations in the area. Many scholars started research in the refactoring filed prior to 2000. Others are relatively new to the field and started their contributions after year 2010. All top 10 authors in the field have a constantly increasing number of publications over the past 20 years. Marouane Kessentini heads the list with a total of 43 publications (51% of them were published during the past five years) followed by Steve Counsell and Danny Dig with a total of 39 and 36 publications, respectively. Marouane kessentini published an average of more than 4 articles per year while all other authors published an average between 1.5 and 2.75 publications per year. Figure 5 is a histogram showing how many publications were issued each year starting from 1990. The number of published journal articles, conference papers, and books has increased dramatically during the last decade, reaching a pick of 265 publications in 2016. During just the last four years (2016-2019), over 1026 papers were published in the field, with an average of 256 papers each year.

Recently, several researchers and practitioners have adopted the use of refactoring operations at higher degrees of abstraction than source code level (e.g., databases, Unified Modeling Language (UML) models, Object Constraint Language (OCL) rules, etc.). As a result, they often had to redefine the principles and guidelines of refactoring according to the requirements and specifications of their domains. For instance, in User Interface Refactoring, developers make changes to the UI to retain its semantics and consistency for all users. These refactorings include, but not limited to, *Align entry field, Apply common button size*, *Apply font*, *Indicate format*, and *Increase color contrast*. In Database Refactoring, developers improve the database schema by applying changes such as *Rename column, Split table, Move method*, *Replace LOB with table*, and *Introduce column constraint*. Henceforth, the refactoring operations are called restructuring operations when applied to artifacts other than the ones related to object-oriented programming. Although the different refactoring communities (e.g., software maintenance and evolution, model-driven engineering, formal methods, search-based software engineering, etc.) are interdependent in many ways, they remain disconnected, which may create inconsistencies. For example, when model-level Refactoring does not match the code-level practice, it can lead to incoherence and technical issues during development. The detachment is visible not only between different refactoring domains but also between practitioners and researchers. The distance between them primarily originates from the lack of insights into both worlds' recent findings and needs. For instance, developers tend to use the refactoring features provided by IDEs due to their accessibility and popularity. Most of the time, they are uninformed of the benefits that can be derived from adopting state-of-the-art advances in academia. All these challenges call for a need to identify, critically appraise, and summarize the existing work published across the different domains. Existing systematic literature reviews examine findings in very specific refactoring areas such as identifying the impact of refactoring on quality metrics [10] or code smells [11]. To the best of our knowledge, no work collects and synthesizes existing research, tools, and recent advances made in the refactoring community. This paper is the most comprehensive synthesis of theories and principles of refactoring intended to help researchers and practitioners make quick advances and avoid reinventing or re-implementing research infrastructure from scratch, wasting time and resources. We also build a refactoring infrastructure that will connect researchers with practitioners in industry and provide a bridge between different refactoring communities in order to advance the field of refactoring research.

1.2 Contributions

The Refactoring area is growing very rapidly, and many advances, challenges, and trends have lately emerged. The primary purpose of this study is to implement a systematic literature review (SLR) for the field of refactoring as a whole. This SLR follows a defined protocol to increase the study's validity and rationality so that the output can be high in quality and evidence-based. We used various electronic databases and a large number of articles to comprise all the possible candidate studies and cover more works than existing SLRs.

This SLR contributes to the existing literature in the following ways:

- We identify a set of 3183 studies related to refactoring published until May 2020, fulfilling the quality assessment criteria. These studies can be used by the research and industry communities as a reliable basis and help them conduct further research on Refactoring.
- We present a comprehensive qualitative and quantitative synthesis reflecting the state-of-the-art in refactoring with data extracted from those 3183 high-rigor studies. Our synthesis covers the following themes: artifacts, refactoring tools, different approaches, and performance evaluation in refactoring research.
- We provide guidelines and recommendations based on our findings to support further research in the area.
- We implement a platform that includes the following components: (1) A searchable repository of refactoring publications based on our proposed taxonomy; (2) A searchable repository of authors who contributed to the refactoring community; (3) Analysis

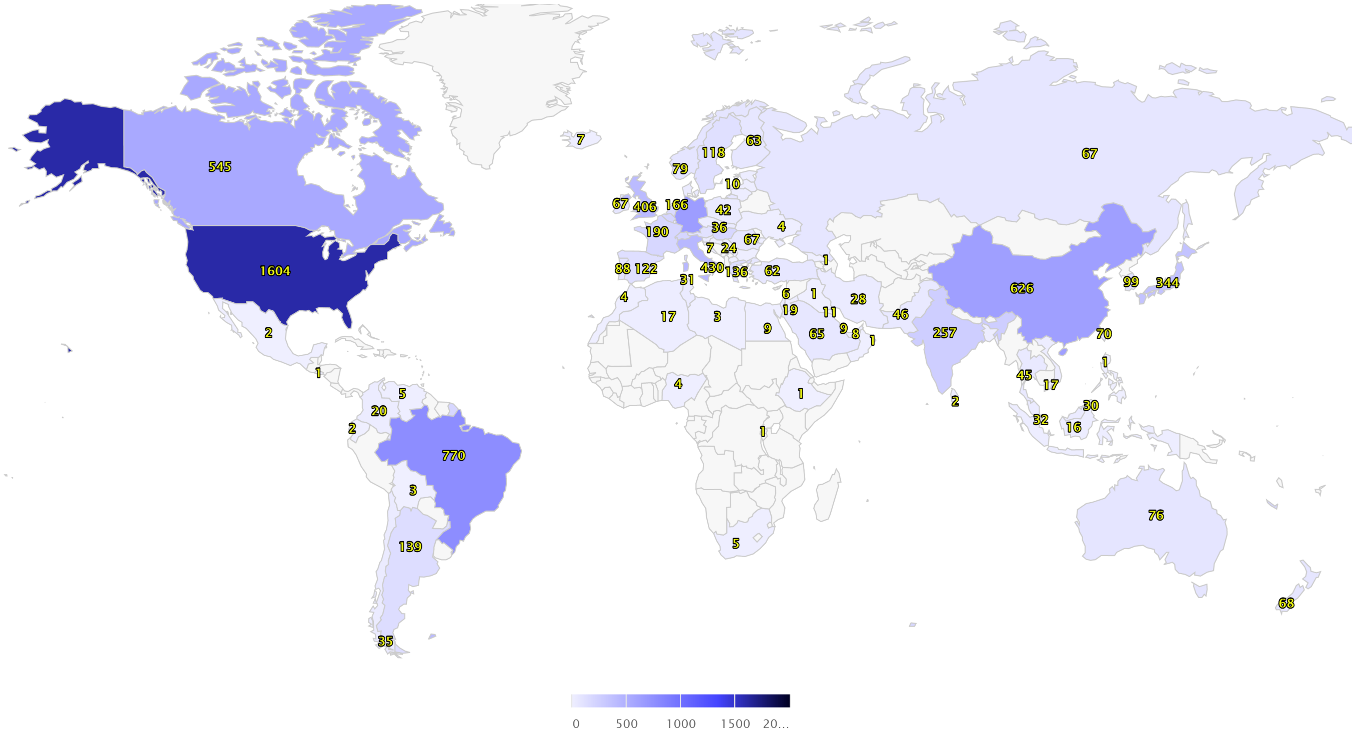


Fig. 1. Distribution of refactoring publications around the world.

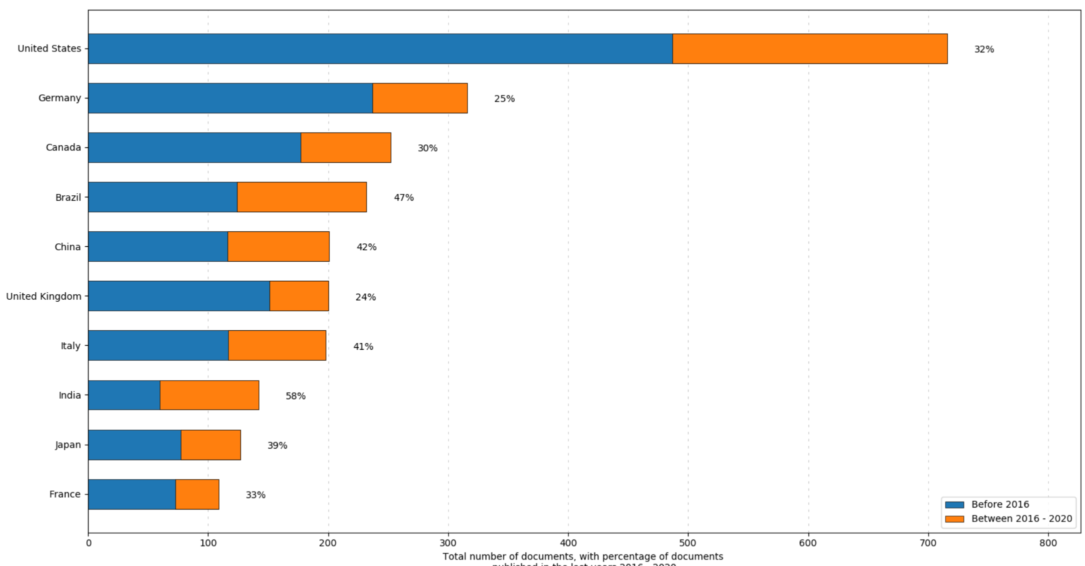


Fig. 2. Number of publications in the top 10 most active countries in the refactoring field

and visualization of the refactoring trends and techniques based on the collected papers. The proposed infrastructure will allow researchers and practitioners to easily report refactoring publications and upload information about active authors in the field of Refactoring. It will also bridge the different communities to advance the field of refactoring research and provide opportunities to educate the next refactoring generation.

1.3 Related Surveys

Mens et al. [12] provided an overview of existing research in the field of software refactoring. They compared and discussed different approaches based on different criteria such as refactoring activities, techniques and formalisms, types of software artifacts that are being refactored, and the effect of refactoring on the software process. Elish et al. [13] proposed a classification of refactoring methods based on their measurable effect on software quality attributes. The investigated software quality attributes are adaptability, completeness, maintainability, understandability, reusabil-

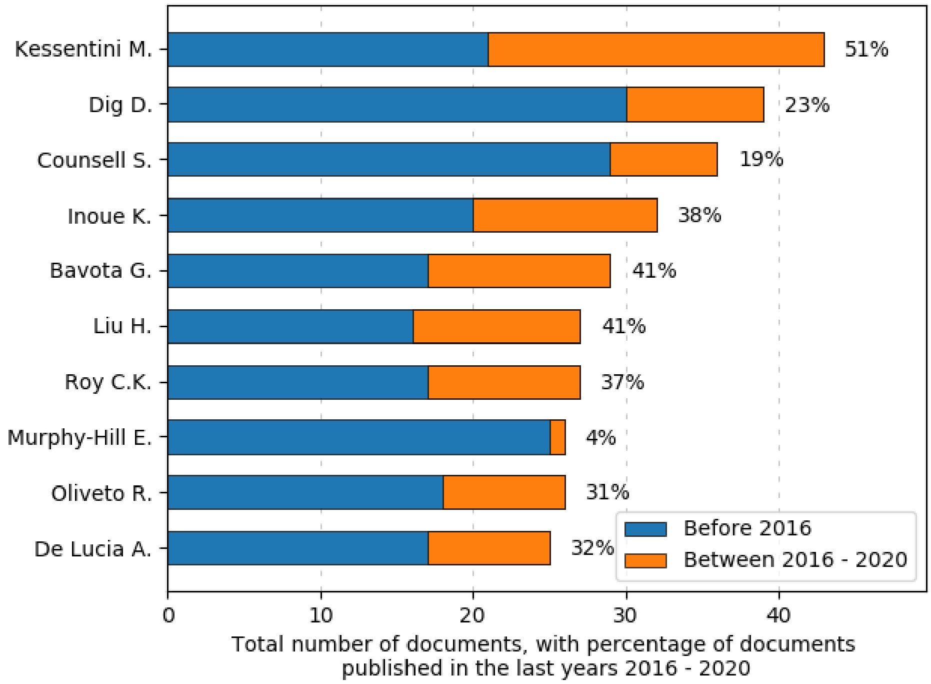


Fig. 3. Top 10 Authors with the highest number of publications and citations in the field of refactoring

ity, and testability. Du Bois et al. [14] provided an overview of the field of software restructuring and Refactoring. They summarized Refactoring's current applications and tool support and discussed the techniques used to implement refactorings, refactoring scalability, dependencies between refactorings, and application of refactorings at higher levels of abstraction. Mens et al. [15] identified emerging trends in refactoring research (e.g., refactoring activities, techniques, tools, processes, etc.), and enumerates a list of open questions, from a practical and theoretical point of views. Misbhauddin et al. [16] provide a systematic overview of existing research in the field of model Refactoring. Al Dallal et al. [17] presented a systematic literature review of existing studies, published through the end of 2013, identifying opportunities for code refactoring activities. In another of their work [10], they presented a systematic literature review that summarizes the impact of refactoring on several internal and external quality attributes. Singh et al. [11] published a systematic literature review of refactoring concerning code smells. However, the review of Refactoring is done in a general manner, and the identification of code smells and antipatterns is performed in-depth. Abebe et al. [18] conducted a study to reveal the trends, opportunities, and challenges of software refactor researches using a systematic literature review. Baqais et al. [19] performed a systematic literature review of papers that suggest, propose, or implement an automated refactoring process.

The different studies mentioned above are mainly about identifying the studies related to very specific or specialized topics. In this paper, we are trying to be as comprehensive as possible by collecting, categorizing, and summarizing all the papers related to refactoring in general that conform to our quality standards.

1.4 Organization

The rest of the paper is organized as follows: First, Section 2 outlines the research method and the underlying protocol for the systematic literature review. Section 3 describes the proposed refactoring infrastructure. The results of this systematic review are reported in Sections 4. Finally, Section 5 presents the conclusions.

2 Research Methodology

Our literature review follows the guidelines established by Kitchenham and Charters [20], which decompose a systematic literature review in software engineering into three stages: planning, conducting, and reporting the review. We have also taken inspiration from recent systematic literature reviews in the fields of empirical software engineering [10] and search-based software engineering [21]. All the steps of our research are well documented, and all the related data are available online for further validation and exploration []. This section details the performed research steps and the protocol of the literature review. First, section 2.1 describes the research questions underlying our survey. Second, section 2.2 details the literature search step. Next, section 2.3 highlights the inclusion and exclusion criteria. The data preprocessing step and our proposed taxonomy are described in sections 2.4 and 2.5, respectively. The quality assessment criteria are defined in section 2.6. Finally, Section 2.7 discusses threats to the validity of our study.

2.1 Research Questions

The following research questions have been derived based on the objectives described in the introduction, which form the basis for the literature review:

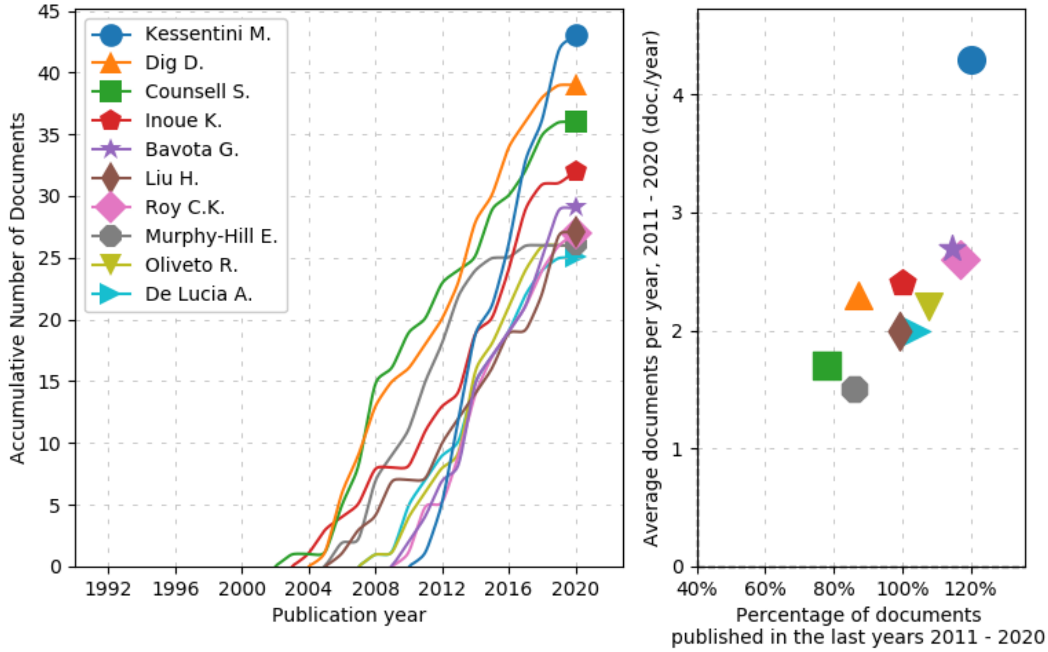


Fig. 4. Evolution of the Top 10 Authors during the past 10 years

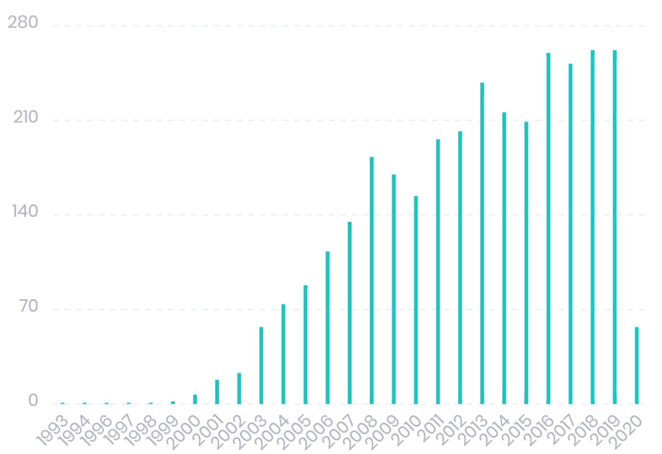


Fig. 5. Trend of publications in the field of refactoring during the last three decades.

- RQ1: What is the refactoring life-cycle?
- RQ2: What are the types of artifacts that are being refactored at each step of the refactoring life-cycle?
- RQ3: Why do software practitioners and researchers perform refactoring?
- RQ4: What are the different approaches used by software practitioners and researchers to perform refactoring?
- RQ5: What types of datasets are used by software practitioners and researchers to validate the refactoring?

2.2 Literature Search Strategy

All the papers have been queried from a wide range of scientific literature sources to make our search as comprehensive as possible:

- **Digital libraries:** ACM Library, IEEE Xplore, ScienceDirect, SpringerLink.
- **Citation databases:** Web of Science (formerly ISI Web of Knowledge), Scopus.
- **Citation search engines:** DBLP, Google Scholar.

We first defined a list of terms covering the variety of both application domains and refactoring techniques. For that, we checked the title, keywords, and abstract of the relevant papers that were already known to us. Synonyms and keywords were derived from this list. These keywords were combined using logical operators ANDs and ORs to create search terms. Before starting collecting the primary studies (PS), we tested the search terms' effectiveness on all the data sources. Then, we refined the queries to avoid getting irrelevant papers. The string adjustments were agreed on by all authors. The final list of search strings are shown in Table 1. These search strings were modified to suit the specific requirements of different electronic databases. We conducted our search on May 31st, 2020, and identified studies published up until that date. The search was done first by the corresponding author and then verified by the rest of the authors. In our systematic review, we followed a multi-stage model to minimize the probability of missing relevant publications as much as possible. The different stages are shown in figure 6 along with the total returned publications at each stage. The first stage consists of executing the search queries on the databases mentioned above; a total of 6158 references were found. Then, we removed the duplicates, which reduced the list of candidate papers to 3882. Then, we performed a manual examination of titles and abstracts to discard irrelevant publications based on the inclusion and exclusion criteria. We also looked at the body of the paper whenever necessary. This decreased the list of candidate papers to 3161 publications. Next, we

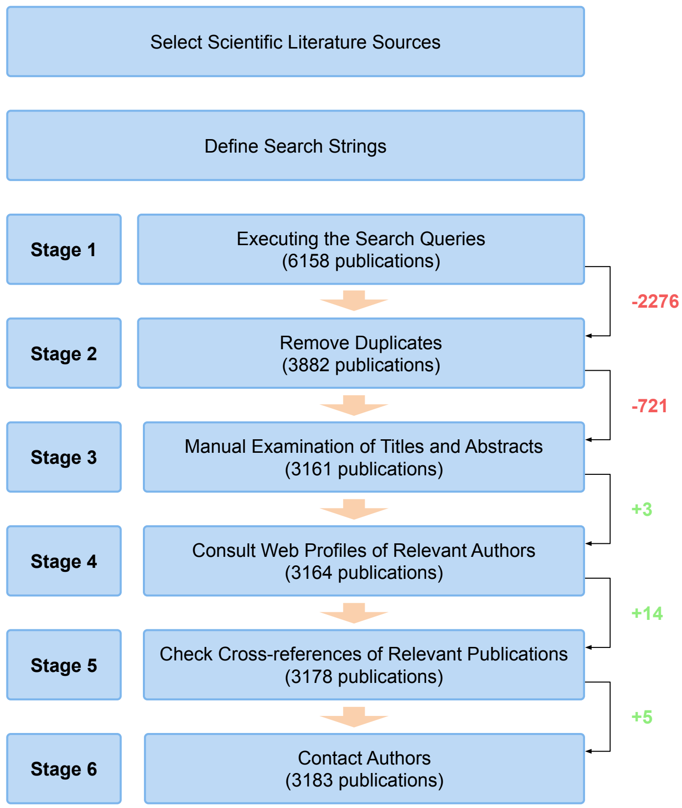


Fig. 6. SLR steps

used the resulting set as input for the snowballing process, recommended by Wohlin [22], to identify additional studies. We consulted web profiles of relevant authors and their networks. We also checked cross-references until no further papers were detected. As a result, 17 new references were added. After that, we contacted the corresponding authors of the identified publications to inquire about any missing relevant studies. This led to adding 5 studies.

2.3 Inclusion and Exclusion Criteria

To filter out the irrelevant articles among those selected in Stage 2 and determine the Primary studies, we considered the following inclusion and exclusion criteria.

2.3.1 Inclusion criteria

All of the following criteria must be satisfied in the selected primary studies:

1) The article must have been published in a peer reviewed journal or conference proceeding between the years 1990 and 2020. The main reason for imposing a constraint over the start year is because the first known use of the term "refactoring" in the published literature was in a September, 1990 article by William Opdyke and Ralph Johnson [1]. We included papers up till May 31st 2020.
2) The article must be related to computer science and engineering and propose techniques, methods and tools for refactoring.
3) The paper must be written in English.
4) In case a conference paper has a journal extension, we would include both the conference and journal publications.
5) The paper must pass the quality assessment criteria that are elaborated in Section 2.6.

2.3.2 Exclusion criteria

Papers satisfying any of the exclusion criteria were discarded, as follows:

1) Studies that are not related to the computer science field.
2) Studies that investigated the impact of general maintenance on code quality. In this case, the maintenance tasks were potentially performed due to several reasons and not limited to refactoring, and therefore, we cannot judge whether the impact was due to refactoring or to other maintenance tasks such as corrective or adaptive maintenance.
3) Grey Literature

2.4 Data Preprocessing

A pre-processing technique was applied to improve reliability and precision, as detailed in the following sub sections.

2.4.1 Simplifying Author's name

In general, scientific and bibliographic databases such as *Web of Science* (WoS) and *Scopus* have the following inconsistencies in authors names:

- Most journals abbreviate the author's first name to an initial and a dot.
- Most journals use the author name's special accents.
- WoS uses a comma between the author's last name and first name initial, but Scopus does not.

These name-related inconsistencies mean that scientometrics scripts cannot find all of the similar author's names. For that reason, ScientoPy script applies the following steps to simplify author's name fields:

- Remove dots and coma from author's name.
- Remove special accents from author's name

2.4.2 Fixing inconsistent country names

Some authors use different naming to refer to the same country (such as USA and United States). For that reason, some country names were replaced based on Table 3.

2.5 Study Classification

According to the research questions listed in Section 2.1, we classified the PSs into five dimensions: (1) refactoring lifecycle (related to RQ1), (2) artifacts affected by refactoring (related to RQ2), (3) refactoring objectives (related to RQ3), (4) refactoring techniques (related to RQ4) and (5) refactoring evaluation (related to RQ5). The determination of the attributes of each dimension was performed incrementally. That is, for each dimension, we started with an empty set of attributes. The authors of this study screened the full texts of the PSs one by one, analyzed each reported study based on the considered dimension, and determined

TABLE 1
final list of search strings

search strings
(software OR system OR code OR service OR diagram OR database OR architecture OR Model OR GUI OR user interface OR UI OR design OR artifact OR developer OR computer OR programming OR object-oriented OR implement OR mobile app OR cloud OR document) AND (refactor OR refactoring)

TABLE 2
PS quality assessment questions [17]

	Question
Design	Are the applied identification techniques for refactoring opportunities clearly described?
	Are the refactoring activities considered clearly stated and defined?
	Was the sample size justified?
	Are the evaluation measures fully defined?
Conduct	Are the data collection methods adequately described?
Analysis	Are the results of applying the identification techniques evaluated?
	Are the data sets adequately described? (size, programming languages, source)
	Are the study participants or observational units adequately described?
	Are the statistical methods described?
	Are the statistical methods justified?
	Is the purpose of the analysis clear?
	Are the scoring systems (performance evaluation) described?
Conclusion	Are all study questions answered?
	Are negative findings presented?
	Are the results compared with previous reports?
	Do the results add to the literature?
	Are validity threats discussed?

TABLE 3
List of countries and their replacements

Country	Replacement
Republic of China	China
USA	United States
England, Scotland and Wales	England
U Arab Emirates	United Arab Emirates
Russia	Russian Federation
Viet Nam	Vietnam
Trinid & Tobago	Trinidad and Tobago

the attributes of that dimension as considered by each PS. Table 4 outlines the keywords extracted for each category. It should be pointed out that, most of the time, we remove all of the affixes (i.e., suffixes, prefixes, etc.) attached to a word in order to keep its lexical base, also known as root or stem or its dictionary form or lemma. For instance, the word *document* allows us to detect the words *documentation* and *documenting*. Also, we did not include bi-grams and tri-grams that can be detected using one uni-gram. For example, *Class Diagram, Object Diagram, Sequence Diagram*, and *Use Case Diagram* can all be detected using the word *Diagram* alone.

The screening of the PSs resulted in determining six stages for the refactoring life-cycle (e.g., detection, prioritization, recommendation, testing, documentation, and prediction). We also classified the papers according to the level of automation of the proposed technique (e.g., automatic, manual, semi-automatic). The results are described in section 4.1. For the second dimension, we identified five artifacts on which the impact of refactoring is studied by at least one of the PSs. These artifacts are code, architecture, model, GUI, and database. The classification of PSs based on these artifacts is discussed in detail in Section 4.2. We subdivided the third dimension into five categories (e.g., External quality, internal quality, performance, migration, and security) to reflect the refactoring objective and six categories (e.g., Object-oriented design, Aspect-oriented design, Model-driven engineering, Documentation, Mobile development, and Cloud computing) to describe the refactoring paradigms. The classification of PSs based on these categories is discussed in detail in Section 4.3. We divided the fourth dimension into four categories (e.g., data mining, search-based algorithms, formal methods, and fuzzy logic) to reveal the refactoring techniques adopted in the studies and into twelve categories (e.g., Java, C, C#, Python, Cobol, PHP, Scala, Smalltalk, Ruby, Javascript, MATLAB, and CSS) to show the most common programming languages used in our PSs. The details of this categorization are reported in section 4.4. Finally, for the fifth dimension, we divide the PSs into two categories: open-source and industrial. The open-source category includes studies that validate their approaches using open source systems. In contrast, the industrial category consists of the studies that validate their work on systems of their industrial collaborators. These findings are outlined in Section 4.5.

2.6 Study Quality Assessment

To ensure a level of quality of papers, we only included venues that are known for publishing high-quality software engineering research in general with an h-index of at least 10, as has been done by [23] . Each of the papers that were published before 2019 has to be cited at least once. The quality of each primary study was assessed based on a quality checklist defined by Kitchenham and Charters [20]. This step aims to extract the primary studies with information suitable for analysis and answering the defined research questions. The quality checklist, (described in table 2) were defined by Galster et al. [23]. They are developed

TABLE 4
List of keywords used to detect the different categories

Category	Keywords
Refactoring life-cycle (RQ1)	
Detection	detect, opportunity, smell, antipattern, design defect
Prioritization	schedul, sequence, priorit
Recommendation	recommend, correction, correcting, fixing, suggest
Testing	test, regression testing, test case, unit test
Documentation	document
Prediction	predict, future release, next release, development history, refactoring history
Level of automation (RQ1)	
Manual	manual
Semi-automatic	semi-automat, semi-manual
Automatic	automat
Artifact (RQ2)	
Code	code, java, object orient, smell, antipattern, anti-pattern, object-orient
Model	design, model, UML, diagram, Unified Modeling Language
Architecture	architecture, hotspot, hierarchy
GUI	gui, user interface, UI
Database	relational, schema, database, Structured Query Language, SQL
Paradigm (RQ3)	
Object-oriented design	object orient, object-orient, oo, java, c, ++, python, C sharp, c#, css, Python, R, PHP, JavaScript, Ruby, Perl, Object Pascal, Objective-C, Dart, Swift, Scala, Kotlin, Common Lisp, MATLAB, Smalltalk
Aspect-oriented design	aspect
Model-driven engineering	model transform, uml, reverse engineering, diagram, Unified Modeling Language
Documentation	document
Mobile development	android, mobile, IOS, phone, smartphone, cellphones
Could computing	web service, wsdl, restful, cloud, Apache Hadoop, Docker, Middleware, Software-as-a-Service, SaaS, XaaS, Anything-as-a-Service, Platform-as-a-Service, PaaS, Infrastructure-as-a-Service, IaaS, AWS, Amazon EC2, Amazon Simple Storage Service, S3
Refactoring Objectives (RQ3)	
Internal Quality	maintainability, cyclomatic, depth of inheritance, coupling, quality, Flexibility, Portability, Re-usability, Readability, Testability, Understandability
Performance	performance, parallel, Response Time, Error Rates, Request Rate, availability
External quality	analysability, changeability, time behaviour, resource, Correctness, Usability, Efficiency, Reliability, Integrity, Adaptability, Accuracy, Robustness
Migration	migrat
Security	secure, safety, Attack surface, virus, hack, vulnerability, vulnerable, spam
Programming languages (RQ4)	
Java	java
C	c, c++
C#	c sharp, c#
Python	python
CSS	css
PHP	php
Cobol	cobol
Scala	scala
Javascript	javascript
Ruby	ruby
Smalltalk	smalltalk
MATLAB	matlab
Adopted methods (RQ4)	
Search-based algorithms	search, search-base, sbse, genetic, fitness, simulated annealing, tabu search, search space, Hill climbing, Multi-objective evolutionary algorithms, multi objective optimization, multi-objective programming, vector optimization, multi-criteria optimization, multi-attribute optimization, Pareto optimization, Evolutionary Multi-objective Optimization, EMO, Single-Objective Optimization, Many-Objective Optimization, multi objective
Data mining	artificial intelligence, ai , machine learning, naive bayes, decision tree, SVM, support vector machine, Cluster, Classification, classify, Association, Neural networks, deep learning, random forest, regression, reinforcement learning, learning
Formal methods	model check, formal method, B-Method, RAISE, Z notation, SPARK Ada
Fuzzy logic	fuzzy
Evaluation method (RQ5)	
Open source	open source, open-source
Industrial	proprietary, industrial, industry, collaborator, collaboration

by considering bias and validity problems that can occur at different stages, including the study design, conduct, analysis, and conclusion. Each question is answered by a ”Yes”, ”Partially”, or ”No”, which correspond to a score of 1, 0.5, or 0, respectively. If a question does not apply to a study, we do not evaluate the study for that question. The quality assessment checklist was independently applied to all 3882 studies by two of the authors. All disagreements on the quality assessment results were discussed, and a consensus was reached eventually. Few cases where agreement could not be reached were sent to the third author for further investigation. 154 studies did not meet the quality assessment criteria.

2.7 Threats to Validity

Several limitations may affect the generalizability and the interpretations of our results. The first is the possibility of paper selection bias. To ensure that the studies were selected in an unbiased manner, we followed the well-defined research protocol and guidelines reported by Kitchenham and Charters [20] instead of proposing nonstandard quality factors. Also, the final decision on the articles with selection disagreements was performed based on consensus meetings. The Primary studies were assessed by one researcher and checked by the other, a technique applied in similar studies [21]. The second threat consists of missing a relevant study. To overcome this threat, we employed several strategies that we mentioned in Section 2.2. Few related studies were detected after performing the automatic search, which indicates that the constructed search strings and the mentioned utilized libraries were comprehensive enough to identify most of the relevant articles. Another critical issue is whether our taxonomy is complete and robust sufficient to analyze and classify the primary studies. To overcome this problem, we used an iterative content analysis method by going through the papers one by one and continuously expand the taxonomy for every new encountered concept. Furthermore, to gather sufficient keywords to detect the different categories, we followed the same iterative process, and we added synonyms based on the authors' expertise in the field of refactoring. Another threat is related to the tagging of the papers according to our taxonomy. To mitigate this problem, we asked 27 graduate students to check the correctness of the classification results by reading the abstract, the title, and keywords. They also check the body of the paper whenever necessary.

3 Refactoring Infrastructure

We implemented a large scale platform [24] that collects, manages, and analyzes refactoring related papers to help researchers and practitioners share, report, and discover the latest advancements in software refactoring research. It includes the following components:

1) **A searchable repository of refactoring publications based on our proposed taxonomy**. Figure 9 shows a screenshot of the publications' tab of the refactoring repository website. The papers can be searched by author, title, or year of publication. Each paper has tags that describe its content based on our taxonomy described in section 2.5. The papers can also be filtered using those tags and sorted alphabetically or chronologically according to the title and year of publication, respectively. The user can export the publications' dataset to many formats, including pdf, excel, and CSV. He can also easily report a new publication by entering its link.
2) **A searchable repository of authors who contributed to the refactoring community**. Figure 8 shows a screenshot of the authors' tab of the refactoring repository website. The authors can be searched and sorted alphabetically by name, affiliation, or country. They can also be sorted based on the total number of refactoring publications. The user can also consult the *Google Scholar* and *Scopus* profiles of the authors if available. Finally, the user can easily report a new author by entering their information and their profile. Furthermore, we defined the refactoring h-index, which shows how many papers about refactoring published by the author have been cited proportionately. A refactoring h-index of X means that the author has X papers about refactoring that have been cited at least X times. Authors can also be sorted according to the refactoring h-index and the total number of citations (see figure 11). Besides, we created a co-author network and corresponding visualizations (see figure 12) to get a snapshot view of the breadth and depth of an individual's collaborations in the field of refactoring research. Finally, we generated a histogram (see figure 7) that shows the number of publications issued by the top institutions active in the refactoring research by considering the authors' affiliations.
3) **Analysis and visualization of the refactoring trends and techniques based on the collected papers**. Figure 10 shows a screenshot of the refactoring repository dashboard. It contains histograms and pie charts that show the distribution and percentages of the categories defined in our taxonomy. It also includes maps that reflect the spread of refactoring activity across the world.

The proposed infrastructure will enable researchers to perform a fair comparison between their new refactoring approaches and state-of-the-art tools; enable researchers to use refactoring data of large software systems; facilitate interactions between researchers from currently disconnected domains/communities of refactoring (model-driven engineering, service computing, parallelism and performance optimization, software quality, testing, etc.); enable practitioners and researchers to quickly identify relevant existing research papers and tools for their problems based on the proposed taxonomy and classification; create benchmarks against which various refactoring approaches can be evaluated; enable effective interactions between practitioners and refactoring researchers to identify relevant problems faced by the software industry.

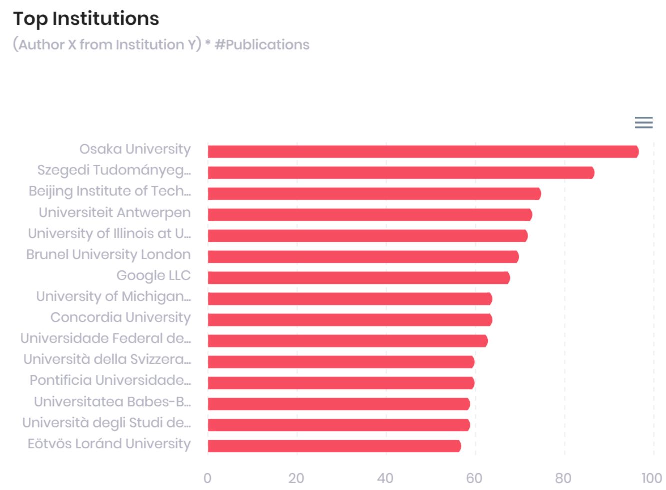


Fig. 7. Top institutions active in the refactoring field

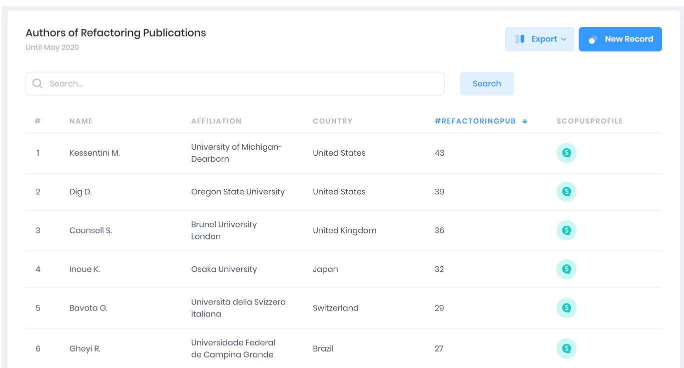


Fig. 8. A screenshot of the authors tab of the refactoring repository Website

4 Results

In this section, we aim to answer the research questions. To provide an overview of the current state of the art in refactoring and guide the reader to a specific set of approaches, tools, and recent advances that are of interest, we classified the 3183 reviewed papers based on the taxonomy described in Section 2.5. Table 5 contains representative references for the categories created for each RQ. We only provided 10 references per category because we cannot possibly report in this paper the categorization of all the studies since we are dealing with a total of 3183 papers. The results of the classification of all the papers are provided in our website [24]. For some taxonomy categories, papers may have multiple values and thus be listed several times. As a result, percentages in the tables may sum up to more than 100 percent. Also, not all the papers were classified in all dimensions. Consequently, percentages in one dimension may not sum up to 100 percent. The rest of this section presents the observations and insights that can be derived from the visualization of the categories.

4.1 Refactoring life-cycle

Going through the primary studies, we have been able to establish a refactoring life-cycle that is composed of six

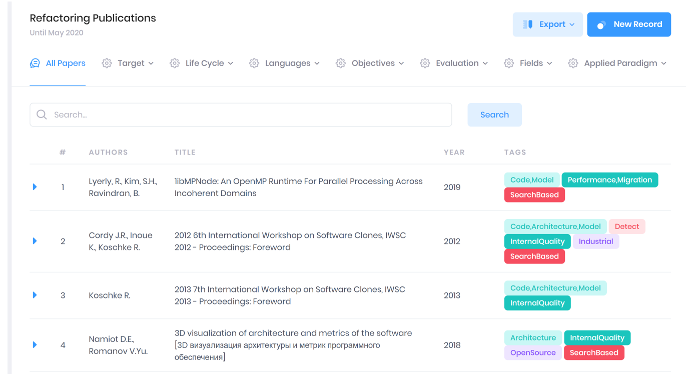


Fig. 9. A screenshot of the publications tab of the refactoring repository Website

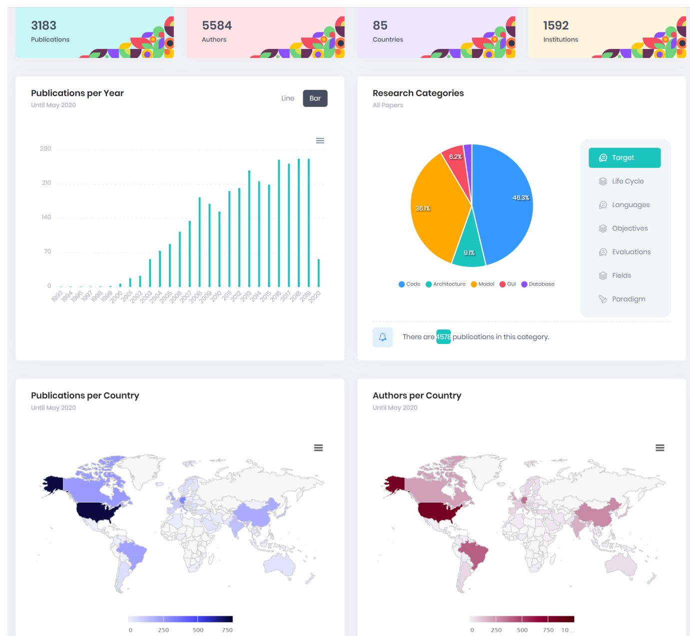


Fig. 10. A screenshot of the Dashboard of the refactoring repository website

stages:

- **Refactoring detection**: Identifying refactoring opportunities is an important stage that precedes the actual refactoring process. It can be done by manually inspecting and analyzing an artifact of a system to identify refactoring opportunities. However, this technique is time-consuming and costly. Researchers in this area typically propose fully or semi-automated techniques to identify refactoring opportunities. These techniques may be applicable to different artifacts and should be evaluated empirically.
- **Refactoring prioritization**: The number of refactoring opportunities usually exceeds the amount of problems that the developer can deal with, particularly when the effort available for performing refactorings is limited. Moreover, not all refactoring opportunities are equally relevant to the goals of the system or its health. In this stage, the refactorings operations are prioritized using different criteria (e.g., maximizing the refactoring of classes with a large

TABLE 5
Representative references for all categories

Category	Percentage	Papers
Refactoring life-cycle (RQ1)		
Detection	28.65%	[S1], [S2], [S3], [S4], [S5], [S6], [S7], [S8], [S9], [S10]
Prioritization	9.43%	[S11], [S12], [S13], [S14], [S15], [S16], [S17], [S18], [S19], [S20]
Recommendation	16.18%	[S3], [S11], [S12], [S21], [S22], [S23], [S24], [S25], [S26], [S27]
Testing	18.44%	[S4], [S6], [S7], [S8], [S13], [S28], [S29], [S30], [S31], [S32]
Documention	5.22%	[S33], [S34], [S35], [S36], [S37], [S38], [S39], [S40], [S41], [S42], [S43]
Prediction	4.818%	[S44], [S45], [S46], [S47], [S48], [S49], [S50], [S51], [S52], [S53]
Level of automation (RQ1)		
Automatic	30.95%	[S54], [S55], [S56], [S57], [S58], [S59], [S60], [S61], [S62], [S63]
Semi-automatic	1.95%	[S64], [S65], [S66], [S67], [S68], [S69], [S70], [S71], [S72], [S73], [S74], [S75]
Manual	8.67%	[S69], [S76], [S77], [S78], [S79], [S80], [S81], [S82], [S83], [S84]
Artifact (RQ2)		
Code	72.89%	[S1], [S2], [S3], [S11], [S65], [S85], [S86], [S87], [S88], [S89]
Model	59.25%	[S1], [S3], [S28], [S29], [S65], [S87], [S89], [S90], [S91], [S92]
Architecture	17.25%	[S28], [S91], [S93], [S94], [S95], [S96], [S97], [S98], [S99], [S100]
GUI	2.58%	[S6], [S8], [S28], [S87], [S89], [S90], [S101], [S102], [S103], [S104]
Database	4.12%	[S27], [S36], [S65], [S100], [S105], [S106], [S107], [S108], [S109], [S110]
Paradigm (RQ3)		
Object-oriented design	34.09%	[S1], [S8], [S30], [S85], [S87], [S88], [S101], [S111], [S112], [S113]
Aspect-oriented	10.87%	[S88], [S96], [S101], [S102], [S103], [S114], [S115], [S116], [S117], [S118]
Model-driven engineering	7.35%	[S3], [S15], [S32], [S58], [S65], [S119], [S120], [S121], [S122], [S123]
Mobile apps development	3.55%	[S23], [S87], [S87], [S95], [S99], [S112], [S124], [S125], [S126], [S127]
Could computing	4.15%	[S128], [S129], [S130], [S131], [S132], [S133], [S134], [S135], [S136], [S137]
Refactoring Objective (RQ3)		
Internal Quality	41.63%	[S3], [S12], [S21], [S29], [S30], [S89], [S90], [S94], [S138], [S139]
Performance	15.93%	[S10], [S12], [S28], [S86], [S88], [S91], [S92], [S96], [S115], [S119]
External quality	22.68%	[S87], [S91], [S92], [S95], [S102], [S140], [S141], [S142], [S143], [S144]
Migration	3.61%	[S95], [S100], [S113], [S145], [S146], [S147], [S148], [S149], [S150], [S151]
Security	3.11%	[S113], [S152], [S153], [S154], [S155], [S156], [S157], [S158], [S159], [S160]
Programming language (RQ4)		
Java	17.15%	[S1], [S8], [S10], [S30], [S85], [S87], [S88], [S112], [S113], [S140]
C	4.65%	[S59], [S96], [S104], [S105], [S111], [S146], [S161], [S162], [S163], [S164]
C#	0.66%	[S61], [S165], [S166], [S167], [S168], [S169], [S170], [S171], [S172], [S173]
Python	0.53%	[S174], [S175], [S176], [S177], [S178], [S179], [S180], [S181], [S182], [S183]
CSS	0.5%	[S147], [S184], [S185], [S186], [S187], [S188], [S189], [S190], [S191], [S192]
PHP	0.35%	[S169], [S193], [S194], [S195], [S196], [S197], [S198], [S199], [S200], [S201]
Cobol	0.31%	[12], [S202], [S203], [S205], [S206], [S207], [S208], [S209]
MATLAB	0.28%	[S210], [S211], [S212], [S213], [S214], [S215], [S216], [S217]
Smalltalk	0.79%	[25], [S219], [S220], [S221], [S222], [S223], [S224], [S225], [S226], [S227]
Ruby	0.22%	[S169], [S181], [S228], [S229], [S230], [S231]
Javascript	0.72%	[S112], [S232], [S233], [S234], [S235], [S236], [S237], [S238], [S239], [S240], [S241]
Scala	4.02%	[S33], [S55], [S86], [S126], [S242], [S243], [S244], [S245], [S246], [S247]
Adopted Method (RQ4)		
Search-based algorithms	25.76%	[S12], [S248], [S249], [S250], [S251], [S252], [S253], [S254], [S255], [S256]
Data mining	15.49%	[S2], [S82], [S107], [S185], [S257], [S258], [S259], [S260], [S261], [S262]
Formal methods	2.92%	[S42], [S199], [S263], [S264], [S265], [S266], [S267], [S268], [S269]
Fuzzy logic	0.28%	[S257], [S270], [S271], [S272], [S273], [S273], [S274]
Evaluation method (RQ5)		
Open source	16.31%	[S1], [S7], [S12], [S30], [S32], [S88], [S112], [S139], [S248], [S275]
Industrial	10.4%	[S9], [S12], [S16], [S115], [S120], [S147], [S276], [S277], [S278], [S279]

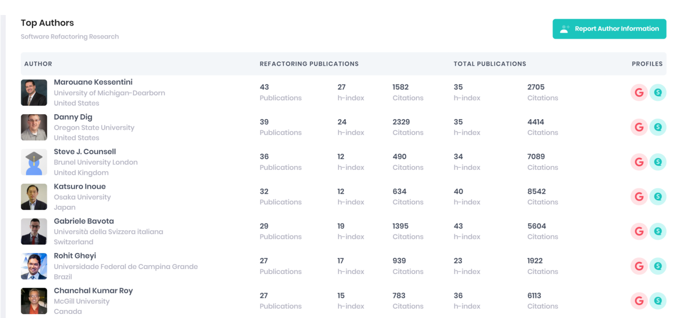


Fig. 11. A screenshot of the refactoring repository dashboard that shows the authors, their h-index and total number of publications and citations

number of anti-patterns or with the previous history of bugs, etc.) according to the needs of developers.

- **Refactoring recommendation**: Several refactoring recommendation tools have been proposed that dynamically adapt and suggest refactorings to developers. The output is sequences of refactorings that developers can apply to improve the quality of systems by fixing, for example, code smells or optimizing security metrics.
- **Refactoring testing**: After choosing the refactorings to be applied, tests need to be done to ensure the correctness of artifacts transformations and avoid future bugs. This is done by checking the satisfaction of the pre-and post-conditions of the refactoring operations and the preservation of the system behavior.
- **Refactoring documentation**: After applying and testing the refactorings, we need to document the refac-

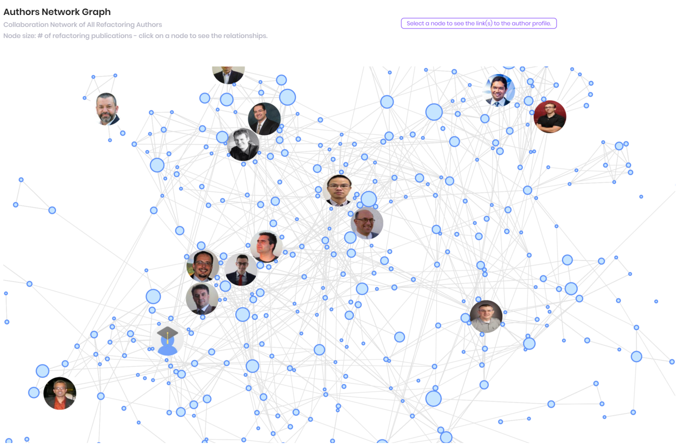


Fig. 12. A screenshot of the authors network graph from the refactoring repository website

torings, their locations, why they have been applied, and the quality improvements.

- **Prediction**: It is interesting for developers to know which locations are likely to demand refactoring in future releases of their software products. This will help them focus on the relevant artifacts that will undergo changes in the future, prepare them for further improvements and extensions of functionality, and optimize the management of limited resources and time. Predicting locations of future refactoring can be done using the development history.

Figure 13 illustrates the percentage of the papers related to each stage of the refactoring life-cycle. 33.08% of the papers deal with testing. Researchers have invested heavily in testing to ensure the reliability of refactoring because changing the structure of code can easily introduce bugs in the program and lead to challenging debugging sessions. A plenty of effort is made towards the automation of the testing process to facilitate the adoption of refactoring [S54], [S55], [S56]. Detecting refactoring opportunities is also a topic of interest to researchers. Several approaches have been proposed to detect refactoring opportunities including but not limited to techniques that depend on quality metrics (e.g., cohesion, coupling, lines of code, etc.), code smells (e.g., feature envy, Blob class, etc.), Clustering (similarities between one method and other methods, distances between the methods and attributes, etc.), Graphs (e.g., represent the dependencies among classes, relations between methods and attributes, etc.), and Dynamic analysis (e.g., analyzing method traces, etc.). Refactoring documentation is an underexplored area of research. Only 5.22% of the collected papers dived into refactoring documentation. Many studies examined the automation of the different refactoring stages to reduce the refactoring effort and, therefore, increase its adaption. Figure 14 shows the count of publications dealing with manual, semi-automatic, and automated refactoring. In fact, 30.95% of the papers deal with the automation of refactoring. Only 1.95% and 8.67% of the papers used manual and semi-automatic refactoring, respectively.

4.2 Artifacts affected by refactoring

As we mentioned before, refactoring is not limited to software code. In fact, it can be applied to any type of software artifacts (e.g., software architectures, database schema, models, user interfaces, and code). Figure 15 shows the percentage of refactoring publications per artifact. The evidence from this histogram shows that the most popular refactoring artifact is code (72.89%). Model refactoring has also received considerable attention, with a percentage of 59.25%. Graphical user interfaces (GUIs) and Database refactoring have received the least attention of all with a fraction of only 4.12% and 2.58%, respectively. This might be due to the fact that database refactoring is conceptually more difficult than code refactoring; code refactorings only need to maintain behavioral semantics while database refactorings also must maintain informational semantics. Also, GUI refactoring is very demanding, requiring the adoption of user interfaces architectural patterns from the early software design stages. Future research should explore database and user interface refactoring further as they are an indispensable part of today's software.

4.3 Refactoring objectives

Five paradigms have been identified from analyzing the primary studies: object-oriented designs, cloud computing, mobile apps, model-driven, and aspect-oriented. Object-oriented programming has gained popularity because it matches the way people actually think in the real world, structuring their code into meaningful objects with relationships that are obvious and intuitive. The increased popularity of the object-oriented paradigm has also increased the interest in object-oriented refactoring. This can be observed in figure 16 where more than 34% of the studies related to refactoring focus on object-oriented designs. Less than 5% of the papers investigated refactoring for cloud computing and mobile app development. For the refactoring objectives classification of the taxonomy, five subcategories are considered: external quality (e.g. correctness, usability, efficiency, reliability, etc.) , internal quality (e. g. maintainability, flexibility, portability, re-usability, readability etc.) , performance (e.g. response time, error rate, request rate, memory use, etc.), migration (e.g. Dispersion in the Class Hierarchy, number of referenced variables, number of assigned variables etc.), security (e.g. time needed to resolve vulnerabilities, Number of viruses and spams blocked, Number of port probes, number of patches applied, Cost per defect, Attack surface etc.). Figure 17 is illustrating the reasons why people refactor their systems. Improving the internal quality takes up the largest portion (41.63%) followed by refactoring to improve the external quality (22.68%). Although security is a major concern for almost all systems, only 3.11% of the papers investigated refactorings for security reasons.

4.4 Refactoring techniques

Object-oriented programming languages have common traits/properties that facilitate the development of widely automated source code analysis and transformation tools. Many studies [25] have given sufficient proof that a refactoring tool can be built for almost any object-oriented language (Python, PHP, Java, and C++). Support for multiple

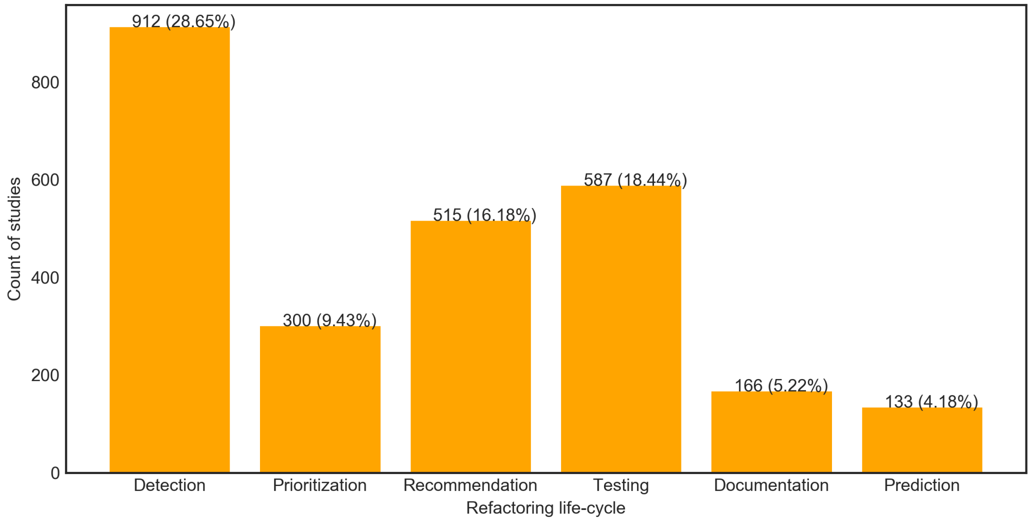


Fig. 13. Histogram illustrating the percentage of refactoring publications per refactoring life-cycle

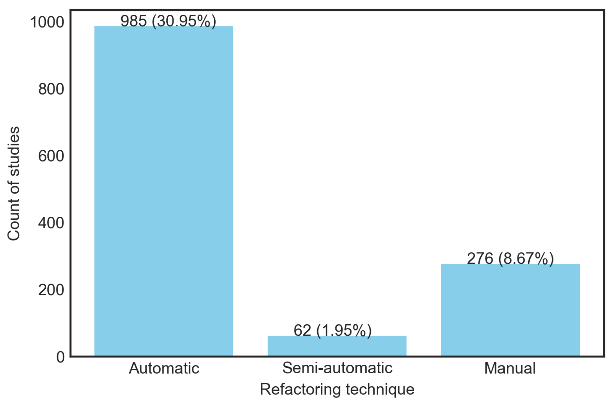


Fig. 14. Histogram illustrating the percentage of publications dealing with manual, semi-automatic and automated refactoring

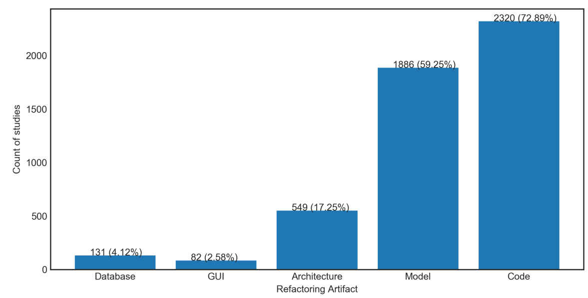


Fig. 15. histogram illustrating the count of refactoring publications per artifact

languages in a refactoring tool is mentioned by [26]. Java is probably the most commercially important recent object-oriented language with an infrastructure that is designed to support analysis. It has generic parsing, tree building, prettyprinting, tree manipulation, source-to-source rewriting, attribute grammar evaluations, control, and data flow analysis. This explains the fact that 17.15% of refactoring studies (see figure 18) provided refactoring techniques and tools that support Java. At the same time, most of the other programming languages have a fraction of less than 1%. We classified the refactoring techniques into four main categories: data mining (e.g., Clustering, Classification, Decision trees, Association, Neural networks, etc.), search-based methods (e.g., Genetic algorithms, Hill climbing, Simulated annealing, Multi-objective evolutionary algorithms, etc.), formal methods (B-Method, the specification languages used in automated theorem proving, RAISE, the Z notation, SPARK Ada, etc.), and fuzzy logic. More than 25% of the papers use Search-based techniques to address refactoring problems (see figure 19). This can be explained by the fact that search-based approaches have been proven to be efficient at finding solutions for complex and labor-intensive tasks. With the growing complexity of software systems, there's an infinite amount of improvement/changes you can make to any piece of artifact. Exact algorithms are hard to use to solve the refactoring problem within an instance-dependent, finite run-time. That's why finding optimal refactoring solutions are sacrificed for the sake of getting perfect solutions in polynomial time using heuristic methods like search-based algorithms. Data mining techniques have also received significant attention (17.59%) as they are known to be efficient at discovering new information, such as unknown patterns or hidden relationships, from huge databases like, for our case, large code repositories.

4.5 Refactoring evaluation

Open-source software systems are becoming increasingly important these days. 61.1% of the studies (see figure 20)

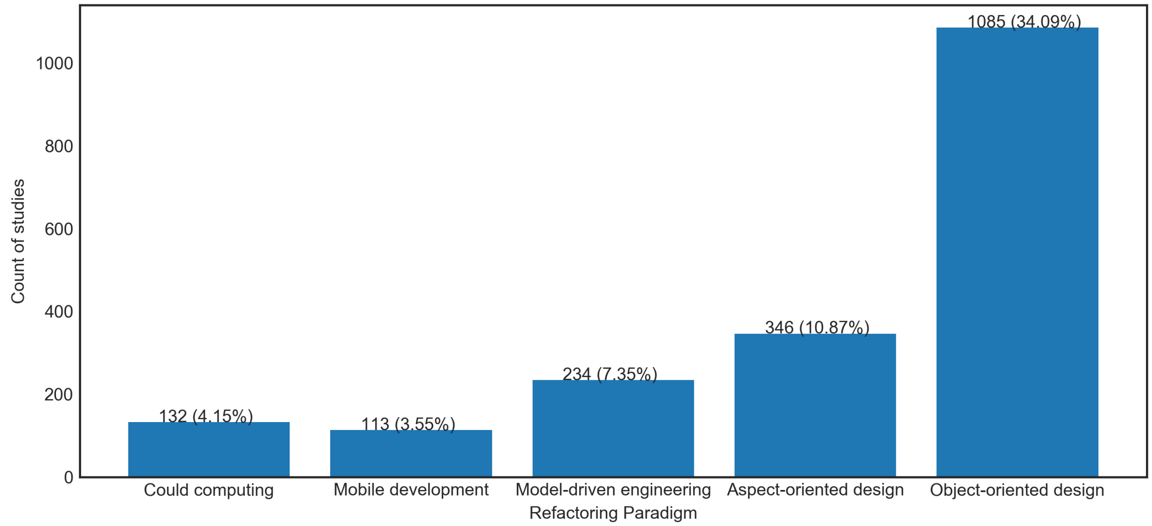


Fig. 16. Histogram illustrating the count of refactoring publications per paradigm

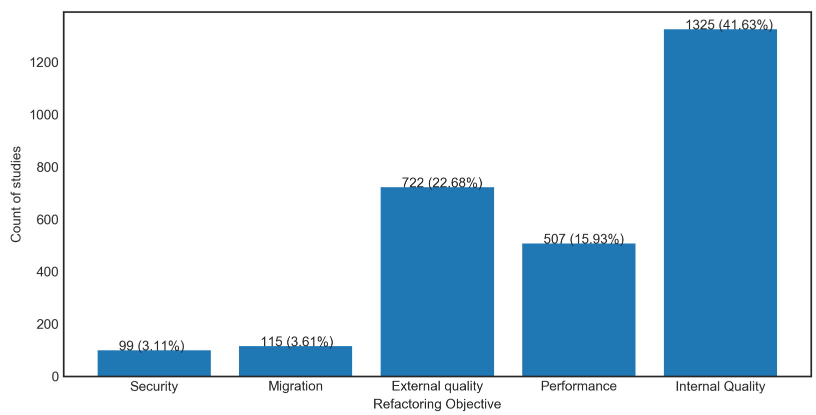


Fig. 17. Histogram illustrating the count of publications per refactoring objective

used open-source systems to validate their work compared to 38.9% of studies that validated their work on industrial projects. This result is expected because of the availability and accessibility of open source systems. However, open-source software is often developed with a different management style than the industrial ones. Thus, refactoring techniques and tools must be validated and checked for quality and reliability using industrial systems. More industrial collaborations are needed to bridge the gap between academic research and the industry's research needs, and therefore, produce groundbreaking research and innovation that solves complex real-world problems.

5 Conclusion

In this paper, we have conducted a systematic literature review on refactoring accompanied by meta-analysis to answer the defined research questions. After a comprehensive search that follows a systematic series of steps and assessing the quality of the studies, 3183 publications were identified. Based on these selected papers, we derived a taxonomy focused on five key aspects of Refactoring: refactoring life-cycle, artifacts affected by refactoring, refactoring objectives, refactoring techniques, and refactoring evaluation. Using this classification scheme, we analyzed the primary studies and presented the results in a way that enables researchers to relate their work to the current body of knowledge and identify future research directions. We also implemented a repository that helps researchers/practitioners collect and report papers about Refactoring. It also provides visualization charts and graphs that highlight the analysis results of our selected studies. This infrastructure will bridge the gap among the different refactoring communities and allow for more effortless knowledge transfer. To conclude, we believe that the results of our systematic review will help advance the refactoring research area. Since we expect this research

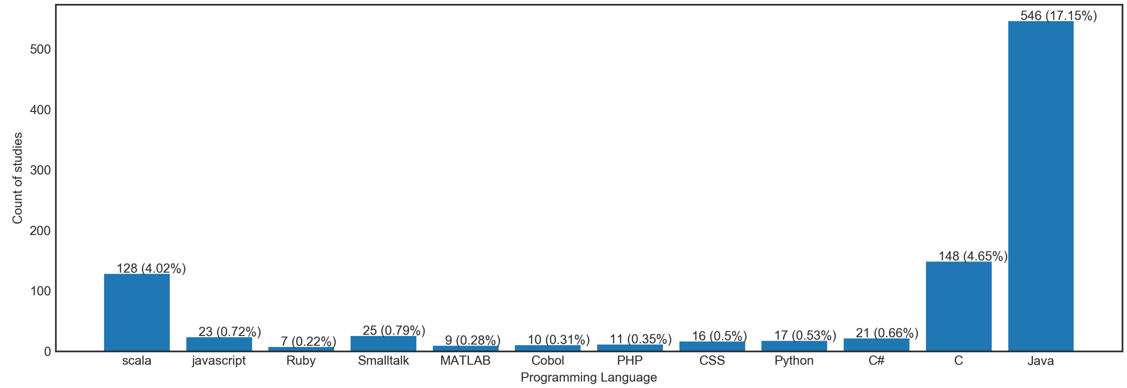


Fig. 18. histogram illustrating the count of refactoring publications per programming language

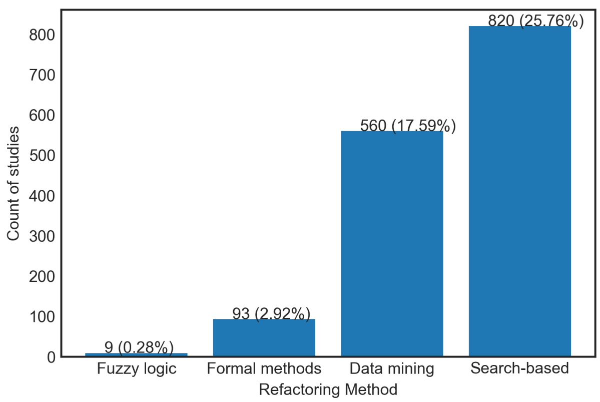


Fig. 19. histogram illustrating the count of refactoring publications per field

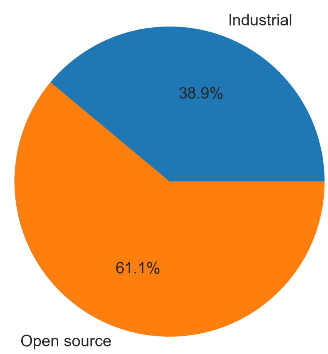


Fig. 20. Pie chart illustrating the percentage of publications in which the authors used industrial and/or open source systems in the validation step

area to continue to grow in the future, we hope that our repository and taxonomy will become useful in organizing, developing and judging new approaches.

REFERENCES

[1] W. F. Opdyke, "Refactoring: An aid in designing application frameworks and evolving object-oriented systems," in *Proc. SOOPPA'90: Symposium on Object-Oriented Programming Emphasizing Practical Applications*, 1990.
[2] W. G. Griswold, "Program restructuring as an aid to software maintenance." 1992.
[3] W. F. Opdyke, "Refactoring object-oriented frameworks," 1992.
[4] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, "Refactoring: Improving the Design of Existing Code," *Xtemp01*, pp. 1–337, 1999.
[5] C. A. C. Coello, G. B. Lamont, D. A. Van Veldhuizen *et al.*, *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007, vol. 5.
[6] W. Ma, L. Chen, Y. Zhou, and B. Xu, "Do We Have a Chance to Fix Bugs When Refactoring Code Smells?" *2016 International Conference on Software Analysis, Testing and Evolution (SATE)*, pp. 24–29, 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7780189/
[7] K. Stroggylos and D. Spinellis, "Refactoring–Does It Improve Software Quality?" *Fifth International Workshop on Software Quality (WoSQ'07: ICSE Workshops 2007)*, pp. 3–8, 2007.
[8] A. Kaur and M. Kaur, "Analysis of Code Refactoring Impact on Software Quality," *MATEC Web of Conferences*, vol. 57, p. 02012, may 2016. [Online]. Available: http://www.matec-conferences.org/10.1051/matecconf/20165702012
[9] G. Bavota, B. De Carluccio, A. De Lucia, M. Di Penta, R. Oliveto, and O. Strollo, "When does a refactoring induce bugs? An empirical study," *Proceedings - 2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation, SCAM 2012*, pp. 104–113, 2012.
[10] J. Al Dallal and A. Abdin, "Empirical evaluation of the impact of object-oriented code refactoring on quality attributes: A systematic literature review," *IEEE Transactions on Software Engineering*, vol. 44, no. 1, pp. 44–69, 2017.
[11] S. Singh and S. Kaur, "A systematic literature review: Refactoring for disclosing code smells in object oriented software," *Ain Shams Engineering Journal*, vol. 9, no. 4, pp. 2129–2151, 2018.
[12] T. Mens and T. Tourwé, "A survey of software refactoring," *IEEE Transactions on software engineering*, vol. 30, no. 2, pp. 126–139, 2004.
[13] K. O. Elish and M. Alshayeb, "A classification of refactoring methods based on software quality attributes," *Arabian Journal for Science and Engineering*, vol. 36, no. 7, pp. 1253–1267, 2011.
[14] B. Du Bois, P. Van Gorp, A. Amsel, N. Van Eetvelde, H. Stenten, S. Demeyer, and T. Mens, "A discussion of refactoring in research and practice," *Reporte Técnico. Universidad de Antwerpen, Bélgica*, 2004.
[15] T. Mens, A. Van Deursen *et al.*, "Refactoring: Emerging trends and open problems," in *Proceedings First International Workshop on*

REFactoring: Achievements, Challenges, Effects (REFACE). University of Waterloo, 2003.

[16] M. Misbhauddin and M. Alshayeb, "Uml model refactoring: a systematic literature review," *Empirical Software Engineering*, vol. 20, no. 1, pp. 206–251, 2015.

[17] J. Al Dallal, "Identifying refactoring opportunities in object-oriented code: A systematic literature review," *Information and software Technology*, vol. 58, pp. 231–249, 2015.

[18] M. Abebe and C.-J. Yoo, "Trends, opportunities and challenges of software refactoring: A systematic literature review," *International Journal of Software Engineering and Its Applications*, vol. 8, no. 6, pp. 299–318, 2014.

[19] A. A. B. Baqais and M. Alshayeb, "Automatic software refactoring: a systematic literature review," *Software Quality Journal*, pp. 1–44, 2019.

[20] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," 2007.

[21] A. Ramirez, J. R. Romero, and C. L. Simons, "A systematic review of interaction in search-based software engineering," *IEEE Transactions on Software Engineering*, vol. 45, no. 8, pp. 760–781, 2018.

[22] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, 2014, pp. 1–10.

[23] M. Galster, D. Weyns, D. Tofan, B. Michalik, and P. Avgeriou, "Variability in software systems—a systematic literature review," *IEEE Transactions on Software Engineering*, vol. 40, no. 3, pp. 282–306, 2013.

[24] (2020) Slr website. URL: https://slr.iselab.us/.

[25] S. Tichelaar, S. Ducasse, S. Demeyer, and O. Nierstrasz, "A meta-model for language-independent refactoring," in *Proceedings International Symposium on Principles of Software Evolution*. IEEE, 2000, pp. 154–164.

[26] M. Ó. Cinnéide and P. Nixon, "A methodology for the automated introduction of design patterns," in *Proceedings IEEE International Conference on Software Maintenance-1999 (ICSM'99).'Software Maintenance for Business Change'(Cat. No. 99CB36360)*. IEEE, 1999, pp. 463–472.

PRIMARY SOURCES

[S1] H. Sajnani, V. Saini, and C. V. Lopes, "A comparative study of bug patterns in java cloned and non-cloned code," in *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*. IEEE, 2014, pp. 21–30.

[S2] J. Ghofrani, M. Mohseni, and A. Bozorgmehr, "A conceptual framework for clone detection using machine learning," in *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI)*. IEEE, 2017, pp. 0810–0817.

[S3] I. Verebi, "A model-based approach to software refactoring," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 606–609.

[S4] M. Tufano, F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia, and D. Poshyvanyk, "An empirical investigation into the nature of test smells," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016, pp. 4–15.

[S5] B. Zhang, G. Huang, Z. Zheng, J. Ren, and C. Hu, "Approach to mine the modularity of software network based on the most vital nodes," *IEEE Access*, vol. 6, pp. 32 543–32 553, 2018.

[S6] G. Balogh, T. Gergely, Á. Beszédes, and T. Gyimóthy, "Are my unit tests in the right package?" in *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2016, pp. 137–146.

[S7] N. Tsantalis, D. Mazinanian, and G. P. Krishnan, "Assessing the refactorability of software clones," *IEEE Transactions on Software Engineering*, vol. 41, no. 11, pp. 1055–1090, 2015.

[S8] G. Soares, R. Gheyi, and T. Massoni, "Automated behavioral testing of refactoring engines," *IEEE Transactions on Software Engineering*, vol. 39, no. 2, pp. 147–162, 2012.

[S9] J. Zhang, S. Han, D. Hao, L. Zhang, and D. Zhang, "Automated refactoring of nested-if formulae in spreadsheets," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 833–838.

[S10] Y. Kataoka, M. D. Ernst, W. G. Griswold, and D. Notkin, "Automated support for program refactoring using invariants," in *Proceedings IEEE International Conference on Software Maintenance. ICSM 2001*. IEEE, 2001, pp. 736–743.

[S11] M. Mondal, C. K. Roy, and K. A. Schneider, "A comparative study on the bug-proneness of different types of code clones," in *2015 IEEE International conference on software maintenance and evolution (ICSME)*. IEEE, 2015, pp. 91–100.

[S12] V. Alizadeh, M. Kessentini, W. Mkaouer, M. Ocinneide, A. Ouni, and Y. Cai, "An interactive and dynamic search-based approach to software refactoring recommendations," *IEEE Transactions on Software Engineering*, 2018.

[S13] W. Snipes, B. Robinson, and E. Murphy-Hill, "Code hot spot: A tool for extraction and analysis of code change history," in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2011, pp. 392–401.

[S14] H. Liu, Q. Liu, Z. Niu, and Y. Liu, "Dynamic and automatic feedback-based threshold adaptation for code smell detection," *IEEE Transactions on Software Engineering*, vol. 42, no. 6, pp. 544–558, 2015.

[S15] V. Cosentino, S. Duenas, A. Zerouali, G. Robles, and J. M. González-Barahona, "Graal: The quest for source code knowledge."

[S16] S. Charalampidou, A. Ampatzoglou, A. Chatzigeorgiou, A. Gkortzis, and P. Avgeriou, "Identifying extract method refactoring opportunities based on functional relevance," *IEEE Transactions on Software Engineering*, vol. 43, no. 10, pp. 954–974, 2016.

[S17] A. Rani and J. K. Chhabra, "Prioritization of smelly classes: A two phase approach (reducing refactoring efforts)," in *2017 3rd International Conference on Computational Intelligence & Communication Technology (CICT)*. IEEE, 2017, pp. 1–6.

[S18] P. Rachow, "Refactoring decision support for developers and architects based on architectural impact," in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2019, pp. 262–266.

[S19] H. Liu, Z. Ma, W. Shao, and Z. Niu, "Schedule of bad smell detection and resolution: A new way to save effort," *IEEE transactions on Software Engineering*, vol. 38, no. 1, pp. 220–235, 2011.

[S20] J. Kim, D. Batory, and D. Dig, "Scripting parametric refactorings in java to retrofit design patterns," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 211–220.

[S21] M. A. Parande and G. Koru, "A longitudinal analysis of the dependency concentration in smaller modules for open-source software products," in *2010 IEEE International Conference on Software Maintenance*. IEEE, 2010, pp. 1–5.

[S22] H. Liu, Z. Xu, and Y. Zou, "Deep learning based feature envy detection," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 385–396.

[S23] R. Morales, R. Saborido, F. Khomh, F. Chicano, and G. Antoniol, "Earmo: An energy-aware refactoring approach for mobile apps," *IEEE Transactions on Software Engineering*, vol. 44, no. 12, pp. 1176–1206, 2017.

[S24] H. Liu, L. Yang, Z. Niu, Z. Ma, and W. Shao, "Facilitating software refactoring with appropriate resolution order of bad smells," in *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2009, pp. 265–268.

[S25] H. Liu, Q. Liu, Y. Liu, and Z. Wang, "Identifying renaming opportunities by expanding conducted rename refactorings," *IEEE Transactions on Software Engineering*, vol. 41, no. 9, pp. 887–900, 2015.

[S26] B. Lin, S. Scalabrino, A. Mocci, R. Oliveto, G. Bavota, and M. Lanza, "Investigating the use of code analysis and nlp to promote a consistent usage of identifiers," in *2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2017, pp. 81–90.

[S27] G. Bavota, R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, "Methodbook: Recommending move method refactorings via relational topic models," *IEEE Transactions on Software Engineering*, vol. 40, no. 7, pp. 671–694, 2013.

[S28] C. Hinds-Charles, J. Adames, Y. Yang, Y. Shen, and Y. Wang, "A longitude analysis on bitcoin issue repository," in *2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN)*. IEEE, 2018, pp. 212–217.

[S29] T. D. Oyetoyan, D. S. Cruzes, and C. Thurmann-Nielsen, "A decision support system to refactor class cycles," in *2015 IEEE*

International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2015, pp. 231–240.

[S30] N. Rachatasumrit and M. Kim, "An empirical investigation into the impact of refactoring on regression testing," in *2012 28th Ieee International Conference on Software Maintenance (Icsm)*. IEEE, 2012, pp. 357–366.

[S31] M. Mirzaaghaei, F. Pastore, and M. Pezze, "Automatically repairing test cases for evolving method declarations," in *2010 IEEE International Conference on Software Maintenance*. IEEE, 2010, pp. 1–5.

[S32] B. Van Rompaey, B. Du Bois, and S. Demeyer, "Characterizing the relative significance of a test smell," in *2006 22nd IEEE International Conference on Software Maintenance*. IEEE, 2006, pp. 391–400.

[S33] A. Sherwany, N. Zaza, and N. Nystrom, "A refactoring library for scala compiler extensions," in *International Conference on Compiler Construction*. Springer, 2015, pp. 31–48.

[S34] S. Paydar and M. Kahani, "A semantic web based approach for design pattern detection from source code," in *2012 2nd International eConference on Computer and Knowledge Engineering (ICCKE)*. IEEE, 2012, pp. 289–294.

[S35] T. Haendler, "A card game for learning software-refactoring principles," 2019.

[S36] C. Kastner, S. Apel, and D. Batory, "A case study implementing features using aspectj," in *11th International Software Product Line Conference (SPLC 2007)*. IEEE, 2007, pp. 223–232.

[S37] T. Viana, "A catalog of bad smells in design-by-contract methodologies with java modeling language," *Journal of Computing Science and Engineering*, vol. 7, no. 4, pp. 251–262, 2013.

[S38] D. Foetsch and E. Pulvermueller, "A concept and implementation of higher-level xml transformation languages," *Knowledge-Based Systems*, vol. 22, no. 3, pp. 186–194, 2009.

[S39] J. Reutelshoefer, J. Baumeister, and F. Puppe, "A data structure for the refactoring of multimodal knowledge," in *Proceedings of the 5th Workshop on Knowledge Engineering and Software Engineering*, 2009, pp. 33–45.

[S40] S. Mouchawrab, L. C. Briand, and Y. Labiche, "A measurement framework for object-oriented software testability," *Information and software technology*, vol. 47, no. 15, pp. 979–997, 2005.

[S41] I. Cassol and G. Arévalo, "A methodology to infer and refactor an object-oriented model from c applications," *Software: Practice and Experience*, vol. 48, no. 3, pp. 550–577, 2018.

[S42] G. De Ruvo and A. Santone, "A novel methodology based on formal methods for analysis and verification of wikis," in *2014 IEEE 23rd International WETICE Conference*. IEEE, 2014, pp. 411–416.

[S43] S. Rebai, O. B. Sghaier, V. Alizadeh, M. Kessentini, and M. Chater, "Interactive refactoring documentation bot," in *2019 19th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2019, pp. 152–162.

[S44] J. Krinke, "Mining execution relations for crosscutting concerns," *IET software*, vol. 2, no. 2, pp. 65–78, 2008.

[S45] D. Bowes, D. Randall, and T. Hall, "The inconsistent measurement of message chains," in *2013 4th International Workshop on Emerging Trends in Software Metrics (WETSoM)*. IEEE, 2013, pp. 62–68.

[S46] J. Liu, "Feature interactions and software derivatives." *Journal of Object Technology*, vol. 4, no. 3, pp. 13–19, 2004.

[S47] A. Swidan, F. Hermans, and R. Koesoemowidjojo, "Improving the performance of a large scale spreadsheet: a case study," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1. IEEE, 2016, pp. 673–677.

[S48] H. Li, S. Thompson, and T. Arts, "Extracting properties from test cases by refactoring," in *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 2011, pp. 472–473.

[S49] S. Ducasse, O. Nierstrasz, N. Schärli, R. Wuyts, and A. P. Black, "Traits: A mechanism for fine-grained reuse," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 28, no. 2, pp. 331–388, 2006.

[S50] R. Ramos, J. Castro, J. Araújo, F. Alencar, and R. Penteado, "Divide and conquer refactoring: dealing with the large, scattering or tangling use case model," in *Proceedings of the 8th Latin American Conference on Pattern Languages of Programs*, 2010, pp. 1–11.

[S51] E. Murphy-Hill, A. P. Black, D. Dig, and C. Parnin, "Gathering refactoring data: a comparison of four methods," in *Proceedings of the 2nd Workshop on Refactoring Tools*, 2008, pp. 1–5.

[S52] A. Derezińska, "A structure-driven process of automated refactoring to design patterns," in *International Conference on Information Systems Architecture and Technology*. Springer, 2017, pp. 39–48.

[S53] E. Selim, Y. Ghanam, C. Burns, T. Seyed, and F. Maurer, "A test-driven approach for extracting libraries of reusable components from existing applications," in *International Conference on Agile Software Development*. Springer, 2011, pp. 238–252.

[S54] Y. Zhang, S. Dong, X. Zhang, H. Liu, and D. Zhang, "Automated refactoring for stampedlock," *IEEE Access*, vol. 7, pp. 104 900–104 911, 2019.

[S55] H. Xue, S. Sun, G. Venkataramani, and T. Lan, "Machine learning-based analysis of program binaries: A comprehensive study," *IEEE Access*, vol. 7, pp. 65 889–65 912, 2019.

[S56] Y. Zhang, S. Shao, H. Liu, J. Qiu, D. Zhang, and G. Zhang, "Refactoring java programs for customizable locks based on bytecode transformation," *IEEE Access*, vol. 7, pp. 66 292–66 303, 2019.

[S57] M. F. Dolz, D. D. R. Astorga, J. Fernández, J. D. García, and J. Carretero, "Towards automatic parallelization of stream processing applications," *IEEE Access*, vol. 6, pp. 39 944–39 961, 2018.

[S58] B. K. Sidhu, K. Singh, and N. Sharma, "A catalogue of model smells and refactoring operations for object-oriented software," in *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*. IEEE, 2018, pp. 313–319.

[S59] F. Medeiros, M. Ribeiro, R. Gheyi, and B. F. dos Santos Neto, "A catalogue of refactorings to remove incomplete annotations." *J. UCS*, vol. 20, no. 5, pp. 746–771, 2014.

[S60] P. Ma, Y. Bian, and X. Su, "A clustering method for pruning false positive of clonde code detection," in *Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)*. IEEE, 2013, pp. 1917–1920.

[S61] G.-S. Cojocar and A.-M. Guran, "A comparative analysis of monitoring concerns implementation in object oriented systems," in *2018 IEEE 12th International Symposium on Applied Computational Intelligence and Informatics (SACI)*. IEEE, 2018, pp. 000 355–000 360.

[S62] S. Negara, N. Chen, M. Vakilian, R. E. Johnson, and D. Dig, "A comparative study of manual and automated refactorings," in *European Conference on Object-Oriented Programming*. Springer, 2013, pp. 552–576.

[S63] T. Chen and C. He, "A comparison of approaches to legacy system crosscutting concerns mining," in *2013 International Conference on Computer Sciences and Applications*. IEEE, 2013, pp. 813–816.

[S64] A. Martini, E. Sikander, and N. Madlani, "A semi-automated framework for the identification and estimation of architectural technical debt: A comparative case-study on the modularization of a software component," *Information and Software Technology*, vol. 93, pp. 264–279, 2018.

[S65] M. T. Valente, V. Borges, and L. Passos, "A semi-automatic approach for extracting software product lines," *IEEE Transactions on Software Engineering*, vol. 38, no. 4, pp. 737–754, 2011.

[S66] K. Garcés, J. M. Vara, F. Jouault, and E. Marcos, "Adapting transformations to metamodel changes via external transformation composition," *Software & Systems Modeling*, vol. 13, no. 2, pp. 789–806, 2014.

[S67] S. A. Vidal, C. Marcos, and J. A. Díaz-Pace, "An approach to prioritize code smells for refactoring," *Automated Software Engineering*, vol. 23, no. 3, pp. 501–532, 2016.

[S68] C. Brown, H. Li, and S. Thompson, "An expression processor: a case study in refactoring haskell programs," in *International Symposium on Trends in Functional Programming*. Springer, 2010, pp. 31–49.

[S69] M. Marin, A. van Deursen, L. Moonen, and R. van der Rijst, "An integrated crosscutting concern migration strategy and its semi-automated application to jhotdraw," *Automated Software Engineering*, vol. 16, no. 2, pp. 323–356, 2009.

[S70] A. O'Riordan, "Aspect-oriented reengineering of an object-oriented library in a short iteration agile process," *Informatica*, vol. 35, no. 4, 2011.

[S71] K. Fujiwara, K. Fushida, N. Yoshida, and H. Iida, "Assessing refactoring instances and the maintainability benefits of them from version archives," in *International Conference on Product*

Focused Software Process Improvement. Springer, 2013, pp. 313–323.

[S72] B. Alkhazi, T. Ruas, M. Kessentini, M. Wimmer, and W. I. Grosky, "Automated refactoring of atl model transformations: a search-based approach," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, 2016, pp. 295–304.

[S73] M. Tanhaei, J. Habibi, and S.-H. Mirian-Hosseinabadi, "Automating feature model refactoring: A model transformation approach," *Information and Software Technology*, vol. 80, pp. 138–157, 2016.

[S74] V. Alizadeh, H. Fehri, and M. Kessentini, "Less is more: From multi-objective to mono-objective refactoring via developer's knowledge extraction," in *2019 19th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2019, pp. 181–192.

[S75] V. Alizadeh, M. A. Ouali, M. Kessentini, and M. Chater, "Refbot: intelligent software refactoring bot," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 823–834.

[S76] Z. Mushtaq, G. Rasool, and B. Shehzad, "Multilingual source code analysis: A systematic literature review," *IEEE Access*, vol. 5, pp. 11 307–11 336, 2017.

[S77] F. Schmidt, S. G. MacDonell, and A. M. Connor, "An automatic architecture reconstruction and refactoring framework," in *Software Engineering Research, Management and Applications 2011*. Springer, 2012, pp. 95–111.

[S78] G. Cong, H. Wen, I.-h. Chung, D. Klepacki, H. Murata, and Y. Negishi, "An efficient framework for multi-dimensional tuning of high performance computing applications," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*. IEEE, 2012, pp. 1376–1387.

[S79] J. Park, M. Kim, and D.-H. Bae, "An empirical study of supplementary patches in open source projects," *Empirical Software Engineering*, vol. 22, no. 1, pp. 436–473, 2017.

[S80] T. L. Nguyen, A. Fish, and M. Song, "An empirical study on similar changes in evolving software," in *2018 IEEE International Conference on Electro/Information Technology (EIT)*. IEEE, 2018, pp. 0560–0563.

[S81] M. Bruntink, A. Van Deursen, T. Tourwe, and R. van Engelen, "An evaluation of clone detection techniques for crosscutting concerns," in *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.* IEEE, 2004, pp. 200–209.

[S82] Y. Kosker, B. Turhan, and A. Bener, "An expert system for determining candidate software classes for refactoring," *Expert Systems with Applications*, vol. 36, no. 6, pp. 10 000–10 003, 2009.

[S83] B. L. Sousa, M. A. Bigonha, and K. A. Ferreira, "An exploratory study on cooccurrence of design patterns and bad smells using software metrics," *Software: Practice and Experience*, vol. 49, no. 7, pp. 1079–1113, 2019.

[S84] O. Mehani, G. Jourjon, T. Rakotoarivelo, and M. Ott, "An instrumentation framework for the critical task of measurement collection in the future internet," *Computer Networks*, vol. 63, pp. 68–83, 2014.

[S85] M. Schäfer, A. Thies, F. Steimann, and F. Tip, "A comprehensive approach to naming and accessibility in refactoring java programs," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1233–1257, 2012.

[S86] D. Dig, "A practical tutorial on refactoring for parallelism," in *2010 IEEE International Conference on Software Maintenance*. IEEE, 2010, pp. 1–2.

[S87] X. Li and J. P. Gallagher, "A source-level energy optimization framework for mobile applications," in *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2016, pp. 31–40.

[S88] R. Khatchadourian, Y. Tang, M. Bagherzadeh, and S. Ahmed, "[engineering paper] a tool for optimizing java 8 stream software via automated refactoring," in *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2018, pp. 34–39.

[S89] Z. Xing and E. Stroulia, "Api-evolution support with diff-catchup," *IEEE Transactions on Software Engineering*, vol. 33, no. 12, pp. 818–836, 2007.

[S90] R. Gheyi, T. Massoni, and P. Borba, "A rigorous approach for proving model refactorings," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, 2005, pp. 372–375.

[S91] B. Cyganek, "Adding parallelism to the hybrid image processing library in multi-threading and multi-core systems," in *2011 IEEE 2nd International Conference on Networked Embedded Systems for Enterprise Applications*. IEEE, 2011, pp. 1–8.

[S92] R. Hardt and E. V. Munson, "An empirical evaluation of ant build maintenance using formiga," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 201–210.

[S93] R. Kolb, D. Muthig, T. Patzke, and K. Yamauchi, "A case study in refactoring a legacy component for reuse in a product line," in *21st IEEE International Conference on Software Maintenance (ICSM'05)*. IEEE, 2005, pp. 369–378.

[S94] D. Strein, R. Lincke, J. Lundberg, and W. Löwe, "An extensible meta-model for program analysis," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 592–607, 2007.

[S95] Y.-W. Kwon, "Automated s/w reengineering for fault-tolerant and energy-efficient distributed execution," in *2013 IEEE International Conference on Software Maintenance*. IEEE, 2013, pp. 582–585.

[S96] B. Adams, H. Tromp, K. De Schutter, and W. De Meuter, "Design recovery and maintenance of build systems," in *2007 IEEE International Conference on Software Maintenance*. IEEE, 2007, pp. 114–123.

[S97] R. Bahsoon and W. Emmerich, "Evaluating architectural stability with real options theory," in *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.* IEEE, 2004, pp. 443–447.

[S98] J. O'neal, K. Weide, and A. Dubey, "Experience report: refactoring the mesh interface in flash, a multiphysics software," in *2018 IEEE 14th International Conference on e-Science (e-Science)*. IEEE, 2018, pp. 1–6.

[S99] M. A. Khan and H. Tembine, "Meta-learning for realizing self-x management of future networks," *IEEE Access*, vol. 5, pp. 19 072–19 083, 2017.

[S100] A. Cleve, "Program analysis and transformation for data-intensive system evolution," in *2010 IEEE International Conference on Software Maintenance*. IEEE, 2010, pp. 1–6.

[S101] D. Binkley, M. Ceccato, M. Harman, F. Ricca, and P. Tonella, "Automated refactoring of object oriented code into aspects," in *21st IEEE International Conference on Software Maintenance (ICSM'05)*. IEEE, 2005, pp. 27–36.

[S102] F. Castor Filho, A. Garcia, and C. M. F. Rubira, "Extracting error handling to aspects: A cookbook," in *2007 IEEE International Conference on Software Maintenance*. IEEE, 2007, pp. 134–143.

[S103] M. Bajammal, D. Mazinanian, and A. Mesbah, "Generating reusable web components from mockups," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 601–611.

[S104] N. A. Kraft, E. B. Duffy, and B. A. Malloy, "Grammar recovery from parse trees and metrics-guided grammar refactoring," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 780–794, 2009.

[S105] D. Spinellis, "Global analysis and transformations in preprocessed languages," *IEEE Transactions on Software Engineering*, vol. 29, no. 11, pp. 1019–1030, 2003.

[S106] C. Noguera, A. Kellens, C. De Roover, and V. Jonckers, "Refactoring in the presence of annotations," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2012, pp. 337–346.

[S107] S. Rongrong, Z. Liping, and Z. Fengrong, "A method for identifying and recommending reconstructed clones," in *Proceedings of the 2019 3rd International Conference on Management Engineering, Software Engineering and Service Sciences*, 2019, pp. 39–44.

[S108] Y. Khan and M. El-Attar, "A model transformation approach towards refactoring use case models based on antipatterns," in *21st International Conference on Software Engineering and Data Engineering (SEDE'12), Los Angeles, California, USA*, 2012, pp. 49–54.

[S109] K. Grolinger and M. A. Capretz, "A unit test approach for database schema evolution," *Information and Software Technology*, vol. 53, no. 2, pp. 159–170, 2011.

[S110] O. Febbraro, K. Reale, and F. Ricca, "Aspide: Integrated development environment for answer set programming," in *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer, 2011, pp. 317–330.

[S111] A. Garrido and R. Johnson, "Analyzing multiple configurations of a c program," in *21st IEEE International Conference on Software Maintenance (ICSM'05)*. IEEE, 2005, pp. 379–388.

[S112] A. Paltoglou, V. E. Zafeiris, E. A. Giakoumakis, and N. Diamantidis, "Automated refactoring of client-side javascript code to es6 modules," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2018, pp. 402–412.

[S113] R. Khatchadourian, J. Sawin, and A. Rountev, "Automated refactoring of legacy java software to enumerated types," in *2007 IEEE International Conference on Software Maintenance*. IEEE, 2007, pp. 224–233.

[S114] M. Marin, L. Moonen, and A. van Deursen, "A classification of crosscutting concerns," in *21st IEEE International Conference on Software Maintenance (ICSM'05)*. IEEE, 2005, pp. 673–676.

[S115] M. Mortensen, S. Ghosh, and J. Bieman, "Aspect-oriented refactoring of legacy applications: An evaluation," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 118–140, 2010.

[S116] M. Mondal, C. K. Roy, and K. A. Schneider, "Automatic identification of important clones for refactoring and tracking," in *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*. IEEE, 2014, pp. 11–20.

[S117] A. Kellens, K. De Schutter, T. D'Hondt, V. Jonckers, and H. Doggen, "Experiences in modularizing business rules into aspects," in *2008 IEEE International Conference on Software Maintenance*. IEEE, 2008, pp. 448–451.

[S118] R. Stoiber, S. Fricker, M. Jehle, and M. Glinz, "Feature unweaving: Refactoring software requirements specifications into software product lines," in *2010 18th IEEE International Requirements Engineering Conference*. IEEE, 2010, pp. 403–404.

[S119] G. Zhao and J. Huang, "Deepsim: deep learning code functional similarity," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 141–151.

[S120] S. Demeyer, S. Ducasse, and O. Nierstrasz, "Object-oriented reengineering: patterns and techniques," in *21st IEEE International Conference on Software Maintenance (ICSM'05)*. IEEE, 2005, pp. 723–724.

[S121] P. Hegedus, "Revealing the effect of coding practices on software maintainability," in *2013 IEEE International Conference on Software Maintenance*. IEEE, 2013, pp. 578–581.

[S122] T. Feng, J. Zhang, H. Wang, and X. Wang, "Software design improvement through anti-patterns identification," in *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.* IEEE, 2004, p. 524.

[S123] S. Meng and L. S. Barbosa, "A coalgebraic semantic framework for reasoning about uml sequence diagrams," in *2008 The Eighth International Conference on Quality Software*. IEEE, 2008, pp. 17–26.

[S124] P. Mayer and A. Schroeder, "Cross-language code analysis and refactoring," in *2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation*. IEEE, 2012, pp. 94–103.

[S125] R. Moser, P. Abrahamsson, W. Pedrycz, A. Sillitti, and G. Succi, "A case study on the impact of refactoring on quality and productivity in an agile team," in *IFIP Central and East European Conference on Software Engineering Techniques*. Springer, 2007, pp. 252–266.

[S126] A. L. Cândido, F. A. Trinta, L. S. Rocha, P. A. Rego, N. C. Mendonça, and V. C. Garcia, "A microservice based architecture to support offloading in mobile cloud computing," in *Proceedings of the XIII Brazilian Symposium on Software Components, Architectures, and Reuse*, 2019, pp. 93–102.

[S127] A. Peruma, "A preliminary study of android refactorings," in *2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 2019, pp. 148–149.

[S128] M. Mascheroni and E. Irrazábal, "A design pattern approach for restful tests: A case study," in *IEEE 12th Colombian Computing Congress*, 2018.

[S129] D. Kermek, T. Jakupić, and N. Vrček, "A model of heterogeneous distributed system for foreign exchange portfolio analysis," *Journal of Information and Organizational Sciences*, vol. 30, no. 1, pp. 83–92, 2006.

[S130] G. Rodriguez, A. Teyseyre, Á. Soria, and L. Berdun, "A visualization tool to detect refactoring opportunities in soa applications," in *2017 XLIII Latin American Computer Conference (CLEI)*. IEEE, 2017, pp. 1–10.

[S131] H. Li, S. Thompson, P. Lamela Seijas, and M. A. Francisco, "Automating property-based testing of evolving web services," in *Proceedings of the ACM SIGPLAN 2014 Workshop on Partial Evaluation and Program Manipulation*, 2014, pp. 169–180.

[S132] D. Athanasopoulos, A. V. Zarras, G. Miskos, V. Issarny, and P. Vassiliadis, "Cohesion-driven decomposition of service interfaces without access to source code," *IEEE Transactions on Services Computing*, vol. 8, no. 4, pp. 550–562, 2014.

[S133] M. Kessentini and H. Wang, "Detecting refactorings among multiple web service releases: A heuristic-based approach," in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 365–372.

[S134] F. Wei, C. Ouyang, and A. Barros, "Discovering behavioural interfaces for overloaded web services," in *2015 IEEE World Congress on Services*. IEEE, 2015, pp. 286–293.

[S135] K. Fekete, A. Pelle, and K. Csorba, "Energy efficient code optimization in mobile environment," in *2014 IEEE 36th International Telecommunications Energy Conference (INTELEC)*. IEEE, 2014, pp. 1–6.

[S136] W. B. Langdon, "Genetic improvement of programs," in *2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. IEEE, 2014, pp. 14–19.

[S137] H. Wang, A. Ouni, M. Kessentini, B. Maxim, and W. I. Grosky, "Identification of web service refactoring opportunities as a multi-objective problem," in *2016 IEEE International Conference on Web Services (ICWS)*. IEEE, 2016, pp. 586–593.

[S138] M. Kim, T. Zimmermann, and N. Nagappan, "An empirical study of refactoringchallenges and benefits at microsoft," *IEEE Transactions on Software Engineering*, vol. 40, no. 7, pp. 633–649, 2014.

[S139] S. M. Olbrich, D. S. Cruzes, and D. I. Sjøberg, "Are all code smells harmful? a study of god classes and brain classes in the evolution of three open source systems," in *2010 IEEE International Conference on Software Maintenance*. IEEE, 2010, pp. 1–10.

[S140] P. S. Kochhar, F. Thung, and D. Lo, "Automatic fine-grained issue report reclassification," in *2014 19th International Conference on Engineering of Complex Computer Systems*. IEEE, 2014, pp. 126–135.

[S141] G. Bastide, A. Seriai, and M. Oussalah, "Dynamic adaptation of software component structures," in *2006 IEEE International Conference on Information Reuse & Integration*. IEEE, 2006, pp. 404–409.

[S142] G. Zhang, L. Shen, X. Peng, Z. Xing, and W. Zhao, "Incremental and iterative reengineering towards software product line: An industrial case study," in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2011, pp. 418–427.

[S143] G. Bavota, R. Oliveto, A. De Lucia, G. Antoniol, and Y.-G. Gueheneuc, "Playing with refactoring: Identifying extract class opportunities through game theory," in *2010 IEEE International Conference on Software Maintenance*. IEEE, 2010, pp. 1–5.

[S144] P. S. Kochhar, Y. Tian, and D. Lo, "Potential biases in bug localization: Do they matter?" in *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, 2014, pp. 803–814.

[S145] R. Khatchadourian and H. Masuhara, "Defaultification refactoring: A tool for automatically converting java methods to default," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 984–989.

[S146] H. K. Wright, D. Jasper, M. Klimek, C. Carruth, and Z. Wan, "Large-scale automated refactoring using clangmr," in *2013 IEEE International Conference on Software Maintenance*. IEEE, 2013, pp. 548–551.

[S147] D. Mazinanian and N. Tsantalis, "Migrating cascading style sheets to preprocessors by introducing mixins," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016, pp. 672–683.

[S148] P. Tonella and M. Ceccato, "Migrating interface implementations to aspects," in *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.* IEEE, 2004, pp. 220–229.

[S149] M. Ceccato, "Migrating object oriented code to aspect oriented programming," 2006.

[S150] D. Majumdar, "Migration from procedural programming to aspect oriented paradigm," in *2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2009, pp. 712–715.

[S151] C. Marcos, S. Vidal, E. Abait, M. Arroqui, and S. Sampaoli, "Refactoring of a beef-cattle farm simulator," *IEEE Latin America Transactions*, vol. 9, no. 7, pp. 1099–1104, 2011.

[S152] R. Khatchadourian and B. Muskalla, "Enumeration refactoring: a tool for automatically converting java constants to enumerated types," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*, 2010, pp. 181–182.

[S153] E. L. Alves, M. Song, T. Massoni, P. D. Machado, and M. Kim, "Refactoring inspection support for manual refactoring edits," *IEEE Transactions on Software Engineering*, vol. 44, no. 4, pp. 365–383, 2017.

[S154] Y. Yu, J. Jurjens, and J. Mylopoulos, "Traceability for the maintenance of secure software," in *2008 IEEE International Conference on Software Maintenance*. IEEE, 2008, pp. 297–306.

[S155] C. Kulkarni, "Notice of violation of ieee publication principles a qualitative approach for refactoring of code clone opportunities using graph and tree methods," in *2016 International Conference on Information Technology (InCITe)-The Next Generation IT Summit on the Theme-Internet of Things: Connect your Worlds*. IEEE, 2016, pp. 154–159.

[S156] A. F. Tappenden, T. Huynh, J. Miller, A. Geras, and M. Smith, "Agile development of secure web-based applications," *International Journal of Information Technology and Web Engineering (IJITWE)*, vol. 1, no. 2, pp. 1–24, 2006.

[S157] P. M. Cousot, R. Cousot, F. Logozzo, and M. Barnett, "An abstract interpretation framework for refactoring with application to extract methods with contracts," in *Proceedings of the ACM international conference on Object oriented programming systems languages and applications*, 2012, pp. 213–232.

[S158] P. Borba, "An introduction to software product line refactoring," in *International Summer School on Generative and Transformational Techniques in Software Engineering*. Springer, 2009, pp. 1–26.

[S159] O. Macek and K. Richta, "Application and relational database co-refactoring," *Computer Science and Information Systems*, vol. 11, no. 2, pp. 503–524, 2014.

[S160] M. S. Feather and L. Z. Markosian, "Architecting and generalizing a safety case for critical condition detection software an experience report," in *2013 1st International Workshop on Assurance Cases for Software-Intensive Systems (ASSURE)*. IEEE, 2013, pp. 29–33.

[S161] P. Muntean, M. Monperrus, H. Sun, J. Grossklags, and C. Eckert, "Intrepair: Informed repairing of integer overflows," *IEEE Transactions on Software Engineering*, 2019.

[S162] S. Demeyer, "Refactor conditionals into polymorphism: what's the performance cost of introducing virtual calls?" in *21st IEEE International Conference on Software Maintenance (ICSM'05)*. IEEE, 2005, pp. 627–630.

[S163] A. Kumar, A. Sutton, and B. Stroustrup, "Rejuvenating c++ programs through demacrofication," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2012, pp. 98–107.

[S164] ——, "The demacrofier," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2012, pp. 658–661.

[S165] I. Sora, "A meta-model for representing language-independent primary dependency structures." in *ENASE*, 2012, pp. 65–74.

[S166] M. F. Zibran, R. K. Saha, M. Asaduzzaman, and C. K. Roy, "Analyzing and forecasting near-miss clones in evolving software: An empirical study," in *2011 16th IEEE International Conference on Engineering of Complex Computer Systems*. IEEE, 2011, pp. 295–304.

[S167] R. Rolim, "Automating repetitive code changes using examples," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 1063–1065.

[S168] W. S. Evans, C. W. Fraser, and F. Ma, "Clone detection via structural abstraction," *Software Quality Journal*, vol. 17, no. 4, pp. 309–330, 2009.

[S169] A. Khan, H. A. Basit, S. M. Sarwar, and M. M. Yousaf, "Cloning in popular server side technologies using agile development: An empirical study," *Pakistan Journal of Engineering and Applied Sciences*, no. 1, 2018.

[S170] T. D. Oyetoyan, R. Conradi, and D. S. Cruzes, "Criticality of defects in cyclic dependent components," in *2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2013, pp. 21–30.

[S171] M. Gatrell, S. Counsell, and T. Hall, "Empirical support for two refactoring studies using commercial c# software," in *13th International Conference on Evaluation and Assessment in Software Engineering (EASE) 13*, 2009, pp. 1–10.

[S172] R. K. Saha, M. Asaduzzaman, M. F. Zibran, C. K. Roy, and K. A. Schneider, "Evaluating code clone genealogies at release level: An empirical study," in *2010 10th IEEE Working Conference on Source Code Analysis and Manipulation*. IEEE, 2010, pp. 87–96.

[S173] A. Derezińska and M. Byczkowski, "Evaluation of design pattern utilization and software metrics in c# programs," in *International Conference on Dependability and Complex Systems*. Springer, 2019, pp. 132–142.

[S174] Y. A. Liu, M. Gorbovitski, and S. D. Stoller, "A language and framework for invariant-driven transformations," *ACM Sigplan Notices*, vol. 45, no. 2, pp. 55–64, 2009.

[S175] I. Lanc, P. Bui, D. Thain, and S. Emrich, "Adapting bioinformatics applications for heterogeneous systems: a case study," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 4, pp. 866–877, 2014.

[S176] L. E. d. S. Amorim, M. J. Steindorfer, S. Erdweg, and E. Visser, "Declarative specification of indentation rules: a tooling perspective on parsing and pretty-printing layout-sensitive languages," in *Proceedings of the 11th ACM SIGPLAN International Conference on Software Language Engineering*, 2018, pp. 3–15.

[S177] Z. Chen, L. Chen, W. Ma, and B. Xu, "Detecting code smells in python programs," in *2016 International Conference on Software Analysis, Testing and Evolution (SATE)*. IEEE, 2016, pp. 18–23.

[S178] C. Chapman and K. T. Stolee, "Exploring regular expression usage and context in python," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, 2016, pp. 282–293.

[S179] J. B. Cabral, B. Sánchez, F. Ramos, S. Gurovich, P. M. Granitto, and J. Vanderplas, "From fats to feets: Further improvements to an astronomical feature extraction tool based on machine learning," *Astronomy and computing*, vol. 25, pp. 213–220, 2018.

[S180] M. Zhu, F. McKenna, and M. H. Scott, "Openseespy: Python library for the opensees finite element framework," *SoftwareX*, vol. 7, pp. 6–11, 2018.

[S181] M. Furr, J.-h. An, and J. S. Foster, "Profile-guided static typing for dynamic scripting languages," in *Proceedings of the 24th ACM SIGPLAN conference on Object oriented programming systems languages and applications*, 2009, pp. 283–300.

[S182] Z. W. Bell, G. G. Davidson, T. M. D'Azevedo, W. Joubert, J. K. Munro Jr, D. R. Patlolla, and B. Vacaliuc, "Python for development of openmp and cuda kernels for multidimensional data," in *Symposium on Application Accelerators in HPC*, 2011.

[S183] Y. Hu, U. Z. Ahmed, S. Mechtaev, B. Leong, and A. Roychoudhury, "Re-factoring based program repair applied to programming assignments," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 388–398.

[S184] C. Wang, S. Hirasawa, H. Takizawa, and H. Kobayashi, "A platform-specific code smell alert system for high performance computing applications," in *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*. IEEE, 2014, pp. 652–661.

[S185] Ç. Biray and F. Buzluca, "A learning-based method for detecting defective classes in object-oriented systems," in *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2015, pp. 1–8.

[S186] W. Hasanain, Y. Labiche, and S. Eldh, "An analysis of complex industrial test code using clone analysis," in *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2018, pp. 482–489.

[S187] D. Mazinanian and N. Tsantalis, "An empirical study on the use of css preprocessors," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1. IEEE, 2016, pp. 168–178.

[S188] ——, "Cssdev: refactoring duplication in cascading style sheets," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2017, pp. 63–66.

[S189] D. Mazinanian, N. Tsantalis, and A. Mesbah, "Discovering refactoring opportunities in cascading style sheets," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 496–506.

[S190] D. D. Perez and W. Le, "Generating predicate callback summaries for the android framework," in *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 2017, pp. 68–78.

[S191] S. Negara, M. Vakilian, N. Chen, R. E. Johnson, and D. Dig, "Is it dangerous to use version control histories to study source code evolution?" in *European Conference on Object-Oriented Programming*. Springer, 2012, pp. 79–103.

[S192] M. Bosch, P. Genevès, and N. Layaïda, "Reasoning with style," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[S193] H. A. Nguyen, H. V. Nguyen, T. T. Nguyen, and T. N. Nguyen, "Output-oriented refactoring in php-based dynamic web applications," in *2013 IEEE International Conference on Software Maintenance*. IEEE, 2013, pp. 150–159.

[S194] B. Chen, Z. M. Jiang, P. Matos, and M. Lacaria, "An industrial experience report on performance-aware refactoring on a database-centric web application," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 653–664.

[S195] L. Eshkevari, F. Dos Santos, J. R. Cordy, and G. Antoniol, "Are php applications ready for hack?" in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 63–72.

[S196] J. L. Overbey and R. E. Johnson, "Differential precondition checking: A lightweight, reusable analysis for refactoring tools," in *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. IEEE, 2011, pp. 303–312.

[S197] J. L. Overbey, R. E. Johnson, and M. Hafiz, "Differential precondition checking: a language-independent, reusable analysis for refactoring engines," *Automated Software Engineering*, vol. 23, no. 1, pp. 77–104, 2016.

[S198] M. Hills, P. Klint, and J. J. Vinju, "Enabling php software engineering research in rascal," *Science of Computer Programming*, vol. 134, pp. 37–46, 2017.

[S199] F. Gauthier, D. Letarte, T. Lavoie, and E. Merlo, "Extraction and comprehension of moodle's access control model: A case study," in *2011 Ninth Annual International Conference on Privacy, Security and Trust*. IEEE, 2011, pp. 44–51.

[S200] M. Hills and P. Klint, "Php air: Analyzing php systems with rascal," in *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. IEEE, 2014, pp. 454–457.

[S201] Y. Yu, Y. Wang, J. Mylopoulos, S. Liaskos, A. Lapouchnian, and J. C. S. do Prado Leite, "Reverse engineering goal models from legacy code," in *13th IEEE International Conference on Requirements Engineering (RE'05)*. IEEE, 2005, pp. 363–372.

[S202] R. Lämmel and J. Visser, "A strafunski application letter," in *International Symposium on Practical Aspects of Declarative Languages*. Springer, 2003, pp. 357–375.

[S203] G. M. Rama, "A desiderata for refactoring-based software modularity improvement," in *Proceedings of the 3rd India software engineering conference*, 2010, pp. 93–102.

[S204] T. Mens and T. Tourwé, "A survey of software refactoring," *IEEE Transactions on software engineering*, vol. 30, no. 2, pp. 126–139, 2004.

[S205] A. Abadi, R. Ettinger, and Y. A. Feldman, "Fine slicing," in *International Conference on Fundamental Approaches to Software Engineering*. Springer, 2012, pp. 471–485.

[S206] M. Lillack, C. Bucholdt, and D. Schilling, "Detection of code clones in software generators," in *Proceedings of the 6th International Workshop on Feature-Oriented Software Development*, 2014, pp. 37–44.

[S207] H. M. Sneed and K. Erdoes, "Migrating as400-cobol to java: a report from the field," in *2013 17th European Conference on Software Maintenance and Reengineering*. IEEE, 2013, pp. 231–240.

[S208] M. K. Smith and T. Laszewski, "Modernization case study: Italian ministry of instruction, university, and research," in *Information Systems Transformation*. Elsevier, 2010, pp. 171–191.

[S209] T. Hatano and A. Matsuo, "Removing code clones from industrial systems using compiler directives," in *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. IEEE, 2017, pp. 336–345.

[S210] T. Gerlitz, Q. M. Tran, and C. Dziobek, "Detection and handling of model smells for matlab/simulink models." in *MASE@ MoDELS*, 2015, pp. 13–22.

[S211] Z. Zhao, X. Li, L. He, C. Wu, and J. K. Hedrick, "Estimation of torques transmitted by twin-clutch of dry dual-clutch transmission during vehicle's launching process," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 6, pp. 4727–4741, 2016.

[S212] K. Aishwarya, R. Ramesh, P. M. Sobarad, and V. Singh, "Lossy image compression using svd coding algorithm," in *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. IEEE, 2016, pp. 1384–1389.

[S213] S. Schlesinger, P. Herber, T. Göthel, and S. Glesner, "Proving correctness of refactorings for hybrid simulink models with control flow," in *International Workshop on Design, Modeling, and Evaluation of Cyber Physical Systems*. Springer, 2016, pp. 71–86.

[S214] S. Makka and B. Sagar, "Simulation of a model for refactoring approach for parallelism using parallel computing tool box," in *Proceedings of First International Conference on Information and Communication Technology for Intelligent Systems: Volume 2*. Springer, 2016, pp. 77–84.

[S215] V. Pantelic, S. Postma, M. Lawford, M. Jaskolka, B. Mackenzie, A. Korobkine, M. Bender, J. Ong, G. Marks, and A. Wassyng, "Software engineering practices and simulink: bridging the gap," *International Journal on Software Tools for Technology Transfer*, vol. 20, no. 1, pp. 95–117, 2018.

[S216] H. Zhu, Y. Yu, W. Qi, S. Liu, Y. Weng, T. Yuan, and H. Li, "The research on fault restoration and refactoring for active distribution network," in *2019 Chinese Automation Congress (CAC)*. IEEE, 2019, pp. 4470–4474.

[S217] V. N. Leonenko, N. V. Pertsev, and M. Artzrouni, "Using high performance algorithms for the hybrid simulation of disease dynamics on cpu and gpu." in *ICCS*, 2015, pp. 150–159.

[S218] S. Tichelaar, S. Ducasse, S. Demeyer, and O. Nierstrasz, "A meta-model for language-independent refactoring," in *Proceedings International Symposium on Principles of Software Evolution*. IEEE, 2000, pp. 154–164.

[S219] T. M. T. T. F. Munoz, "Beyond the refactoring browser: Advanced tool support for software refactoring," 2003.

[S220] A. Garrido and R. Johnson, "Challenges of refactoring c programs," in *Proceedings of the international workshop on Principles of software evolution*, 2002, pp. 6–14.

[S221] K. Mens and T. Tourwé, "Delving source code with formal concept analysis," *Computer Languages, Systems & Structures*, vol. 31, no. 3-4, pp. 183–197, 2005.

[S222] Y. Y. Lee, N. Chen, and R. E. Johnson, "Drag-and-drop refactoring: intuitive and efficient program transformation," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 23–32.

[S223] V. U. Gómez, A. Kellens, K. Gybels, and T. D'Hondt, "Experiments with pro-active declarative meta-programming," in *Proceedings of the International Workshop on Smalltalk Technologies*, 2009, pp. 68–76.

[S224] M. Unterholzner, "Improving refactoring tools in smalltalk using static type inference," *Science of Computer Programming*, vol. 96, pp. 70–83, 2014.

[S225] D. Vainsencher, "Mudpie: layers in the ball of mud," *Computer Languages, Systems & Structures*, vol. 30, no. 1-2, pp. 5–19, 2004.

[S226] O. Callaú, R. Robbes, É. Tanter, D. Röthlisberger, and A. Bergel, "On the use of type predicates in object-oriented software: The case of smalltalk," in *Proceedings of the 10th ACM Symposium on Dynamic languages*, 2014, pp. 135–146.

[S227] P. Tesone, G. Polito, L. Fabresse, N. Bouraqadi, and S. Ducasse, "Preserving instance state during refactorings in live environments," *Future Generation Computer Systems*, 2020.

[S228] V. Arnaoudova and C. Constantinides, "Adaptation of refactoring strategies to multiple axes of modularity: characteristics and criteria," in *2008 Sixth International Conference on Software Engineering Research, Management and Applications*. IEEE, 2008, pp. 105–114.

[S229] E. Rodrigues Jr, R. S. Durelli, R. W. de Bettio, L. Montecchi, and R. Terra, "Refactorings for replacing dynamic instructions with static ones: the case of ruby," in *Proceedings of the XXII Brazilian Symposium on Programming Languages*, 2018, pp. 59–66.

[S230] P. Sommerlad, G. Zgraggen, T. Corbat, and L. Felber, "Retaining comments when refactoring code," in *Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, 2008, pp. 653–662.

[S231] T. Corbat, L. Felber, M. Stocker, and P. Sommerlad, "Ruby refactoring plug-in for eclipse," in *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*, 2007, pp. 779–780.

[S232] R. Chen and H. Miao, "A selenium based approach to automatic test script generation for refactoring javascript code," in *2013 IEEE/ACIS 12th International Conference on Computer and Information Science (ICIS)*. IEEE, 2013, pp. 341–346.

[S233] H. V. Nguyen, H. A. Nguyen, T. T. Nguyen, and T. N. Nguyen, "Babelref: detection and renaming tool for cross-language program entities in dynamic web applications," in *2012 34th Interna-*

tional Conference on Software Engineering (ICSE). IEEE, 2012, pp. 1391–1394.

[S234] K. An and E. Tilevich, "D-goldilocks: Automatic redistribution of remote functionalities for performance and efficiency," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2020, pp. 251–260.

[S235] C.-Y. Hsieh, C. Le My, K. T. Ho, and Y. C. Cheng, "Identification and refactoring of exception handling code smells in javascript," *Journal of Internet Technology*, vol. 18, no. 6, pp. 1461–1471, 2017.

[S236] T. Mendes, M. T. Valente, and A. Hora, "Identifying utility functions in java and javascript," in *2016 X Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*. IEEE, 2016, pp. 121–130.

[S237] N. Van Es, Q. Stievenart, J. Nicolay, T. D'Hondt, and C. De Roover, "Implementing a performant scheme interpreter for the web in asm. js," *Computer Languages, Systems & Structures*, vol. 49, pp. 62–81, 2017.

[S238] L. Gong, M. Pradel, and K. Sen, "Jitprof: pinpointing jit-unfriendly javascript code," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 357–368.

[S239] A. M. Fard and A. Mesbah, "Jsnose: Detecting javascript code smells," in *2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2013, pp. 116–125.

[S240] C. Schuster, T. Disney, and C. Flanagan, "Macrofication: Refactoring by reverse macro expansion," in *European Symposium on Programming*. Springer, 2016, pp. 644–671.

[S241] J. Portner, J. Kerr, and B. Chu, "Moving target defense against cross-site scripting attacks (position paper)," in *International Symposium on Foundations and Practice of Security*. Springer, 2014, pp. 85–91.

[S242] G. Ortiz, J. A. Caravaca, A. García-de Prado, J. Boubeta-Puig *et al.*, "Real-time context-aware microservice architecture for predictive analytics and smart decision-making," *IEEE Access*, vol. 7, pp. 183 177–183 194, 2019.

[S243] M. U. Khan, M. Z. Iqbal, and S. Ali, "A heuristic-based approach to refactor crosscutting behaviors in uml state machines," in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 557–560.

[S244] R. Terra, M. T. Valente, and N. Anquetil, "A lightweight re-modularization process based on structural similarity," in *2016 X Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*. IEEE, 2016, pp. 111–120.

[S245] M. Bialy, M. Lawford, V. Pantelic, and A. Wassyng, "A methodology for the simplification of tabular designs in model-based development," in *2015 IEEE/ACM 3rd FME Workshop on Formal Methods in Software Engineering*. IEEE, 2015, pp. 47–53.

[S246] P. Langer, M. Wimmer, P. Brosch, M. Herrmannsdörfer, M. Seidl, K. Wieland, and G. Kappel, "A posteriori operation detection in evolving software models," *Journal of Systems and Software*, vol. 86, no. 2, pp. 551–566, 2013.

[S247] A. T. Sampson, J. M. Bjorndalen, and P. S. Andrews, "Birds on the wall: Distributing a process-oriented simulation," in *2009 IEEE Congress on Evolutionary Computation*. IEEE, 2009, pp. 225–231.

[S248] Y. Wang, H. Yu, Z. Zhu, W. Zhang, and Y. Zhao, "Automatic software refactoring via weighted clustering in method-level networks," *IEEE Transactions on Software Engineering*, vol. 44, no. 3, pp. 202–236, 2017.

[S249] A. Ouni, M. Kessentini, M. Ó Cinnéide, H. Sahraoui, K. Deb, and K. Inoue, "More: A multi-objective refactoring recommendation approach to introducing design patterns and fixing code smells," *Journal of Software: Evolution and Process*, vol. 29, no. 5, p. e1843, 2017.

[S250] H. Wang, M. Kessentini, and A. Ouni, "Bi-level identification of web service defects," in *International Conference on Service-Oriented Computing*. Springer, Cham, 2016, pp. 352–368.

[S251] A. Ghannem, G. El Boussaidi, and M. Kessentini, "On the use of design defect examples to detect model refactoring opportunities," *Software Quality Journal*, vol. 24, no. 4, pp. 947–965, 2016.

[S252] B. Amal, M. Kessentini, S. Bechikh, J. Dea, and L. B. Said, "On the use of machine learning and search-based software engineering for ill-defined fitness function: a case study on software refactoring," in *International Symposium on Search Based Software Engineering*. Springer, Cham, 2014, pp. 31–45.

[S253] M. Kessentini, A. Ouni, P. Langer, M. Wimmer, and S. Bechikh, "Search-based metamodel matching with structural and syntactic measures," *Journal of Systems and Software*, vol. 97, pp. 1–14, 2014.

[S254] M. Kessentini, R. Mahaouachi, and K. Ghedira, "What you like in design use to correct bad-smells," *Software Quality Journal*, vol. 21, no. 4, pp. 551–571, 2013.

[S255] A. Ghannem, M. Kessentini, and G. El Boussaidi, "Detecting model refactoring opportunities using heuristic search," in *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research*, 2011, pp. 175–187.

[S256] M. Boussaa, W. Kessentini, M. Kessentini, S. Bechikh, and S. B. Chikha, "Competitive coevolutionary code-smells detection," in *International Symposium on Search Based Software Engineering*. Springer, Berlin, Heidelberg, 2013, pp. 50–65.

[S257] E. Erturk and E. A. Sezer, "A comparison of some soft computing methods for software fault prediction," *Expert systems with applications*, vol. 42, no. 4, pp. 1872–1879, 2015.

[S258] C. S. Melo, M. M. L. da Cruz, A. D. F. Martins, T. Matos, J. M. da Silva Monteiro Filho, and J. de Castro Machado, "A practical guide to support change-proneness prediction," 2019.

[S259] L. Kumar, S. M. Satapathy, and A. Krishna, "Application of smote and lssvm with various kernels for predicting refactoring at method level," in *International Conference on Neural Information Processing*. Springer, 2018, pp. 150–161.

[S260] R. Hill and J. Rideout, "Automatic method completion," in *Proceedings. 19th International Conference on Automated Software Engineering, 2004.* IEEE, 2004, pp. 228–235.

[S261] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies, "Automatic query reformulations for text retrieval in software engineering," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 842–851.

[S262] G. M. Ubayawardana and D. D. Karunaratna, "Bug prediction model using code smells," in *2018 18th International Conference on Advances in ICT for Emerging Regions (ICTer)*. IEEE, 2018, pp. 70–77.

[S263] Z. Aliyu, L. A. Rahim, and E. E. Mustapha, "A combine usability framework for imcat evaluation," in *2014 International Conference on Computer and Information Sciences (ICCOINS)*. IEEE, 2014, pp. 1–5.

[S264] A. Herranz and J. J. Moreno-Navarro, "Formal extreme (and extremely formal) programming," in *International Conference on Extreme Programming and Agile Processes in Software Engineering*. Springer, 2003, pp. 88–96.

[S265] L. Quan, Q. Zongyan, and Z. Liu, "Formal use of design patterns and refactoring," in *International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*. Springer, 2008, pp. 323–338.

[S266] J. W. Ko and Y. J. Song, "Graph based model transformation verification using mapping patterns and graph comparison algorithm," *International Journal of Advancements in Computing Technology*, vol. 4, no. 8, 2012.

[S267] T. Ruhroth and H. Wehrheim, "Model evolution and refinement," *Science of Computer Programming*, vol. 77, no. 3, pp. 270–289, 2012.

[S268] S. Stepney, F. Polack, and I. Toyn, "Patterns to guide practical refactoring: examples targetting promotion in z," in *International Conference of B and Z Users*. Springer, 2003, pp. 20–39.

[S269] T. v. Enckevort, "Refactoring uml models: using openarchitectureware to measure uml model quality and perform pattern matching on uml models with ocl queries," in *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, 2009, pp. 635–646.

[S270] D. Luciv, D. Koznov, H. A. Basit, and A. N. Terekhov, "On fuzzy repetitions detection in documentation reuse," *Programming and Computer Software*, vol. 42, no. 4, pp. 216–224, 2016.

[S271] D. Arcelli, V. Cortellessa, and C. Trubiani, "Performance-based software model refactoring in fuzzy contexts," in *International Conference on Fundamental Approaches to Software Engineering*. Springer, 2015, pp. 149–164.

[S272] C. Wang and S. Kang, "Adfl: An improved algorithm for american fuzzy lop in fuzz testing," in *International Conference on Cloud Computing and Security*. Springer, 2018, pp. 27–36.

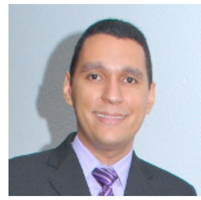
[S273] P. Lerthathairat and N. Prompoon, "An approach for source code classification using software metrics and fuzzy logic to improve code quality with refactoring techniques," in *International Conference on Software Engineering and Computer Systems*. Springer, 2011, pp. 478–492.

[S274] Z. Avdagic, D. Boskovic, and A. Delic, "Code evaluation using fuzzy logic," in *Proceedings of the 9th WSEAS International Conference on Fuzzy Systems*. World Scientific and Engineering Academy and Society (WSEAS), 2008, pp. 20–25.
[S275] H. Liu, J. Jin, Z. Xu, Y. Bu, Y. Zou, and L. Zhang, "Deep learning based code smell detection," *IEEE Transactions on Software Engineering*, 2019.
[S276] Y. Wang, "What motivate software engineers to refactor source code? evidences from professional developers," in *2009 IEEE International Conference on Software Maintenance*. IEEE, 2009, pp. 413–416.
[S277] J. Grigera, A. Garrido, and G. Rossi, "Kobold: web usability as a service," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 990–995.
[S278] M. W. Mkaouer, M. Kessentini, S. Bechikh, K. Deb, and M. Ó Cinnéide, "Recommendation system for software refactoring using innovization and interactive dynamic optimization," in *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, 2014, pp. 331–336.
[S279] V. Alizadeh and M. Kessentini, "Reducing interactive refactoring effort via clustering-based multi-objective search," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 464–474.

Chaima Abid is currently a PhD student in the intelligent Software Engineering group at the University of Michigan. Her PhD project is concerned with the application of intelligent search and machine learning in different areas such as web services, refactoring and security. Her current research interests are Search-Based Software Engineering, web services, refactoring, security, data analytics and software quality.

Vahid Alizadeh is currently a Ph.D. student in the intelligent Software Engineering group at the University of Michigan. His Ph.D. project is concerned with the application of intelligent search and machine learning in different software engineering areas such as refactoring, testing, and documentation. His current research interests are Search-Based Software Engineering, Refactoring, Artificial Intelligence, data analytics and software quality.

Marouane Kessentini is a recipient of the prestigious 2018 President of Tunisia distinguished research award, the University distinguished teaching award, the University distinguished digital education award, the College of Engineering and Computer Science distinguished research award, 4 best paper awards, and his AI-based software refactoring invention, licensed and deployed by industrial partners, is selected as one of the Top 8 inventions at the University of Michigan for 2018 (including the three campuses), among over 500 inventions, by the UM Technology Transfer Office. He is currently a tenured associate professor and leading a research group on Software Engineering Intelligence. Prior to joining UM in 2013, He received his Ph.D. from the University of Montreal in Canada in 2012. He received several grants from both industry and federal agencies and published over 110 papers in top journals and conferences. He has several collaborations with industry on the use of computational search, machine learning and evolutionary algorithms to address software engineering and services computing problems.



Thiago do Nascimento Ferreira is a Post-doctoral Researcher at University of Michigan-Dearborn under the supervision of Dr. Marouane Kessentini in the ISELab. He received my PhD Degree in Computer Science from the Federal University of Parana in 2019. His research mainly focuses on the use of Preference and Search Based Software Engineering to address several software engineering problems such as Software Testing and Software Refactoring.

Danny Dig is an associate professor of computer science at the University of Colorado, and an adjunct professor at University of Illinois and Oregon State. He successfully pioneered interactive program transformations by opening the field of refactoring in cutting-edge domains including mobile, concurrency and parallelism, component-based, testing, and end-user programming. He earned his Ph.D. from the University of Illinois at Urbana-Champaign where his research won the best Ph.D. dissertation award, and the First Prize at the ACM Student Research Competition Grand Finals. He did a postdoc at MIT. He (co-)authored 50+ journal and conference papers that appeared in top places in SE/PL. According to Google Scholar his publications have been cited 4000+ times. His research was recognized with 8 best paper awards at the flagship and top conferences in SE, 4 award runner-ups, and 1 most influential paper award (N-10 years) at ICSME'15. He received the NSF CAREER award, the Google Faculty Research Award (twice), and the Microsoft Software Engineering Innovation Award (twice). He released 9 software systems, among them the world's first open-source refactoring tool. Some of the techniques he developed are shipping with the official release of the popular Eclipse, NetBeans, and Visual Studio development environments (of which Eclipse alone had more than 14M downloads in 2014).