5-4-2018

Advances and Challenges in Software Refactoring: A Tertiary Systematic Literature Review

Mazen Alotaibi
mfa2886@rit.edu

ROCHESTER INSTITUTE OF TECHNOLOGY

MASTER'S THESIS

Advances and Challenges in Software Refactoring: A Tertiary Systematic Literature Review

---

Author:
Mazen ALOTAIBI

Supervisor:
Dr. Mohamed Wiem MKAOUER

A thesis submitted in fulfillment of the requirements
for the degree of Master of Science in Software Engineering

in the

Department of Software Engineering
B. Thomas Golisano College of Computing and Information Sciences

May 4, 2018

The thesis "Advances and Challenges in Software Refactoring: A Tertiary Systematic Literature Review " by Mazen ALOTAIBI, has been examined and approved by the following Examination Committee:

Dr. Mohamed Wiem Mkaouer

Assistant Professor

Thesis Committee Chair

Dr. Christian Newman

Assistant Professor

Dr. Yasmine El-Glaly

Lecturer

Dr. Scott Hawker

Associate Professor

Graduate Program Director

Acknowledgements

I would like to express my special thanks of gratitude to my adviser, Dr. Mohamed Wiem Mkaouer, for the invaluable support, comments, and suggestions he gave me throughout this thesis. From the start, his endless enthusiasm encouraged me and energized me after every meeting. His guidance, assistance, and expertise were indispensable to me, and without him, this thesis would have been impossible.

I would like to thank Dr. Christian Newman for all the advice and feedback he provided on my thesis.

I would like to thank Dr. Yasmine El-Glaly for her feedback on my thesis and on the inserted, she shewed on my research.

I would like to thank everyone, who supported me in making this study possible.

Finally, I would like to thank my parents for providing my needs and let me travel half the world to pursue my dreams. They offer me endless support and inspire me throughout my entire academic career, and for never once failing to remind me that they believed in me.

Rochester Institute of Technology

Abstract

Department of Software Engineering

B. Thomas Golisano College of Computing and Information Sciences

Master of Science in Software Engineering

Advances and Challenges in Software Refactoring: A Tertiary Systematic Literature Review

by Mazen ALOTAIBI

Software refactoring is one of the most critical aspects of software maintenance. It improves the quality of the software, reduces potential occurrence of bugs and keeps the code easier to maintain, extend and read. The process of refactoring supports and enables the developers to improve the design of software without changing the behavior. However, the automation of this process is complex for developers and software engineers since it is subjective, time and resource consuming. In this context, many literature reviews have analyzed the existing effort made by researchers to facilitate refactoring, as a core software engineering practice. This paper, aims in integrating all the existing research outcomes by performing a tertiary study on all the secondary studies, done in the area of refactoring. Based on our analysis we notice that there are many area of software refactoring that are under studied. As an outcome of this review, several classifications of existing studies were provided to showcase all the studies targeting the automation of refactoring along with explaining what metrics and objectives were used as means to drive refactoring and how it was assessed. This thesis also aims in unveiling areas of future directions for the research community in order to consolidate their efforts in improving the refactoring as a practice.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Refactoring is the process of restructuring the internal structure of the software without changing its external behaver to improve its quality[1]. Since refactoring is a vital task in software engineering in general, and in software maintenance in particular, it has been the focus of much research. Various studies look into how to recommend code changes to idealize and correct existing models, in order to support software developers with handling the increasing complexity of today's designs of software systems. That's why, refactoring is considered one of the most important practices of software evolution. Yet, this practice tends to be complex, subjective and manually demanding. Many primary studies have been conducted to automate and improve this process, and many secondary studies have analyzed the trends among the primary studies. In this thesis, we plan on conducting a tertiary study to connect the existing knowledge of previous secondary studies and present a more summarized overview of all the concepts and methodologies that researches and practitioners have been following when automating and recommending refactorings.

Based on our search for literature reviews, surveys, and systematic mappings in refactoring, we resulted with selecting 10 secondary studies in different area of refactoring to review. Since all our selected papers are secondary studies, we followed the practices of performing a tertiary study. A Tertiary study is a systematic review of systematic reviews, that helps in uncovering the

missing area of current research and answer wider research questions [2]. A tertiary study follow the same methodology as in a Systematic Literature Review (SLR). An SLR is a well-known methodology to analyze existing studies on a specific research area. The means of SLR is to analyze research papers, also known as Primary Studies (PS), to gather information about a specific topic in software engineering or answering research questions[3]. SLR studies also known as secondary studies which is the review of primary studies. SLRs can be conducted in many forms, for example, surveys, Systematic Mapping (SM) and Systematic Literature Reviews (SLR) studies[4].

The following thesis was conducted because there is no existing tertiary study in the area of refactoring, and because I believe that there is a need of gathering the current advances in the knowledge of refactoring along with exposing the current challenges and future directions of this area. Such a study will be helpful to researchers and practitioners who want to know what out there in the area of software refactoring.

Chapter 2

Background and related work

As a part of our related work, we follow a similar approach as [4] in conducting related work, where we analyzed other tertiary studies in the field of software engineering. This helps in capturing what areas of software engineering these existing studies have been covering. Based on our knowledge and search we did not find any tertiary study in the are of software refactoring. Nevertheless, we found an overall of 14 tertiary studies [3]–[16] in various areas related to testing, design, requirements, etc. Our search results for tertiary studies in SE are shown in Table 2.1, which are sorted by the year of publication. Since these tertiary studies have similar research setting, we review them briefly.

Based on our findings, the first tertiary study was introduced in 2009 by Kitchenham. Where the authors have thought of building a literature review to asses the other existing literature reviews [5]. They applied the concept of evidence-based software engineering (EBSE) by performing systematic literature review to assess the impact of other SLRs in software engineering. Kitchenham et al. reviewed 20 relevant studies as their primary studies set, where they analyzed the quality of their set of primary studies and have found that the quality of SLRs is improving and more researchers are becoming interested in SLR [5]. The same paper [5] was updated a year later[3]. The updated paper has 35 additional SLRs on a broad automated search [3]. The paper concluded that the quality of SLRs published in conference and workshop has improved and more and more researchers are using SLR guidelines that they advocated for in their initial papers

TABLE 2.1: List of systematic review of systematic reviews in software engineering.

#	Topic	Number of secondary studies	Year	Rfe.
1	SLRs in SE – A SLR	20	2009	[5]
2	SLRs in SE– A tertiary study	33	2010	[3]
3	Critical appraisal of SLRs in SE from the perspective of the research questions	53	2010	[6]
4	Research synthesis in SE-A tertiary study	49	2011	[7]
5	Six years of SLRs in SE-An updated tertiary study	67	2011	[8]
6	Signs of Agile Trends in Global SE Research-A Tertiary Study	12	2011	[9]
7	Systematic approach for identifying relevant studies in SE	38	2011	[10]
8	SLRs in Distributed SE-A Tertiary Study	14	2012	[11]
9	A tertiary study: experiences of conducting SLRs in SE	116	2013	[12]
10	A SR of systematic review process research in software engineering	68	2013	[13]
11	Risks and risk mitigation in global software development: A tertiary study	37	2014	[14]
12	SR in requirements engineering: A tertiary study	53	2014	[15]
13	A SLR of literature reviews in software testing	101	2016	[4]
14	SLR in agile software development:A tertiary study	28	2017	[16]

[3]. Another update of the two previous studies [3] was reported in[8]. The updated study found 67 new SLRs where the authors of the paper concluded that software engineering community is starting to adapt SLRs as a research method.

A critical appraisal of SLR in SE from the perspective of the research questions asked in the reviews study was proposed by Silva and Santos [6]. Silva and Santos work analyzed 53 literature reviews that had been gathered in two previous tertiary studies [3], [5]. The study found that over 65% of the research questions were exploratory questions. However, 15% of the questions were casual questions.

The better understanding of types, methods and challenges in synthesizing software engineering research and implication for the progress of research and practices was the objective of [7]. The study analyzed 49 reviews and more then half did not include any research synthesis.

Another study [9] investigated the role of Agile trends in global/distributed software engineering (GSE) research. The study reported that, despite recent beliefs that agile and global are two incoherent, global agile development has become more and more accepted. The paper concluded that there are signs that both globalaization and "agilization" of software companies are stable trends for the future, but there is a strong need for future studies in this area of research.

The article in [10] provides a systematic approach to design, execute, and evaluate search strategy to retrieve literature from digital libraries. The authors claimed that the search strategy is one of the critical steps in conducting SLRs because this step is time-consuming and error-prone.

The tertiary study in [11] reviewed SLRs in Distributed Software Development (DSD) to create a catalog reference of SLRs in DSD area. The study analyzed 14 SLRs, seven address aspects of managing distributed development. Four SLRs addressed topics of the engineering process, and the rest of SLRs were related to requirement, design, and software engineering education in the topic of distributed software development.

The authors of [12] conduct a tertiary study to report the experiences of conducting an SLR for the benefit of new researchers. The study has gathered 116 studies that have implicitly or explicitly reported their experiences in conducting SLRs in software engineering.

A Systematic review (SR) reported in [13] that is discussing the problem with SRs methodology and suggesting for an improvement of the SRs methodology. The paper analyzed the 2007 guideline for SRs in software engineering and belief its time to update the guideline. The study has identified 68 papers from 2005 to mid-2012 and recommend removing advice to use structured quotations to contract search string for the SRs.

The tertiary study reported in [14] conduct and SLR in risk and risk mitigation in global/distributed software development.

An SRs in requirements engineering (RE) [15] is the first tertiary study that fully focused on published SLRs in RE. The study has conducted an automated and manual search of RE and related SLRs. The authors of the study have notice that the quality of SLRs in RE has been decreasing in the recent years and the authors stated there is strong need to replicate these SLR and increase there quality.

An SLR of literature reviews in software testing [4] performed a systematic map study to secondary studies in software testing topic. The paper analyzed 101 studies between 1994 and 2015 in the area of software testing. The paper found that there is a lack of secondary studies in many important sub-areas of software testing, e.g., test management, the role of product risk in testing, human factors in software testing, etc.

The tertiary study in [16] provides an overview of SLRs on agile software development (ASD). The study looks at 28 SLRs and found that ten of the SLRs that studying ASD focus on: adaptation, methods, practices, human and social aspects, CMMI, usability, global software engineering (GSE), organizational agility, embedded systems, and software product line engineering as research area.

Chapter 3

Research Methodology

This tertiary study follows the same research methodology as advocated by Kitchenham et al. [2]. To conduct this tertiary study, we have the choice of performing an SLR or SM. Kitchenham et al.[2] presented detailed guidelines on how to conduct SLR in SE. While, Petersen et al.[17] provided guidelines to conduct SM. The guidelines provide insight into building and structuring classification schema. Although we are conducting an SLR, some of Petersen et al.[17] guidelines can be useful for our study, where we incorporated some techniques related to the assessment of the selected studies, from Petersen et al. with Kitchenham et al.[2] guidelines to perform an SLR.

3.1 GQM and Research questions

We followed Goal-Question-Metric (GQM) methodology [18] in developing our research questions. The goal of this study is to systematically review the current state-of-art literature in secondary studies in the area of software refactoring, to find out the current states of how researcher and practitioners identify software refactoring opportunities and how refactoring strategy have been applied and validated. We also want to grasp how research in software refactoring has been developed as a secondary study. Based on our goal, we raised 5 Research Questions (RQs) that investigate the area of software refactoring in this study:

- RQ1: What are the system levels that are covered by refactorings in the literature?
  Refactoring can be applied to many software artifacts (e.g., requirements, design model, etc.), but commonly refactoring applied to source code [19]. Answering this RQ will help us to learn which system levels are commonly covered by refactoring. Knowing such information will shed the light on what is the most studied software layer, and also what are the other software artifact that are under-studied in terms of refactoring.

- RQ2: What are the existing strategies to detect refactoring opportunities? How do researchers identify refactoring opportunities?
  This RQ presents various strategies used by research to detect refactoring opportunities. Initially, refactoring opportunities are usually identified by the level of quality in a software, and this quality is evaluated either through design metrics or code smells. Answering this allows us to understand the different ways that existing tools identify refactoring opportunities.

- RQ3: What are the existing strategies to automate the application of refactorings?
  Answering this RQ will help us know the existing mechanisms of refactoring execution automation. The degree of automation of refactoring tool can be classified into: Fully-automated, Semi-automated, and Manual. A fully-automated tool is stable to apply the detection opportunities and the execution of refactoring strategies to software artifact without any user interaction. Semi-automated refactoring tools require some user interaction, either by suggesting the changes to the user, or either through providing the user the possibility to express a preference during the generation of refactorings. The Manual refactoring tools rely on the user's expertise to decide about the refactoring opportunity and they just automate the execution of refactoring i.e., they just automate the decisions made by the user and they also ensure the behavior preservation of the refactored system [20].

- RQ4: How do approaches verify the correctness of refactorings? How do they test the behavior preservation?

Answering this RQ will give us insight on how refactoring techniques have been validated in research. Do studies verify the correctness of refactoring by techniques using qualitative analysis and through human validation? or through only static analysis? or dynamic analysis?

- RQ5: How refactoring strategies have been validated?

  This research question will uncover how refactoring approaches have been validated in literature, and what are the datasets used in the evaluation process. Knowing such information will point out what validation strategy needs to be improved.

3.2 Primary studies selection

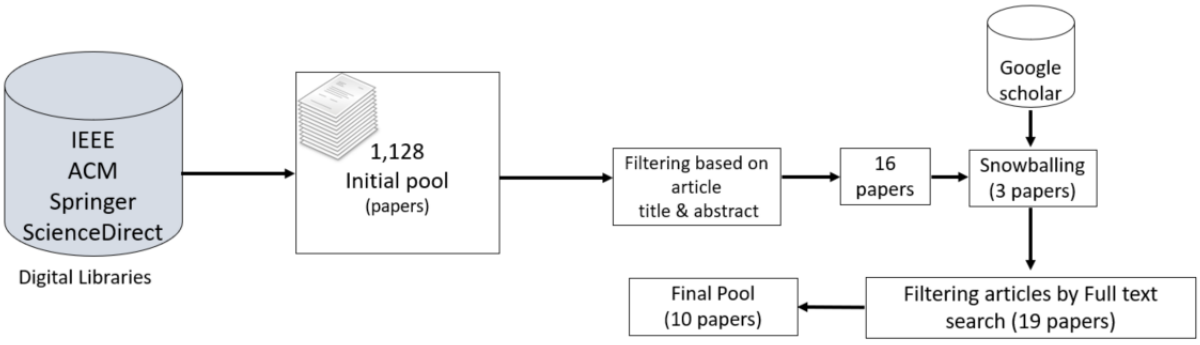

FIGURE 3.1: Literature Search Process

Based on the process that was followed by this study (Fig. 3.1), the first phase was identifying digital libraries and constructing the search phrase. In this step we followed the following steps in order:

- Digital libraries selection and search keywords.
- Inclusion and exclusion criteria.
- Final pool of primary studies.

TABLE 3.1: Number of studies per Database

Database	Search result
IEEE	338
ACM	326
ScienceDirect	231
Springer	233
Total	1128

- Quality assessment.

3.2.1 Digital libraries selection and search keywords

According to the guideline for performing an SLR [2], to find primary studies, we conducted our search in four major digital libraries: ACM DigitalLibrary, IEEE Xplore, SpringerLink and ScienceDirect. Our search keywords for the digital libraries:

```
((refactor*) AND (survey OR review OR mapping))
```

We search the digital libraries by the above-mentioned keywords, where we limited the search in articles title by variation of the word refactor and a full-text search by one of these words survey, review or mapping. Our search terms have been modified to fit the capability of digital libraries search engines.

The initial search resulted in 1128 papers, Table 3.1 shows the number of papers per digital libraries. Moreover, we performed the snowballing (forward and backward)[21] to reduce the risk of missing any studies, where we found 3 more studies as shown in Fig. 3.1. The search for related articles conducted until January 2018 without specifying a time limit.

3.2.2 Inclusion and exclusion criteria

We included articles based on the following criteria:

- Articles must be in the form of: SLRs, SMs, or Surveys.
- Articles must be related to software refactoring topics.
- Articles must be written in English.

We excluded articles based on:

- Master studies that are not published in the listed conferences or journals.
- Short and tool papers were excluded.
- Articles that not fully focused on refactoring as a central topic.

3.2.3 Final pool of primary studies

After applying inclusion and exclusion criteria, our final pool resulted in 10 secondary studies (also called primary studies). We assess the quality of the studies and extract data from the studies based on our research questions. We will be referring to the secondary studies by these labels [SSX], where X represents the paper ID, throughout the paper. Appendix A.1. contains a full list of the primary studies.

3.2.4 Quality assessment

Each primary study was assessed using the same set of quality criteria that have been adopted by many research studies (e.g., by Kitchenham) in tertiary studies [4], [16]. The quality criteria were defined by the Center for Reviews and Dissemination (CDR) Database of Abstracts of Reviews of Effects (DARE) in the York University [22]. Also, we added the last question [23] to quality criteria for assessing the aim of each primary study. Our quality assessment criteria are composed of five questions:

1. Are the review's inclusion and exclusion criteria described and appropriate?
2. Is the literature search likely to have covered all relevant studies?

TABLE 3.2: Primary studies quality assessment.

Study ID	Q1	Q2	Q3	Q4	Q5	Paper score
[SS1]	1	1	0	1	1	4
[SS2]	0	1	0.5	1	1	3.5
[SS3]	0	0.5	0.5	1	1	3
[SS4]	1	0.5	1	1	1	4.5
[SS5]	1	1	0.5	1	1	4.5
[SS6]	1	1	0	1	1	4
[SS7]	1	1	1	0.5	1	4.5
[SS8]	1	1	1	0.5	1	4.5
[SS9]	1	1	1	0.5	1	4.5
[SS10]	1	1	1	0.5	1	4.5

3. Did the reviewers assess the quality/validity of the included studies?

4. Were the primary data/studies adequately described?

5. Is there a clear statement of the aims of the study?

Each question was scored by the same scoring scale that was proposed by Kitchenham et al. [5]. All primary studies on our pool were evaluated by calculating the quality, by assessing the score of {0, 0.5, 1} to each question and adding them up to get the overall score of a primary study.

3.3 Data extraction

We reviewed each study in our pool with a focus on each RQ and extracted the required information from primary studies to answer our RQs. Additionally, we extracted all meta-data information (bibliography) from all secondary studies as shown in Table 3.3. Also, we classify papers based on their types (e.g., SLRs, surveys, or SMs) of each primary studies in our pool, also, we look at the secondary studies that were reviewed by each SLR in our pool.

We handled data extracting activity by creating an online spreadsheet on Google Doc. The spreadsheet has all the detailed information about the primary studies such as bibliography, type of secondary study, quality assessment and our RQs with their answer per primary studies.

TABLE 3.3: Primary Studies bibliography info.

Study ID	Authors	Paper title	Publication year	Venue
[SS1]	Michael Mohan, Des Greer	A survey of search-based refactoring for software maintenance	2018	Journal of Software Engineering Research and Development
[SS2]	Tom Mens, Member, IEEE, Tom Tourwe	A Survey of Software Refactoring	2004	IEEE Transactions on Software Engineering
[SS3]	Outi Räihä	A survey on search-based software design	2010	Computer Science Review
[SS4]	Satwinder Singh, Sharanpreet Kaur	A systematic literature review: Refactoring for disclosing code smells in object oriented software	2017	Ain Shams Engineering Journal
[SS5]	Miguel A. Laguna, Yania Crespo	A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring	2013	Science of Computer Programming
[SS6]	Thainá Mariani, Silvia Regina Vergilio	A systematic review on search-based refactoring	2017	Information and Software Technology
[SS7]	Jehad Al Dallal, Anas Abdin	Empirical Evaluation of the Impact of Object-Oriented Code Refactoring on Quality Attributes: A Systematic Literature Review	2018	IEEE Transactions on Software Engineering
[SS8]	Jehad Al Dallal	Identifying refactoring opportunities in object-oriented code: A systematic literature review	2015	Information and Software Technology
[SS9]	Mesfin Abebe, Cheol-Jung Yoo	Trends, Opportunities and Challenges of Software Refactoring: A Systematic Literature Review	2014	International Journal of Software Engineering and Its Applications
[SS10]	Mohammed Misbhauddin, Mohammad Alshayeb	UML model refactoring: a systematic literature review	2013	Empir Software Eng

Chapter 4

Results

4.1 RQ1. What are the system levels that are covered by refactorings?

We classified all primary studies in our pool by system level that are covered by refactorings. The classification falls into four categories that are mostly studied by research: source code, model, product line and package. We noticed that the most studied type is source code, where all secondary studies in our pool except [SS10], were mainly focused on studying source code refactoring. In design model refactoring, we found out 6 studies have mentioned software model refactoring as part of their study to software refactoring. Although, the majority of primary studies on our pool have covered both source code and design model refactoring, [SS10] devoted only to studying design model refactoring. There was only one study devoted to understanding how the changes in requirements impact the changes in models and source code. The title and abstract of [SS5] give the intuition of a study in product line refactoring, but after reading the paper, we realized that it was focused on how product lines influence the design and implementation of existing

TABLE 4.1: Types of System refactoring per study

Type of system refactoring	Number of secondary studies	References
Source code	9	[SS1, SS2, SS3,SS4, SS5, SS6, SS7, SS8, SS9]
Model	6	[SS1, SS2, SS3, SS5, SS6, SS10]
Product line	1	[SS5]
Package	2	[SS1, SS5]

software systems. Table 4.1, show each primary studies per type of system levels that are covered by refactoring.

4.2 RQ2. What are the existing strategies to detect refactoring opportunities? How papers identify refactoring opportunities?

The first step in applying refactoring is to detect refactoring opportunities. Refactoring opportunities mainly can be detected in a software either by metrics or bad smells [19]. Most secondary studies in our pool stated that metrics and bad smells are the main drives to detect refactoring opportunities, except [SS3]. In [SS3], the process to detect refactoring opportunities is different than the traditional approaches that mentioned by other studies in our pool. The secondary study mentioned that fitness function is used to measure software quality after a refactoring solution has been applied randomly to a software. [SS5] is a systematic mapping study for re-engineering legacy system to software product line (SPL) mentioned that refactoring opportunities detected in SPL by metrics, usually quality metrics (e.g., COC, WMC, etc.), yet there are a specific metrics that are designed to use with SPL to detect refactoring opportunities (e.g., PrR, IPrR, etc.).

The secondary study [SS1] reported that the methods used to detect refactoring either by detecting bad smell or using quality metrics (fitness function) to refactor the software randomly. [SS2], [SS4] and [SS6] studies, majority of refactoring opportunities detected by identifying code smells and some by using object-oriented metrics (QMOOD). As for [SS4], the authors identified the most detected code smell based on their pool of studies: 1. Gad class. 2. blob. 3. feature envy. In [SS7] and [SS8], most used approaches to identify refactoring opportunities are based on quality metrics. [SS9] mentioned that refactorings are identified by the detection of bad smells and then the application of specific refactoring patterns.

The secondary study [SS10] has been gathering studies that detect different type system of

TABLE 4.2: Most detected bad smells in secondary studies

Type of bad smell	Smell Name	References
Class	Blob	[SS4, SS6, SS9, SS10]
	God Class	[SS4, SS9, SS10]
	Large Class	[SS4, SS9]
	Lazy Class	[SS6]
Method	Feature Envy	[SS4, SS6]
	Log Parameter List	[SS4, SS6]
	Functional Decomposition	[SS10]
	Log Method	[SS4]

level refactoring than other above-mentioned studies. The study investigated UML model refactoring. The paper identified three basic approaches to detect model smells which are Metrics-based approach, Pattern-based approach and Rule-based approach.

Since we identified what refactoring detection strategies that are mention in literature we also, identified the most detected bad smells in source code and UML model. Table 4.2 has the most studied bad smells in the application of refactoring.

4.2.1 Search-based refactoring

Several studies tackle the challenge of refactoring as an optimization problem. These studies constitute the state-of-art of Search-Based Software Engineering (SBSE). SBSE is defined as the application of search-based approaches to solving optimization problems in software engineering [24]. Once a software engineering task is framed as a search problem, there are numerous approaches that can be applied to solving that problem, from local searches such as exhaustive search and hill-climbing to meta-heuristic searches such as Genetic Algorithms (GAs) and ant colony optimization. Many contributions have been proposed for various problems, mainly in cost estimation, testing, and maintenance. According to Harman [25], SBSE methodology can be summarized in the following steps:

Solution representation: The formulation of a given SE problem is achieved by defining a possible solution representation that solves that problem.

Solution Evaluation: For multiple candidate solutions to the same problem, the evaluation of their quality is assessed by a fitness function, which can be defined by the degree of which, it is meeting the expected result for the problem.

Solution variation: In each search algorithm, the variation operators play the key role of moving candidate solutions within the search space with the aim of driving them towards optimal solutions. These recombination operators need to be defined respectfully to the solution presentation and their application should derive new solutions with eventually different fitness values. The deployed algorithm has the responsibility to conduct the search and evolve the candidate solutions until stopping criteria are being met.

Consequently, the tackled refactoring problems are presented in the primary studies according to the above mentioned steps. More particularly, the refactoring problem formulation shows a popularity in using a population-based, single and multi-objective optimization, where solutions are defined similarly to genes, their reproduction is maintained by crossover and mutation operators along with the repeated calculation of their fitness values to select the best solutions and constitute the next generation. Through the generations, solutions are being guided in the search space using the problem's fitness functions until stopping criteria is being met, and a near optimum is found.

It is important to note that, Based on these SLRs, treating software engineering (SE) problems from a single-objective perspective is insufficient, as most SE problems are naturally complex in which many conflicting objectives need to be optimized. As a consequence, many newer refactoring studies has been formulated as multi/many objectives. Figure 4.1 has the most used search-based technique by secondary studies in the application of refactoring.

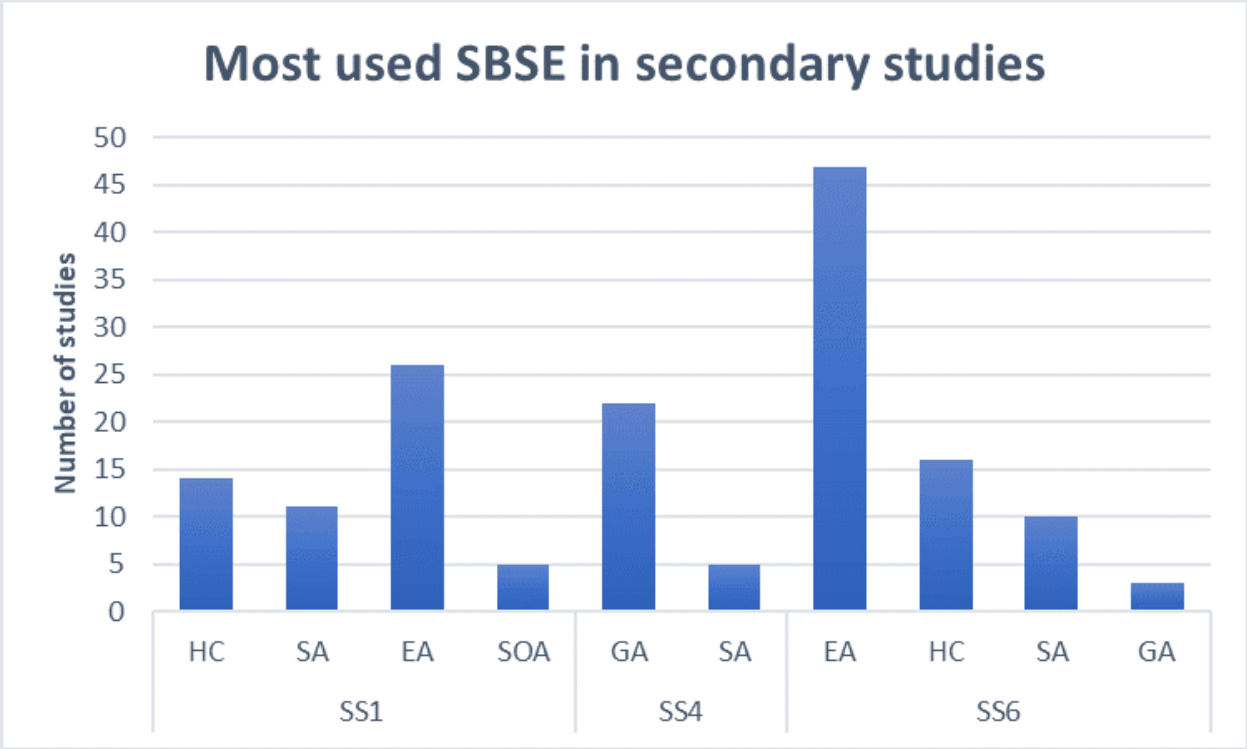

FIGURE 4.1: Most used search-based in secondary studies.[a]

[a] HC: Hill climbing, SA: Simulated Annealing, EA: Evolutionary algorithm, GA: Greedy Algorithm, SOA: Swarm optimization algorithms

4.3 RQ3. What are the existing strategies to automate the application of refactorings?

As we mentioned in the previous section the degree of automation in refactoring tools are generally classified to Fully, Semi and Manual automation. In the secondary studies [SS1, SS3] of search-based refactoring, [SS1] study identified 7 fully automation tools (e.g., DPT, TrueRefactor, etc.) in a pool of 50 studies about search-based refactoring. Where [SS3] present an overview of search based software engineering (SBSE) in designing software from requirements until the maintenance phase of a software. The study identified different approach for fully automating refactoring but did not mention the tools for applying these approaches. The secondary studies [SS2, SS5, SS6, SS7, SS8, SS9], identifies two level of automation, semi and fully in the application of refactoring. [SS4], has analyzed metrics-based refactoring approaches (semi-automated) mainly focus on tools like JDeodorant to apply refactoring. Where in [SS10], a secondary study that studies UML Model refactoring. The SLR identified 63 primary studies, 62% of these studies are supported by tools, 8 studies have fully automated tool for applying refactoring to UML Model, 24 studies are semi-automated and 6 studies are manual.

Based on our results we gather to answer RQ3, we notice that the most used approach in the application of refactoring is semi-automated. Although semi-automated approach required human interaction and consume time, it leaves the decision of applying refactoring to the developers.

We looked at all refactoring scenario that have been accounted in our pool and summarizing them into a table with their studies, Table A.5. The table is self-explanatory, it seems that refactoring activity for source code is the most studied one.

TABLE 4.3: Most used Metrics.

Type	metric	# PS	Layer	Measurement
Estimated External	Maintainability	2	Model/Source	Customized combinations of internal design metrics
	Reusability	5	Model/Source	QMOOD
	Understandability	5	Model/Source	QMOOD
	Flexibility	5	Model/Source	QMOOD
	Adaptability	2	Model/Source	Combination of internal metrics
	Testability	2	Model/Source	Combination of internal metrics
	Extensibility	5	Model/Source	QMOOD
	Effectiveness	5	Model/Source	QMOOD
	Completeness	2	Model/Source	Combination of internal metrics
	Functionality	5	Model/Source	QMOOD
	Modularity			Neighbors (i.e., "number of neighbor modules connected via dependencies"
Measured External	Reliability	2	Model/Source	# of defected classes, B-R, Bug Fix Rate, Bug Fix Time, Post Release Failures, # of faults
	Maintainability	2	Model/Source	# changed files, # check-ins, Analyzability, Change Entropy, Changeability, relative churn, total churn, # of changes, #changed delta, #changed lines, #lines in changed files

Continuation of Table 4.3

Type	Metric	# PS	Layer	Measurement
	Efficiency	2	Model/Source	Resource Utilization, Time Behavior
	Testability	3	Model/Source	re-test SLOC
Internal	Coupling	10	Model/Source	OCMEC, Afferent Coupling, Afferent Coupling (Ca), Aggregated import coupling, C, CBO, CC (Class Coupling), CCBC, CDBC, CCC, CF, Class Coupling, DAC, DAC2, DCC, Efferent Coupling, Export Coupling, Fan In, Fan Out, General Coupling, ICP, LD, MPC, NOCM, NR, Number of Parameters, RFC, SeCoupling, StCoupling, ATFD
	Cohesion	10	Model/Source	C3, CAAI, CAIW, CAM, CBMC, CMAI, CMW, Coh, Connectivity, coverage, DCD, DCI, ICBMC, ICH, LCC, LCCD, LCCI, LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, LSCC, MSC, Non-normalized Cohesion, Normalized Cohesion, OL2, overlap, PCCC, SCOM, SeCohesion, StCohesion, TCC, tightness, LAA, CC

Continuation of Table 4.3

Type	Metric	# PS	Layer	Measurement
	Complexity	9	Model/Source	CC (Cyclomatic Complexity), CDE, Classes in a Cycle, execution modular size, function parameters, immediate base classes, Lines of Code Per Class, Lines of Code Per Method, Max_Loc, Max_MCC, McCabe Per Method, McCabe Per Method (MVG), Member reads, Member writes, Method Size, MLOC, NOA, NOM, NPM, type declarations in local functions, WMC, AMW, ALCM
	Size	10	Model/Source	#blocks, #classes, #functions, #local variables, #parameters, AMS, ANA, Attributes Per Class, CIS, CS, DSC, Duplicated Code Blocks, JavaDoc Comment Blocks, LOC, NOM (in a program), Non-JavaDoc Comment Blocks, Number of Java Classes, Number of Static Methods (in a program), SLOC, CDP
	Inheritance	7	Model/Source	CAI, CMI, CSP, DIT, MFA, NOC, NOH
	Combination of attributes			Q, Q1, Q2, Q3, Entity Placement

Continuation of Table 4.3

Type	Metric	# PS	Layer	Measurement
	Composition	4	Source	MOA, CPCC
	Data encapsulation	4	Source	CCDA, CIDA, COA, DAM
	Polymorphism	4	Source	NOP
	Information hiding	2	Source	AHF, MHF

4.4 RQ4. How approaches verify the correctness of refactorings? How they test the behavior preservation?

Behavior preservation is a fundamental part of refactoring since it gives the complete sense of its definition. This explains how it was initially originated when refactoring where introduced by William Opdyke [26].

The original definition of behavior preservation presents the notion of preconditions. An example of refactoring precondition can be seen when considering extract class refactoring in which naming conflicts must be avoided. Other studies have introduced later the notion of postcondition to guarantee the success the refactoring execution. Even though, there were many studies specialized in better optimizing the definition and execution of pre/postconditions, none of the existing SLRs has explicitly focused on it.

The concept of preserving the behavior was vaguely mentioned as part of the refactoring definition, without going into the details of the possible approaches who discussed their efficiency and their classification. This is one of the main limitations that we encountered in these studies.

4.5 RQ5. How refactoring strategies have been validated?

We checked all studies in our pool for evaluation method to evaluate refactoring strategies. Majority of papers (7 out 10) do not show explicitly the evaluation method of refactoring. Other studies [SS6, SS8, SS10] have provided a section of the paper that have all evaluation method that has been gathered by the secondary studies. The kind of evaluation conducted on refactoring strategies mostly empirical and case studies. Usually, the evaluation method performed by paper either to answer research questions or to validate a hypothesis.

Since not that many papers have provided information about refactoring evaluation method, we decided to investigate each research question in each secondary studies. Where we classify each RQs based classification scheme proposed by [27] and adapted in [4]. Table A.4 contains the classification for RQs in secondary studies.

We also, checked the secondary studies for the most used software system to evaluate refactoring approaches. Figure 4.2, show number of studies that have been analyzed by each secondary studies. The figure compares software system among 5 secondary studies.

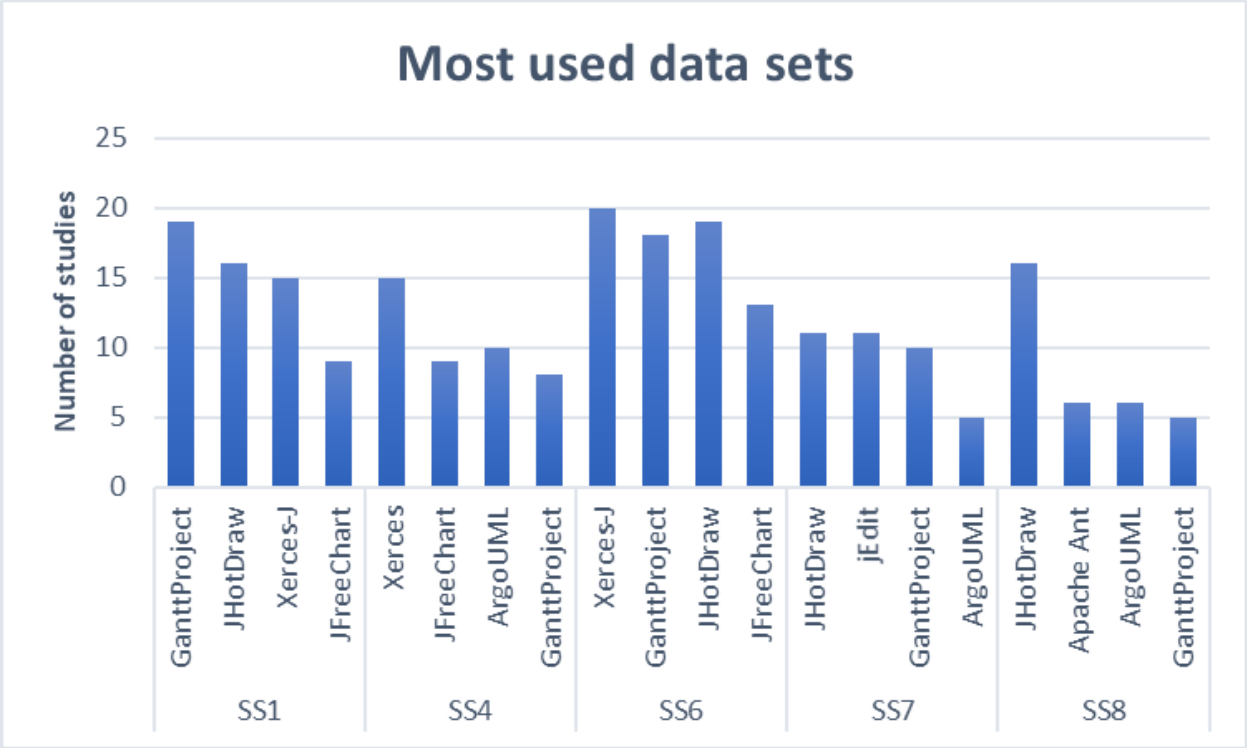

FIGURE 4.2: Most used data sets in secondary studies

Chapter 5

Discussions

In this tertiary study focusing on software refactoring, we analyzed 1128 related articles, 10 was selected as primary studies. Majority of these studies primarily studying source code refactoring. Only one study [SS10] that was focusing UML model refactoring. In this section, we will be discussing our finding based on our RQs.

- (RQ1) "What are the system levels that are covered by refactorings?" this is the first question we asked in this study. Based on the results we can clearly see that as we mentioned above that source code is the most studied one, in comparison to other type of system refactoring. There are two mainly reasons behind that, first refactoring by nature was made initially for source code. The second reason is that source code is the most practical aspect that developers deal with on daily basis.

- (RQ2) "What are the existing strategies to detect refactoring opportunities? How papers identify refactoring opportunities?" The results of this question show that two approaches to detect refactoring opportunities have been identified literature which are either metrics or bad smells. With exception of 3 studies, all other studies identify refactoring by metrics or bad smells detection. The three studies have another approach in detecting refactorings. Where search-based refactoring methods are implemented to tackle the different refactorings.

- (RQ3) "What are the existing strategies to automate the application of refactorings?"The results of this question show that semi-automated approach is the most used one in our pool of studies. The reason behind that is semi-automated give the developers a control over applying certain refactoring approach and ensure the refactoring will not change the behavioral of the software.

- (RQ4) "How approaches verify the correctness of refactorings? How they test the behavior preservation?" Behavior preservation is a fundamental part of refactoring that ensures the preservation of software behavior. Research in this area is lacking behind. Many studies that were mentioning the concept of Behavior preservation vaguely.

- (RQ5) "How refactoring strategies have been validated?" Based on the results we got in answering this question. It seems that not that many studies provide an evaluation method for the refactoring. This is one of the limitations we notice in these studies.

5.1 Threats to validity

Since this is a tertiary study, that is limited to the number of literature to be reviewed. There is a possibility of studies that have been missed out because there are not an SLR or survey. A main threats to validity to this study are the selection of search engine, inaccurate data extraction, possibility of missed studies due to the limitation of the search term, and researcher bias with the regard to include or exclude paper based on inclusion/exclusion criteria.

To limit any threats related to the construction of the study, i.e., research questions and their suitability. we used the Goal-Questions-Metrics approach better emphasize on the aim of this study along with how it is being answered through various research questions. We also linked metrics to each research questions. The used RQs were conceived to make a coverage of all the spectrum of software refactoring along with summarizing the existing results of the secondary

studies. Questions addressed according to specific metrics, like advocated in [3]. For the assessment of the secondary studies, we minimized the bias by performing two rounds of evaluation and the scores were assigned based on a voting system.

Chapter 6

Conclusion

In this thesis, we performed a tertiary study in the form of a systematic literature review on software refactoring. The survey contained 10 various existing SLRs. Based on results found, the thesis has shown the spectrum of refactoring application that targets researchers to better coin the existing areas in which refactoring has been advancing. To do so, we have defined 5 research questions that we answer throughout the thesis.

we have exposed all the techniques utilized to detect refactoring opportunities, our findings show the strong correlation to the use of design metrics in the form of their internal or external attributes to drive refactoring operations. Other studies have been faithful to the original definition of refactoring as a response to the existence of code smells. This body of knowledge has shown that the automation of refactorings tends to be challenging. Since the problem of generating the suitable set of refactoring operations is computationally expensive, several studies rely on the use of search-based algorithms to approximate near optimal solutions to refactoring their systems. Another important finding relies on the absence of explicit studies on how the behavior preservation of refactoring is being verified.

By reviewing existing studies, we summarized the knowledge of several studies targeting different layers of software, sub-locations, and given the evaluation methods of their refactoring.

Also, the thesis presented the bibliometry of all the primary studies included in the 10 analyzed studies. The thesis also followed the guidelines of assessing the existing studies and provided

guidelines on how to improve them.

6.1 Future Work

To summarize, this study has shown preliminary results of a systematic review that needs to be expanded by exploring with more details all the results of all the selected studies. We believe that combining their results will shed the light where the refactoring research community is falling short.

There are a number of future work that can be done after this tertiary study. For example, a replication of this study can be done later after there are a good amount of available secondary studies on software refactoring. Also, this study can be conducted in sub-area of software engineering e.g., software requirements, etc.

Appendix A

Appendix

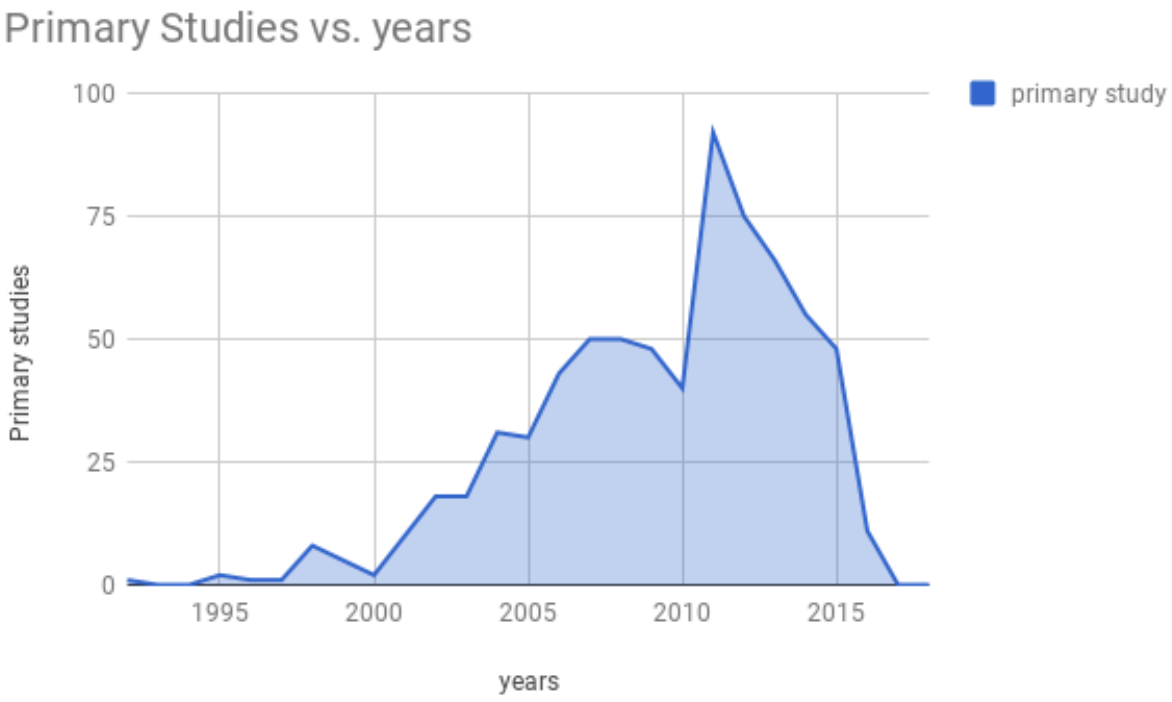

FIGURE A.1: Primary studies vs years

TABLE A.1: Top-3 cited secondary studies based on total number of citations.

References	Paper title	Study Type	Publication year	Total number of citations
[SS2]	A survey of Software Refactoring	Survey	2004	1267
[SS3]	A survey on search-based software design	Survey	2010	159
[SS5]	A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring	SM	2013	69

TABLE A.2: Type of secondary studies.

Paper Type	Number of studies per type	References
SLRs	6	[SS4, SS6, SS7, SS8, SS9, SS10]
Regular surveys	3	[SS1, SS2, SS3]
SMs	1	[SS5]

TABLE A.3: Secondary studies pool of papers.

Study ID	# of primary studies before exclusion	# of primary studies in final pool	Ratio
[SS1]	408	50	12.25%
[SS2]	111	111	1
[SS3]	157	157	1
[SS4]	1053	238	22.60%
[SS5]	83	74	89.16%
[SS6]	283	71	25.09%
[SS7]	2259	76	3.36%
[SS8]	2338	47	2.01%
[SS9]	1358	58	4.27%
[SS10]	3259	94	2.85%

TABLE A.4: Research questions classification in secondary studies.

Reference	RQs	Type
[SS1]	RQ1. How many papers were published per year?	frequency distribution
	RQ2. What are the most common methods of publication for the papers?	description and classification
	RQ3. Who are the most prolific authors investigating search-based refactoring in software maintenance?	frequency distribution
	RQ4. What types of studies were used in the papers?	description and classification
	RQ5. What refactoring approaches were used in the literature?	description and classification
	RQ6. What search techniques were used in the refactoring studies?	description and classification
	RQ7. What types of programs were used to evaluate the refactoring approaches?	description and classification
[SS4]	RQ1. What is the current status of refactoring with respect to code smells and anti-patterns?	description and classification
	RQ2. What are the different approaches used for the detection of code smells and how the smells are removed using these approaches?	frequency distribution
	RQ3. What are the different tools used by the researchers to identify code smells?	descriptive-process
	RQ4. What are the different datasets used by the authors in order to detect code smells?	causality
	RQ5. What are the different types of code smells spotted in the papers?	description and classification

Continuation of Table A.4

Reference	RQs	Type
[SS5]	RQ1 - What approaches have been proposed on SPL oriented evolution and what is their focus and origin?	description and classification
	RQ1.1 - Which methods or techniques have been investigated and to what extent?	existence
	RQ1.2 - What is the maturity level of the approach? Which tools are already available and which ones are currently used in industry? What types of validation studies are represented and to what extent?	frequency distribution
	RQ2 - Which challenges for SPL oriented evolution have been identified?	existence
[SS6]	RQ1. What type of artifact is refactored, and how is the artifact represented?	description and classification
	RQ2. What are the considered refactorings?	description and classification
	RQ3. What are the methods employed to preserve behavior?	description and classification
	RQ5. What are the most common metrics used to assess the software quality during the search	frequency distribution
	RQ6. Does the approach take into consideration the consistency with other software artifacts?	causality
	RQ7. What are the most common obtained solutions and their representations?	frequency distribution
	RQ8. What are the most common used search-based algorithms?	frequency distribution

Continuation of Table A.4

Reference	RQs	Type
	RQ9. Is any additional information used to guide the optimization process?	existence
	RQ10. What are the used evaluation methods?	description and classification
	RQ11. What are the most common used systems?	frequency distribution
	RQ12. What are the most common used refactoring tools?	frequency distribution
[SS7]	RQ1. What refactoring scenarios were accounted for in the PSs?	description and classification
	RQ2. What quality attributes and measures were considered in the PSs?	description and classification
	RQ3. What approaches and statistical techniques were considered by the PSs to investigate the impact of refactoring on software quality?	description and classification
	RQ4. What datasets were used in the PSs to explore the linkage between refactoring scenarios and software quality?	description and classification
	RQ5. What is the overall impact of refactoring scenarios on software quality across empirical studies?	description and classification
[SS8]	RQ1. What are the refactoring activities considered in the PSs?	description and classification
	RQ2. What are the approaches followed by the PSs to identify the refactoring opportunities?	descriptive-process
	RQ3. What are the approaches followed by the PSs to empirically evaluate the proposed or existing identification techniques for refactoring opportunities?	descriptive-process

Continuation of Table A.4

Reference	RQs	Type
	RQ4. What data sets were used to evaluate the identification techniques proposed in the PSs?	description and classification
[SS9]	RQ. What are the trends, opportunities, challenges and gaps in software refactoring research activities?	description and classification
	SQ. What are the general studies areas (classification) in software refactoring research activities?	description and classification
	SQ. Which part of the research area is exhaustively studied and what are the significant contributions of each study area?	frequency distribution
	SQ. Which part of the research area does not received sufficient attention of study as well as what are the gaps in each study area?	descriptive-comparative
[SS10]	RQ1. Which model specification and transformation languages (also known as Model Transformation System (MTS)) are used to perform model refactoring?	descriptive-process
	RQ2. What model smell detection strategies have been used to identify refactoring opportunities for model refactoring? Do these strategies consider a single or multiple UML views?	description and classification
	RQ3. Which UML model(s) have been used for model refactoring application and what refactoring operations are defined for each of them?	frequency distribution

Continuation of Table A.4

Reference	RQs	Type
	RQ4. How is model behavior defined in each approach and how is behavior preservation verified after model refactoring?	description and classification
	RQ5. What techniques or methods are used to study the effect of model refactoring on model quality?	descriptive-process
	RQ6. Is the refactoring approach integrated seamlessly within existing CASE tools? If integration is not offered, prototype tools in order to facilitate the use of the approach will be considered.	existence

TABLE A.5: Refactoring scenario accounted in secondary studies.

Reference	Study area	Refactoring Scenario
[SS4]	source code	Move method, Move attribute, Extract class and Inline class, Extract Method
[SS5]	SPL	Extract Method/Resource to Aspect, Extract Context, Extract Before/After Block, Move Field to Aspect, Move Import Declaration to Aspect, Move Interface Declaration to Aspect, Move Method to Aspect, Move Extends Declaration to Aspect, Extract Introduction, Extract Advice, Extract Beginning, Extract End, Extract Before/After Call, Addition at the beginning, Addition at the end of the method, Addition anywhere with a hook method, Overwrite method , Move entire method, Move field, Remove field modifiers declarations, Move entire class, Renaming of files and functions, splitting of long files, moving of functions from one module to another, conversion of macros to inline functions, changing of data type,Removal of internal and external code clones, Merging of different implementations and realization using conditional compilation, Reduction of the scale and complexity of functions.

Continuation of Table A.5

Reference	Study area	Refactoring Scenario
[SS6]	source code	Pull up method, Move method, Push down method, Pull up field, Push down field, Extract class, Move field, Inline class, Collapse hierarchy, Extract superclass, Rename method, Add parameter, Extract interface, Encapsulate field, Extract method, Replace delegation with inheritance, Replace inheritance with delegation, Inline method, Remove parameter, Extract subclass, Extract hierarchy, Encapsulate collection, Encapsulate downcast, Hide method, Remove setting method, Self encapsulate field, Form template method, Make class abstract,Make class concrete, Decrease method visibility, Increase method visibility, Decrease field visibility, Increase field visibility, Rename field, Move class, Extract package, Remove method, Rename class, Remove class, Remove interface, Merge packages, Delete generalization, Add relationship, Change superclass down, Change superclass up.
[SS7]	source code	Move method, Extract class, Extract method, Encapsulate field, Pull up method, Extract subclass, Introduce null object, Move class, Move field, Replace method with Method object, Replace data value with object, Replace magic number with symbolic constant, push down method, Replace conditional with polymorphism, Replace type code with state/strategy.

Continuation of Table A.5

Reference	Study area	Refactoring Scenario
[SS8]	source code	Extract Subclass, Move Method, Extract Class, Extract Method, Move Class, Replace Method with Method Object, Replace Data Value with Object, Pull Up Method ,Extract Superclass, Pull Up Method, Form Template Method, Parameterize Method, Pull Up Constructor Form Template Method, Remove Parameter, Eliminate Return Value, Separate Query from Modifier, Encapsulate Downcast, Replace Temp with Query Pull Up Method, Form Template Method, Extract Interface, Remove Parameter, Pull Up Method, extract method Replace Type Code with State/Strategy, Replace Conditional with polymorphism.

A.1 Secondary Studies

[SS1] M. Mohan, D. Greer, A survey of search-based refactoring for software maintenance, Journal of Software Engineering Research and Development 6 (1) (2018) 3

[SS2] T. Mens, T. Tourw'e, A survey of software refactoring, IEEE Transactions on software engineering 30 (2) (2004) 126–139

[SS3] O. Raiha, A survey on search-based software design, Computer Science Review 4 (4) (2010) 203–249

[SS4] S. Singh, S. Kaur, A systematic literature review: Refactoring for disclosing code smells in object oriented software, Ain Shams Engineering Journal

[SS5] M. A. Laguna, Y. Crespo, A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring, Science of Computer Programming 78 (8) (2013) 1010–1034

[SS6] T. Mariani, S. R. Vergilio, A systematic review on search-based refactoring, Information and Software Technology 83 (2017) 14–34

[SS7] J. Al Dallal, A. Abdin, Empirical evaluation of the impact of object-oriented code refactoring on quality attributes: A systematic literature review, IEEE Transactions on Software Engineering 44 (1) (2018) 44–69

[SS8] J. Al Dallal, Identifying refactoring opportunities in object oriented code: A systematic literature review, Information and software Technology 58 (2015) 231–249

[SS9] M. Abebe, C.-J. Yoo, Trends, opportunities and challenges of software refactoring: A systematic literature review, International Journal of Software Engineering & Its Applications 8

[SS10] M. Misbhauddin, M. Alshayeb, Uml model refactoring: a systematic literature review, Empirical Software Engineering 20 (1) (2015) 206–251

References

[1] M. Fowler and K. Beck, *Refactoring: Improving the design of existing code*. Addison-Wesley Professional, 1999.

[2] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering", Keele University and Durham University Joint Report, Tech. Rep. EBSE 2007-001, 2007.

[3] B. Kitchenham, R. Pretorius, D. Budgen, O. Pearl Brereton, M. Turner, M. Niazi, and S. Linkman, "Systematic literature reviews in software engineering - a tertiary study", *Inf. Softw. Technol.*, vol. 52, no. 8, pp. 792–805, Aug. 2010, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2010.03.006. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2010.03.006.

[4] V. Garousi and M. V. Mäntylä, "A systematic literature review of literature reviews in software testing", *Information and Software Technology*, vol. 80, pp. 195 –216, 2016, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2016.09.002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584916301446.

[5] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering – a systematic literature review", *Information and Software Technology*, vol. 51, no. 1, pp. 7 –15, 2009, Special Section - Most Cited Articles in 2002 and Regular Research Papers, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2008.09.009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584908001390.

[6] F. Q. B. da Silva, A. L. M. Santos, S. C. B. Soares, A. C. C. França, and C. V. F. Monteiro, "A critical appraisal of systematic reviews in software engineering from the perspective of the research questions asked in the reviews", in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '10, Bolzano-Bozen, Italy: ACM, 2010, 33:1–33:4, ISBN: 978-1-4503-0039-1. DOI: 10.1145/1852786.1852830. [Online]. Available: http://doi.acm.org/10.1145/1852786.1852830.

[7] D. S. Cruzes and T. Dybå, "Research synthesis in software engineering: A tertiary study", *Information and Software Technology*, vol. 53, no. 5, pp. 440 –455, 2011, Special Section on Best Papers from XP2010, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2011.01.004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095058491100005X.

[8] F. Q. da Silva, A. L. Santos, S. Soares, A. C. C. França, C. V. Monteiro, and F. F. Maciel, "Six years of systematic literature reviews in software engineering: An updated tertiary study", *Information and Software Technology*, vol. 53, no. 9, pp. 899 –913, 2011, Studying work practices in Global Software Engineering, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2011.04.004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584911001017.

[9] G. K. Hanssen, D. mite, and N. B. Moe, "Signs of agile trends in global software engineering research: A tertiary study", in *2011 IEEE Sixth International Conference on Global Software Engineering Workshop*, 2011, pp. 17–23. DOI: 10.1109/ICGSE-W.2011.12.

[10] H. Zhang, M. A. Babar, and P. Tell, "Identifying relevant studies in software engineering", *Information and Software Technology*, vol. 53, no. 6, pp. 625 –637, 2011, Special Section: Best papers from the APSEC, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2010.12.010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584910002260.

[11] A. B. Marques, R. Rodrigues, and T. Conte, "Systematic literature reviews in distributed software development: A tertiary study", in *2012 IEEE Seventh International Conference on Global Software Engineering*, 2012, pp. 134–143. DOI: 10.1109/ICGSE.2012.29.

[12] S. Imtiaz, M. Bano, N. Ikram, and M. Niazi, "A tertiary study: Experiences of conducting systematic literature reviews in software engineering", in *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '13, Porto de Galinhas, Brazil: ACM, 2013, pp. 177–182, ISBN: 978-1-4503-1848-8. DOI: 10.1145/2460999.2461025. [Online]. Available: http://doi.acm.org/10.1145/2460999.2461025.

[13] B. Kitchenham and P. Brereton, "A systematic review of systematic review process research in software engineering", *Information and Software Technology*, vol. 55, no. 12, pp. 2049 –2075, 2013, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2013.07.010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584913001560.

[14] J. Verner, O. Brereton, B. Kitchenham, M. Turner, and M. Niazi, "Risks and risk mitigation in global software development: A tertiary study", *Information and Software Technology*, vol. 56, no. 1, pp. 54 –78, 2014, Special sections on International Conference on Global Software Engineering – August 2011 and Evaluation and Assessment in Software Engineering – April 2012, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2013.06.005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584913001341.

[15] M. Bano, D. Zowghi, and N. Ikram, "Systematic reviews in requirements engineering: A tertiary study", in *2014 IEEE 4th International Workshop on Empirical Requirements Engineering (EmpiRE)*, 2014, pp. 9–16. DOI: 10.1109/EmpiRE.2014.6890110.

[16] R. Hoda, N. Salleh, J. Grundy, and H. M. Tee, "Systematic literature reviews in agile software development: A tertiary study", *Information and Software Technology*, vol. 85, pp. 60 –70, 2017, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2017.01.007.

[Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584917300538.

[17] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update", *Information and Software Technology*, vol. 64, pp. 1–18, 2015.

[18] V. R. Basili, "Software modeling and measurement: The goal/question/metric paradigm", Tech. Rep., 1992.

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

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

[21] 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*, ser. EASE '14, London, England, United Kingdom: ACM, 2014, 38:1–38:10, ISBN: 978-1-4503-2476-2. DOI: 10.1145/2601248.2601268. [Online]. Available: http://doi.acm.org/10.1145/2601248.2601268.

[22] U. Centre for Reviews and Dissemination, "The database of abstracts of reviews of effects (dare)", *Effectiveness Matters*, vol. 6, no. 2, pp. 1–4, Dec. 2002.

[23] B. Kitchenham and P. Brereton, "A systematic review of systematic review process research in software engineering", *Information and software technology*, vol. 55, no. 12, pp. 2049–2075, 2013.

[24] M. Harman and B. F. Jones, "Search-based software engineering", *Information and software Technology*, vol. 43, no. 14, pp. 833–839, 2001.

[25] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications", *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, p. 11, 2012.

[26] W. F. Opdyke, "Refactoring object-oriented frameworks", UMI Order No. GAX93-05645, PhD thesis, Champaign, IL, USA, 1992.

[27] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for software engineering research", in *Guide to advanced empirical software engineering*, Springer, 2008, pp. 285–311.

[28] M. Mohan and D. Greer, "A survey of search-based refactoring for software maintenance", *Journal of Software Engineering Research and Development*, vol. 6, no. 1, p. 3, 2018.

[29] O. Räihä, "A survey on search-based software design", *Computer Science Review*, vol. 4, no. 4, pp. 203–249, 2010.

[30] S. Singh and S. Kaur, "A systematic literature review: Refactoring for disclosing code smells in object oriented software", *Ain Shams Engineering Journal*, 2017.

[31] M. A. Laguna and Y. Crespo, "A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring", *Science of Computer Programming*, vol. 78, no. 8, pp. 1010–1034, 2013.

[32] T. Mariani and S. R. Vergilio, "A systematic review on search-based refactoring", *Information and Software Technology*, vol. 83, pp. 14–34, 2017.

[33] 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, 2018.

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

[35] M. Abebe and C.-J. Yoo, "Trends, opportunities and challenges of software refactoring: A systematic literature review", *International Journal of Software Engineering & Its Applications*, vol. 8, 2014.