International Journal of Software Engineering and Its Applications
Vol. 9, No. 6 (2015), pp. 197-214
http://dx.doi.org/10.14257/ijseia.2015.9.6.19

Non-Source Code Refactoring: A Systematic Literature Review

Siti Rochimah, Siska Arifiani and Vika F. Insanittaqwa

Department of Informatics
Institut Teknologi Sepuluh Nopember
siti@if.its.ac.id, arifiani.tc09@gmail.com, vika.fitra@gmail.com

Abstract

Since software refactoring techniques were introduced, the process is commonly applied to alter the structure of source code. However, there is also increasing popularity in the topic of refactoring in other software artifacts at non-source code level. This paper provides a systematic literature review of existing studies in non-source code refactoring. We use two digital libraries as publication source, IEEExplore and Science Direct, to obtain published articles in non-source code refactoring topic published in between 2002 – 2014 with certain keywords. The 20 selected literatures then processed based on certain criteria, including the refactoring method and refactoring identification source. Then we use this information to provide categorization of non-source code refactoring activity and the advantages and disadvantages of each category. The systematic literature review performed has provided categorization of non-source code refactoring method and has shown that each method has certain advantages and disadvantages. Refactoring detection at non-source code level can be done in the software design model, source code with non-conventional detection, and other software artifacts. The methods used can be categorized as heuristic method, where the refactoring identification is done based on certain rules, and non-heuristic method, where the refactoring identification is done with a certain algoritm that explores every possibility of refactoring opportunities. The advantage of heuristic method is the speed and precision. The disadvantage of this method is the needed effort to produce the rules and the possibility of the non optimal result. The advantage of non-heuristic method is it can yield a generally more optimal result. The disadvantage of non-heuristic method is that the result depends strongly on the robustness of each algorithm.

Keywords: *Software Refactoring, Non-Source Code Refactoring, High Level Refactoring*

1. Introduction

Software refactoring is the process of applying changes to the internal structure of software without changing its observable behavior [**1**]. This process is a part of software evolution process, which aim is to improve software quality after undergoing a lot of code rewriting, modification, feature enhancement, and other changes throughout its lifetime.

Refactoring process is often referred as a reconstruction process of the source code to minimize fault or defect, either as an act of correction or prevention. For example, code clones and other redundancy in a code are considered as threat for future development of the software. These faults in source code are known as “bad smells” in code. Other benefit of refactoring apart from improving the code quality is to improve the structure of the software so that it becomes easier to understand. The understandability of a code reflects the good quality of the software from the perspective of human-based development.

During the early years since software refactoring techniques were introduced, the refactoring process is mainly emphasized on changing the structure of source code. Many

ISSN: 1738-9984 IJSEIA
Copyright ⓒ 2015 SERSC

literatures published on this topic were mainly based on code refactoring catalog by Fowler *et al.,* [**1**], where the diversity of the research lies in the variety of bad smells, refactoring activities, or refactoring scenarios. There were also a lot of research dedicated to create tools to aid the refactoring process, such as Refactoring Browser [**2**], XRefactory [**3**], and Code Imp [**4**]. These tools are applicable in the refactoring of source code in order to get rid of the bad smells.

Amongst the early trend in refactoring, there was a small number of publications addressing the refactoring process in higher software abstraction, such as in high-level model, design pattern, or software architechture [**5**]. The first known publication on this topic was the refactoring of Unified Modelling Language (UML) diagram by Sunyé *et al.,* [**6**], published in 2001. This particular area of research, which we will refer to as "non-source code refactoring", has received an increasing amount of attention and has implemented the refactoring process in other software artifacts.

Publications in non-source code refactoring show a promising research opportunity, which is to implement refactoring activity outside the scope covered in Fowler catalog. In this Systematic Literature Review (SLR), we aim to cover the techniques that had been used in non-source code refactoring activity and to analyze the advantage and disadvantage of each technique. We used two online sources, IEEExplore and Science Direct, to obtain literatures published since 2002 to 2014. After conducting literature selection based on certain criteria, the selected studies then processed and analized to answer the research questions. The result is presented and concluded at the end of this paper.

The rest of the paper is organized as follows: Section 2 presents relevant literatures in the area of software refactoring literature review. Section 3 provides theoritical background about software refactoring and non-source code refactoring. Section 4 describes our method of systematic review. Section 5 reports the results and findings based on the research questions. Finally, the conclusions are presented in Section 6.

2. Related Works

There are few related SLRs regarding software refactoring. Mens dan Tourwé [**7**] performed an extensive survey about software refactoring and compared the refactoring opportunities detection techniques. The downside of this research is that, due to the early publication date, they only consider the publications prior to 2004. The survey also did not follow systematic literature review approach.

Other review conducted by Zhang *et al.,* [**8**] with systematic approach was focused on code bad smells and the refactoring opportunities. Misbhauddin and Alshayeb [**9**] published an SLR about UML model refactoring. It analized 94 literatures about refactoring of UML model and gave an insight of the refactoring impact to model quality. The difference between the study and our SLR is that we also include refactoring process in other models and also in source code which doesn't apply Fowler detection guide of bad smells.

Al Dallal [**10**] published an SLR about refactoring opportunities in object-oriented code which covered refactoring activities in object-oriented perspective only. And recently, Abebe and Yoo [**11**] published an SLR about trends, opportunities, and challenges in software refactoring in wider scope.

It is evident from the list of existing SLRs performed in software refactoring that there hasn't been any attempt to systematically review the refactoring process in non-source code level. This is the distinctive feature and contribution of this paper in a hope to provide a systematic review about the subject.

 Copyright ⓒ 2015 SERSC

3. Background

3.1. Software Refactoring

The term software refactoring was used for the first time by William F. Opdyke in his PhD dissertation [**12**] and became more commonly used after the publication of the book "Refactoring: Improve the Design of Existing Code" by Fowler *et al.,* in 1999 [**1**]. Since then, software refactoring is known as a common practice to improve and enhance software quality.

Before the restructuring process is applied, there is an identification process to pinpoint where the refactoring should be done. The most common sign of refactoring opportunity is the presence of bad smells in source code. Fowler *et al.,* [**1**] provide a list of bad smells with their corresponding refactoring techniques which makes the refactoring decision easier. Generally, there are about 25 bad smells and more than 75 software refactoring techniques for use.

Abebe and Yoo [**11**] provides a flowchart about common software refactoring process, which is shown in Figure 1. The process includes the following activities.

a. Apply unit test to the program.
b. Identify which part of the code needs the refactoring using code smells.
c. Select a refactoring technique to remove the identified code smell.
d. Apply the selected refactoring technique.
e. Apply regression testing to the refactored code.
f. Assess the effect of the refactoring using software quality characters or the process.
g. Maintain consistency between the program and the other artifacts.

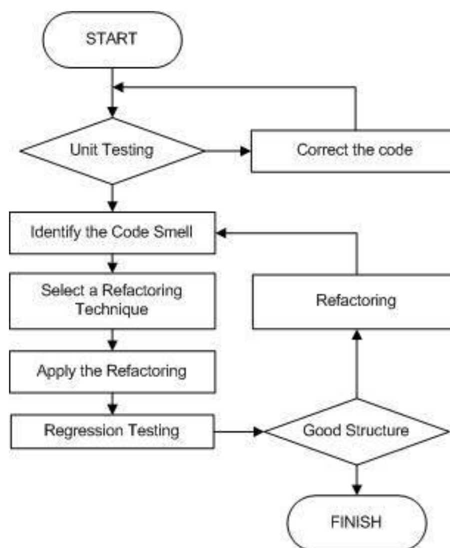

Figure 1. The Common Steps in Software Refactoring Process

There are few benefits from software refactoring. The benefits are as follows.

a. To remove duplicated code and other bad smells.
b. To improve software design quality.
c. To increase understandability of the code.
d. To reduce project evolution time, especially in source code management activities.

Copyright ⓒ 2015 SERSC

Fowler *et al.,* [**1**] defines 7 groups of software refactoring activities that can be implemented to source code. The 7 groups are as follows.

a. Composing Methods
b. Moving features
c. Organizing data
d. Simplifying conditional expression
e. Making methods call simple
f. Dealing with generalization.
g. Big refactoring

There are other tools to aid software refactoring detection and implementation process, such as Refactoring Browser [**2**], XRefactory [**3**], Code Imp [**4**], and many others. These tools are able to detect some of refactoring opportunities described in Fowler catalog. There is also a tendency to integrate automated refactoring module to Integrated Development Environments (IDEs), such as Smalltalk VisualWorks [**13**], Eclipse [**14**], and IntelliJ IDEA [**15**].

In this paper, the detection of refactoring opportunities by the code bad smells will be reffered to as "conventional detection". There are also few ways to detect refactoring opportunities in source code other than by bad smells. We will refer to them as "non-conventional detection", which will be reviewed in this paper.

8.1. Non-source Code Refactoring

Non-source code refactoring refers to two kinds of software refactoring: 1) refactoring activities in other software artifact besides source code, and 2) refactoring activities in source code with non-conventional detection techniques. There are already few implementations in this type of refactoring, such as in UML models, requirement specifications, software architechture, and source code refactoring to a particular design pattern.

This type of refactoring is applicable to higher abstraction other than source code, such as high-level model, refactoring to design patterns, and software architechture [**5**]. One of the first known publications about non-source code refactoring is "Refactoring UML Models" written by Sunyé *et al.,* in 2001 [**6**].

4. Research Method

The research method in this paper is based on the guide by Kitchenham and Charters [**16**]. The main objective of this paper is to present an overview about the techniques in non-source code refactoring and to analize the advantages and disadvantages of each technique.

The first stage of this research is the planning stage, which is to define the scope, research questions, and steps needed to answer the research questions. The next stage is conducting the review itself by searching for papers in two publication sources, filtering process based on inclusion and exclusion criteria, quality assessment, data extraction, and analysis process. The last stage of the research is the documentation stage, which includes the writing of the paper and validation process. Figure 2 shows the research method used in this SLR.

 Copyright ⓒ 2015 SERSC

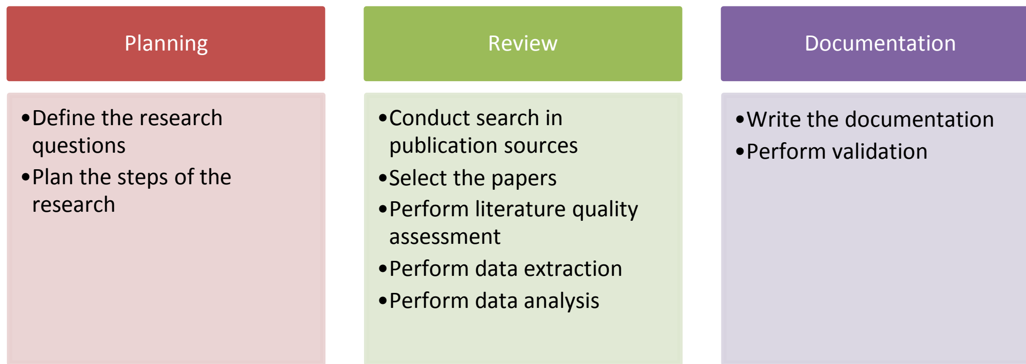

Figure 2. Research Method

The planning process consists of the following activities.

- *Define the research questions*
  - Determine the scope of non-source code refactoring
    This activity aims to define the kinds of software refactoring that can be classified as non-source code refactoring.
  - Determine the questions that need to be addressed throughout the research
    This activity aims to make clear direction of the research to make it more focused.
- *Plan the steps of the research*
  - Make written documentation about the steps needed in the research, including the inputs (such as search keywords, inclusion and exclusion criteria, and others) and the desired outcome of each steps (documentation of search query result, and so on).

After planning, we conduct main review process which consists of the following activities.

- *Conduct search in publication sources*
  - Conduct the search with previously defined keywords.
  - Conduct the search with publication type and year filter.
  - Write down the result of the process
- *Select the papers*
  Select or filter the search results based on inclusion and exclusion criteria which are defined in planning activities. The desired outcome of this process is the list of main studies that will be used in analysis process.
- *Perform literature quality assessment*
  This process aims to assess the quality of selected literature based on completeness of writing, publication type, publication year, and number of citing articles.
- *Perform data extraction*
  This process aims to provide data needed from literatures, such as refactoring detection source, refactoring techniques, and other information for analysis process.
- *Perform data analysis*
  This process will use information from previous step and digest it to answer research questions and to find the conclusion of the research.

The last stage of the research is the documentation stage which consists of the following activities.

- *Write the documentation*
  This process will generate a systematically written report about the SLR, including the findings of the SLR and conclusions.
- *Perform paper validation*

Copyright ⓒ 2015 SERSC

This process aims at assessing the quality of the report. Paper revisions and improvisations will be done in this process until the final report is produced.

4.1. Research Questions

There are three research questions (RQ) which will be explored, analyzed and concluded about non-source code refactoring. The research questions are as follows.

- RQ1: How many kinds of refactoring techniques applied in non-source code refactoring?
- RQ2: What are the approaches or methods used in non-source code refactoring?
- RQ3: What are the advantages and disadvantages of each kind of methods?

4.2. Search Strategy

There are two digital libraries used in the search process which can be accessed online: IEEExplore [**17**] and Science Direct [**18**]. They were chosen due to limitation of freely available digital library access in writer's institution. Other digital libraries aren't available for free article download and thus excluded as main source.

The keywords used in literature search are as follows.

- Software Refactoring
- Design Refactoring
- Model Level Refactoring
- Non Source-code Refactoring
- "Design pattern" AND Refactoring
- Architectural Refactoring

All of the keywords are used as search query in two digital libraries to search any journal article or conference paper published from 2002 to 2014. The search results then will be selected in the next process.

4.3. Data Selection

There are few criteria to select the main literatures used in analysis process. The inclusion criteria determine which literature will be included in the data selection stage. The inclusion criteria are as follows.

- Studies about a non-source code refactoring research, where the refactoring method or approach used is applied at higher abstraction level (such as UML models) or is applied at source code with non-conventional detection of refactoring opportunities.
- Studies published in journal or conference.
- Studies published since 2002 to 2014.
- Studies written in English.

The data selection process about which refactoring method is included is illustrated by Figure 3. Only the refactoring applied at higher level than source code and the refactoring in source code with non-conventional detection will be selected. In Figure 3, the selected study is depicted by boxes in solid outline.

Copyright ⓒ 2015 SERSC

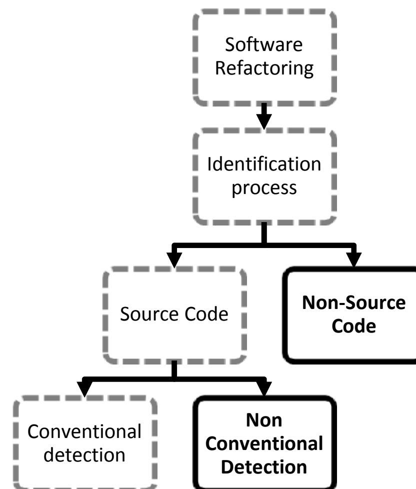

Figure 3. Inclusion Criteria about Refactoring Method in Data Selection

There are also exclusion criteria on data selection. The exclusion criteria are as follows.

- Studies about software refactoring at source code level which uses conventional detection approach
- Studies which are not about software refactoring, such as hardware model refactoring.
- Studies which are not published in a journal or conference, such as theses or dissertation.
- Studies published prior to 2002.
- Studies which are not written in English.
- Studies with duplicated title.

Table 1 shows the search result obtained from the search queries. There are a total of 7669 literatures in total, which hasn't been further selected. The selection process is then conducted in four stages. Figure 4 shows the selection process and the number of literatures selected in each stage.

Table 1. Search Result in Online Libraries

Keywords	IEEExplore			Science Direct		Sub-Total
	Confe-rence	Journals & Magazines	Books & Ebooks	Journals	Books	
Software Refactoring	701	85	1	1081	286	2154
Design Refactoring	287	37	2	1093	293	1712
Model Level Refactoring	48	2	0	1022	242	1314
Non Source-code Refactoring	4	1	0	701	112	818
"Design pattern" AND Refactoring	73	6	0	829	224	1132
Architectural Refactoring	37	6	0	363	133	539
TOTAL						7669

Copyright ⓒ 2015 SERSC

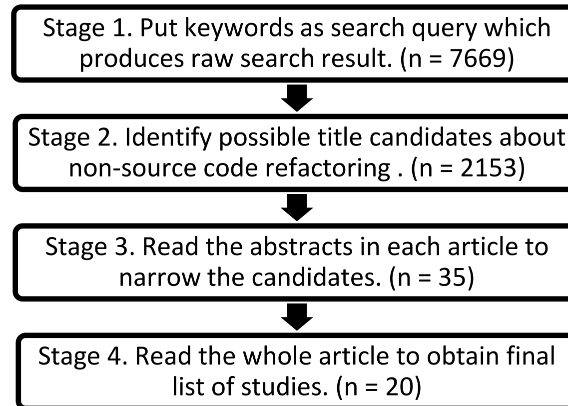

Figure 4. Four Selection Stages (Where *n* is the Number of Selected Literatures in Each Stage)

The first step of selection process is to put the keywords as search queries in two digital libraries, which yields a total of 7669 studies. The titles then selected to find candidates about non-source code refactoring. If the title clearly indicates that it's not about software refactoring, then the title will be eliminated immediately. The titles which imply that the refactoring process is implemented in source code with bad smell detection will also be eliminated. This selection stage yields 2153 study candidates.

To narrow down the candidates, the result then processed in the next stage. After reading the abstracts, the candidates are down to 35 titles only. This wide gap from the previous stage is mainly because of the oftenly misleading title about refactoring in general. The most common case is there is no clarity in the title regarding where the refactoring detection is applied. For example, the title "Cloning and Expanding Graph Transformation Rules for Refactoring" may imply that the refactoring is done at higher abstraction. However, the abstract states otherwise, which reveals that the study actually presents a refactoring technique at source code level with conventional detection. This misleading title also occurs in many other titles, which is why the resulting selection from third stage only yields significantly smaller number.

The next stage is to read the whole study to determine the final selected titles. There are 20 main articles which are yielded from this stage that will be further processed in quality assessment before can be used in data extraction process. The 20 titles are as follows.

- Identifying Refactoring Opportunities In Process Model Repositories [**19**]
- Evaluation Of Model Transformation Approaches For Model Refactoring [**20**]
- On The Refactoring Of Activity Labels In Business Process Models [**21**]
- Considerations On The Development Of A Refactoring-Based Navigation Model For On-Line Transaction Systems [**22**]
- Refactoring large process model repositories [**23**]
- Feature Unweaving: Refactoring Software Requirements SpecificationsInto Software Product Lines [**24**]
- From Requirements to Features: An Exploratory Study of Feature-Oriented Refactoring [**25**]
- Refactor Software Architecture Using Graph Transformation Approach [**26**]
- Graph Modelling Of A Refactoring Process For Product Line Architecture Design [**27**]
- Automated Refactoring Using Design Differencing [**28**]
- Software Refactoring At The Package Level Using Clustering Techniques [**29**]
- Toward automated refactoring of crosscutting concerns into aspects [**30**]

 Copyright ⓒ 2015 SERSC

- Supporting Requirements Traceability through Refactoring [**31**]
- Traceability-Enabled Refactoring for Managing Just-In-Time Requirements [**32**]
- Search-based refactoring for software maintenance [**33**]
- Automated refactoring to the Strategy design pattern [**34**]
- Constructing models for predicting extract subclass refactoring opportunities using object-oriented quality metrics [**35**]
- A technique for automatic component extraction from object-oriented programs by refactoring [**36**]
- Identifying Extract Class refactoring opportunities using structural and semantic cohesion measures [**37**]
- Refactoring of Crosscutting Concerns with Metaphor-Based Heuristics [**38**]

4.4. Study Quality Assessment

The quality of this SLR has direct corelation to the data quality. The quality of the articles is reflected by the relevance and credibility of the study. There are four quality assessment questions to determine the quality of the data.

Q1. Is the method of refactoring process described clearly and thoroughly?

This question aims to assess the clarity aspect of the literature. The description of the method should be given clearly in the article, whether it's in one particular section or more. The lack of documentation, such as evaluation or implementation process, indicates bad literature quality.

Q2. What is the type of publication of the literature?

This question aims to inform whether the article is published in journal or conference.

Q3. When was the literature published?

This question aims to inform whether the article can be considered as outdated. The more recent articles are considered better than, for example, the article that was published a decade ago.

Q4. How many other studies refer to the literature?

This question aims to assess the quality of contribution and relevance of the article, where good study is reflected in how useful it is to future research.

The final selected study are then analysed with quality assessment questions. The articles which don't provide ample descriptions of the study are marked as "No" in Q1. There are two articles which have this condition. The result of this assessment is presented in Table 2.

Table 2. Literature Quality Assessment

No.	No Ref.	Q1	Q2	Q3	Q4	No.	No Ref.	Q1	Q2	Q3	Q4
1	[**19**]	Yes	Jurnal	2011	12	11	[**29**]	Yes	Jurnal	2011	2
2	[**20**]	Yes	Jurnal	2014	1	12	[**30**]	Yes	Jurnal	2013	0
3	[**21**]	Yes	Jurnal	2011	24	13	[**31**]	Yes	Conference	2013	1
4	[**22**]	No	Conference	2012	0	14	[**32**]	Yes	Conference	2014	0
5	[**23**]	Yes	Jurnal	2011	57	15	[**33**]	Yes	Conference	2006	14
6	[**24**]	No	Conference	2010	2	16	[**34**]	Yes	Jurnal	2012	2
7	[**25**]	Yes	Conference	2011	3	17	[**35**]	Yes	Jurnal	2012	5
8	[**26**]	Yes	Conference	2012	0	18	[**36**]	Yes	Jurnal	2005	16
9	[**27**]	Yes	Conference	2013	0	19	[**37**]	Yes	Jurnal	2011	26
10	[**28**]	Yes	Conference	2012	1	20	[**38**]	Yes	Jurnal	2009	6

4.5. Data Extraction and Analysis

The data in the selected articles is then extracted to answer the research questions. The data extracted from the articles are refactoring opportunity detection source, refactoring

Copyright ⓒ 2015 SERSC

methods or approach, implementation or case study, tools, and advantages and disadvantages of the method.

These data is procured by reading the 20 articles. After all the data is gathered, it will be analysed based on the research questions. The detail of the data extraction is provided in the Appendix with the exception of the advantages and disadvantages which are presented in Section 5.

5. Result

5.1. RQ1: Non-source Code Refactoring Categorization

There are two main classifications about the 20 articles found in data selection. The first classification indicates which software artifact is used as a source in refactoring opportunity detection. The second classification indicates the kinds of methods used in performing the refactoring activity.

5.1.1. Categorization based on Refactoring Opportunity Detection Sources: There are three kinds of sources that can be used as refactoring opportunity detection in non-source code refactoring. They are software design diagrams, source code, and other software documentations. The articles are mapped to the three groups, as shown in Table 3.

Table 3. Categorization based on the Refactoring Idenfication Source

Source Category	Description	Cited Ref.
Software design diagrams	Refactoring opportunity is identified through software design diagrams, such as UML diagrams, process models, and other diagrams.	[**19**], [**20**], [**21**], [**22**], [**23**], [**26**], [**27**], [**28**], [**37**], [**38**]
Source Code	Refactoring opportunity is identified through source code without using conventional detection method.	[**29**], [**30**], [**32**], [**34**], [**36**]
Other Software Documentations	Refactoring opportunity is identified through other software artifact other than diagrams and source code.	[**22**], [**24**], [**25**], [**31**], [**33**], [**35**]

There are ten studies which used software design diagram as refactoring opportunity detection source. Four studies used UML class diagram as the identification source [**20**] [**22**] [**28**] [**37**] [**38**], three studies used bussiness proses model [**19**] [**21**] [**23**], and two studies used software architechtural diagram [**26**] [**27**].

There are also studies which detect the refactoring opportunity from source code with non-conventional detection method. The refactoring activity is generally applied not to remove faults in the code, but to shift to a certain programming paradigm or to improve the code based on certain quality metrics. The five studies found have different aims. Source code can be refactored to comply the structure of design pattern, such as Strategy design patern [**34**], or to modify the structure of the code based on Aspect Oriented Programming (AOP) [**30**] or component-based programming with class relation graph as code model [**36**]. Cohesion and coupling quality can be used as a motivation for source code refactoring [**29**]. Lastly, software bug or error report can be used to locate refactoring opportunity in source code [**32**].

Other software artifacts can also be potentially used to detect refactoring activity. Five studies have shown that refactoring opportunity can be obtained from software requirement model [**24**], software features [**25**], software documents or written artifacts [**31**], and design quality model [**33**] [**35**].

 Copyright ⓒ 2015 SERSC

5.1.2. Categorization based on Refactoring Methods: Apart from the refactoring identification source, non-source code refactoring studies also used various methods in the refactoring activity. There are three categories of the refactoring methods or approaches used in the studies: "heuristic method", "non-heuristic method", and "others".

A refactoring method will be included in heuristic category if the refactoring process involves certain rules or previously defined queries as the main model for refactoring decision. The rules or queries are generally derived from the findings in previous studies. On the other hand, a refactoring method will be categorized as non-heuristic method if the refactoring identification is done thoroughly without model reference, so that every possible candidates of refactoring is considered through exhaustive search or other approaches. Studies which don't use either kind of method will be categorized as "others". The analysis process yields categorization result in Table 4.

Table 4. Categorization based on the Refactoring Methods

Method Category	Description	Cited Ref.
Heuristic method	Refactoring activity is done according to previously defined rules or queries	[**26**], [**27**], [**28**], [**34**], [**38**]
Non-heuristic method	Refactoring activity is done with certain algorithm or approach to explore every possible candidates for refactoring without model reference	[**19**], [**21**], [**29**], [**30**], [**31**], [**32**], [**33**], [**35**], [**36**], [**37**]
Others	The study doesn't use heuristic or non-heuristic approach	[**20**], [**22**], [**23**], [**24**], [**25**]

From the results shown in Table 4, there are 5 studies which used heuristic method in refactoring activity at non-source code level. Two studies conducted experiments which involve graph to model the software architechture and detect refactoring opportunity with graph refactoring rule implemented with Attribute Graph Grammar to remove architechture bad smells [**26**] and with Product Line Architecture (PLA) feature paths derived from desired model [**27**]. There are also rules derived from UML class diagram by forming queries based on design differencing [**28**] and by modeling the code to certain class structure called "octopus" or "black sheep" [**38**]. The modeling of "octopus" and "black sheep" refers to the number of attributes and methods in each class. The design pattern rule is used to model the code to Strategy design pattern with polymorphism algorithm [**34**].

The non-heuristic method is used in half of the main studies with various algoritm and approaches. Amongst 10 studies, three of them use clustering techniques as refactoring decision in package level of source code based on cohesion and coupling metrics with Adaptive K-Nearest Neighbor (A-KNN) algorithm [**29**], in eXtensible Markup Language (XML) model of software requirement [**32**], and in component architecture model to find similar components for refactoring [**36**]. Two studies use non-heuristic search method to enable requirement traceability in a software [**31**] and to produce new software design model based on design quality model [**33**]. Besides clustering and searching algortithm, there are also few other studies using different approaches, such as fragment similarity in Refined Process Structure Tree [**19**], Natural Language Processing (NLP) for refactoring of activity label on business process models [**21**], Markov model for refactoring based on AOP [**30**], univariate logistic regression analysis algorithm applied in software quality metrics for refactoring opportunity [**35**], and Max Flow Min Cut algorithm to detect class bad smell in graph representation of the class [**37**].

The last 5 studies don't introduce a method or approach in refactoring activity, thus the studies don't belong to either heuristic or non-heuristic method. Two studies are about introducing refactoring opportunity catalog in business process model [**23**] and

Copyright ⓒ 2015 SERSC

refactoring in Software Product Line (SPL) [**25**]. The first catalog consists of 8 process smells and 11 refactoring activity in process model. The second catalog introduces 8 refactoring opportunity in SPL. Besides catalog, there is one study comparing five model transformation approaches (QVT-E, ATL, Kermeta, UML-RSDS, and GrGen.Net) in refactoring implementation [**20**]. Two other literatures [**22**] [**24**] didn't include full explanation of the approach so that it was impossible to conclude about the refactoring method used in both studies.

5.2. RQ2: Approaches or Methods used in Non-source Code Refactoring

Previous analysis about categorization of the methods shows that there are many different approaches in refactoring activity in non-source code level. Many refactoring detections are done with the aid of established refactoring rules [**26**] [**27**] [**28**] [**38**] [**34**], which enables fast and easy detection of refactoring opportunity. Clustering [**29**] [**32**] [**36**] and searching methods [**31**] [**33**] are also used in few studies to detect refactoring at non-source code level, in which the detection process is done exhaustively from every possibility.

Other methods used in refactoring identification are similarity matching between fragments [**19**], NLP [**21**], Markov model [**30**], univariate logistic regression analysis [**35**], and Max Flow Min Cut [**37**] in graph representation of class.

5.3. RQ3: Advantages and Disadvantages of Each Kind of Methods in Non-source Code Refactoring

We also gather the writer's evaluation of their methods and use the data to draw an insight about the advantages and the disadvantages of each method in refactoring, especially in heuristic and non-heuristic method as categorization of refactoring methods in non-source code refactoring.

One advantage of heuristic method is the short time needed to find refactoring opportunities based on certain rules, as shown in few studies [**26**] [**27**] [**28**] [**34**] [**38**]. This is due to the fact that the refactoring identification process doesn't consider every possibility in refactoring activity and relies heavily on the defined rules. The other advantage of this method is the precision of the refactoring detection, because the rules are generally derived from findings in previous studies.

The disadvantage of the heuristic method is the possibility that the rules can not yield optimal results. One example is refactoring detection with few architechture bad smells rule with AGG [**26**] that only yielded three refactoring activity from 8 possible bad smells. The rule itself is another disadvantage of this method, because precise detection is strongly determined by the rule definition beforehand. To derive "correct" rules, another study or effort is needed to make sure the correctness of refactoring detection. The reviewed studies show that the rules also needs to be implemented or defined before the detection process, as seen in architechtural bad smells rules in AGG tools [**26**] and rules from class diagram design differencing [**27**]. Other two studies implement the rules based on a particular design pattern [**34**] and previously defined rules through analysis process [**38**].

The non-heuristic method differs mainly on the scope of detection process, where the non-heuristic is generally done thoroughly and exhaustively. The process itself is done with different approaches, such as clustering [**29**] [**32**] [**36**], searching methods [**31**] [**33**], and others [**19**] [**21**] [**30**] [**35**] [**37**]. This process tends to yield more optimal result than heuristic method.

However, even though one algorithm can perform exhaustive search on refactoring opportunities, the scope of the detection result mainly depends on the robustness of algorithm itself. That being said, there is no guarantee that a non-heuristic algorithm can detect all refactoring catalog in non-source code refactoring. For example, fragment

 Copyright ⓒ 2015 SERSC

similarity detection in Refined Process Structure Tree can only detect 4 from 7 possible refactoring activities in business process model [**19**]. Another example, NLP methods implemented in refactoring process in activity label in process model can only detect labels in English and still has few difficulties regarding compound sentence, irregular verbs, and other linguistic limitations [**21**].

Thus, it can be concluded that each method has its own advantages and disadvantages. Heuristic method is fast and precise, but the rule definition needs extra effort and there's a possibility that the result isn't optimal. On the other hand, non-heuristic method can obtain exhaustive and optimal result, but the performance depends strongly on the robustness of each algorithm.

6. Limitations of the Study

This research has few limitations. From the research method defined in Section 4, it has been stated that there are only two digital libraries used in this SLR. This number may look inferior compared with eight digital libraries recommended by Brereton *et al.,* [**39**] in conduction literature review. This limitation is justifiable because of access limitations in other digital libraries in writer's institution.

Other limitation is that this SLR only use the studies filtered by certain keywords defined in Section 4. There is always possibility that there are other titles that don't show up with the queries. This limitation can be also caused by the limitation in the search engine and the fact that we only use two digital libraries as main sources.

Lastly, there is no definitive evaluation metrics in this research which makes us incapable of performing a full and systematic quality assessment. The agreement between writers itself is done informally with no clear documentation whatsoever. However, the research is done under close supervision of one credible senior researcher with relevant expertise in software evolution field. Therefore the result of this SLR can be regarded as a sufficient attempt in providing systematic literature review to the research questions posed about the refactoring activity at non-source code level.

6. Conclusions

This research covers many findings from the data analysis, which can be concluded as follows.

1) Refactoring detection at non-source code level can be done in software design model, source code with non-conventional detection, and other software artifacts. The methods used can be categorized as heuristic method, where the refactoring identification is done based on certain rules, and non-heuristic method, where the refactoring identification is done with certain algoritm that explores every possibility of refactoring opportunities. The studies which propose neither method are generally survey papers about refactoring catalog or comparative review.
2) Refactoring activity with heuristic method is done with certain pre-defined rules, which the refactoring identification is based on. Refactoring activity with non-heuristic method is done with certain algorithm that can explore every possibility of refactoring opportunities, such as clustering techniques, searching techniques, and others.
3) The advantage of heuristic method is the speed and precision. The disadvantage of this method is the needed effort to produce the rules and the possibility of the non optimal result. The advantage of non-heuristic method is the generally more optimal result. The disadvantage of non-heuristic method is the result depends strongly on the robustness of each algorithm.

Copyright ⓒ 2015 SERSC

APPENDIX

Data extraction result.

No ref.	Refactoring source	Method type	Method used	Data set	Tools
[19]	Process Model	Non-heuristic	Detect few refactoring opportunities with finding similar process parts in Refined Process Structure Tree	IBM Insurance Application Architechture model (IAA) and SAP Reference Model (SAP-RM)	-
[20]	Class diagram	Others (Survey paper)	The paper provides systematic evaluation for comparing model transformation approaches	-	-
[21]	Process Model	Non-heuristic	Refactoring operation which implements rename activity label refactoring with Natural Language Processing (NLP)	SAP Reference Model, TelCo, Signavio collection	Stanford Parser and WordNet
[22]	Class diagram	Others (unclear)	Applying refactoring technique to improve navigation in online transaction system by modeling navigation flow with class diagram which produces “navigation structure diagram”	-	-
[23]	Process Model	Others (survey paper)	The paper proposes a catalogue of 8 process model "smells" for identifying refactoring opportunities and introduces 11 techniques for refactoring large process repositories.	-	Eclipse
[24]	Requirement Model	Others (unclear)	The paper propose a technique called “feature unweaving” to identify and extract features from graphical software requirement model for refactoring process	-	-
[25]	Feature	Others (Survey paper)	The paper identified 8 refactoring patterns that describe how to extract the elements of features in Software Product Line (SPL)	VODPlayer and Gantt Project	-
[26]	Software architechture	Heuristic	Detect few refactoring opportunities in software architechture model to remove architechtural bad smells using a set of rules in AGG	A case study of a client-server configuration (CS-config) model	(Attribute Graph Grammar (AGG)
[27]	Software architechture	Heuristic	Refactoring of software architechture in Software Product Line (SPL) to produce similar product architechture of a domain with Bosch heuristic to obtain optimal candidate for Product Line Architechture (PLA)	A case study of PLA in Robotic Industry	-
[28]	Class diagram	Heuristic	Refactoring detection using class diagram based on 8 implemented queries related to 14 source-level refactoring to modify current model into desired model	Six open source java projects (JHotDraw, JGraphX, JTar, HtmlUnit, GanttProject, XOM)	JDEvAn, UMLDiff, CodeImp
[29]	Source code	Non-heuristic	Identifying ill-structured packages reflected in the intra-package cohesion and inter-package coupling with clustering technique.	Trama	-
[30]	Source code	Non-heuristic	Refactor an object-oriented-system into an aspect-oriented using association rules and Markov model	J2EE system (Java Pet Store)	-
[31]	Software artifact	Non-heuristic	Modern traceability tools employing information retrieval (IR) methods to generate candidate traceability links.	iTrust, eTour, WDS	-
[32]	Source code	Non-heuristic	Detecting refactoring based on bug-	Apache, Mozilla	-

Copyright ⓒ 2015 SERSC

			information or error, to get a new software requirement		
[**33**]	Quality Model	Non-heuristic	Refactoring detection using software quality metrics using search algorithm (probability distribution) to find an optimal probability	**-**	**-**
[**34**]	Source code	Heuristic	Refactoring source code into "Strategy" design pattern	JDeodorant Eclipse plug-in	**-**
[**35**]	Quality Model	Non-heuristic	Source code is replaced by new source code structure which is fulfill with 'State' or 'Strategy' criteria	6 java open source programs	**-**
[**36**]	Source code	Non-heuristic	Detecting extract class refactoring opportunity from the information of class' relation modeled with graph with clustering	**-**	**-**
[**37**]	Class diagram	Non-heuristic	Refactoring from graph which represent relation between classes. The relation is represented by cohesion and coupling with Max Flow Min Cut algorithm	GanttProject System	**-**
[**38**]	Class Diagram	Heuristic	Refactoring is done with diagram heuristics: 'Octopus' and 'Black Sheep' which show class relation with each other based on the attributes.	Java Code	**-**

References

[1] M. Fowler, K. Beck, J. Brant, W. Opdyke and D. Roberts, Refactoring: Improving the Design of Existing Code.: Addison-Wesley Professional, **(1999)**.
[2] "Refactoring Browser", [Online]. http://c2.com/cgi/wiki?RefactoringBrowser.
[3] "Xrefactory", [Online]. http://www.xref.sk/xrefactory-java/main.html.
[4] "Code Imp", [Online]. http://ulir.ul.ie/handle/10344/2220.
[5] B. D. Bois, *et al.,* "A Discussion of Refactoring in Research and Practice", IEEE Transactions on Software Engineering, vol. 27, no. 6, **(2002)**, pp. 512-530.
[6] G. Sunyé, D. Pollet, L. T. Yvez and J.-M. Jézéquel, "Refactoring UML Models", in Proceeding of The Unified Modeling Language: Modeling Languages, Concepts, and Tools, **(2001)**, pp. 134-148.
[7] T. Mens and T. Tourwé, "A Survey of Software Refactoring", IEEE Transactions on Software Engineering, vol. 30, no. 2, **(2004)** February, pp. 126-139.
[8] M. Zhang, T. Hall and N. Baddoo, "Code Bad Smells: a review of current knowledge", Journal of Software Maintenance and Evolution: Research and Practice, vol. 23, no. 3, **(2011)** April, pp. 179-202.
[9] M. Misbhauddin and M. Alshayeb, "UML model refactoring: a systematic literature review", Empirical Software Engineering, **(2013)**, pp. 1-46.
[10] J. A. Dallal, "Identifying refactoring opportunities in object-oriented code: A systematic literature review", Information and Software Technology, vol. 58, **(2014)** February, pp. 231–249.
[11] M. Abebe and C.-J. Yoo, "Trends, Opportunities and Challenges of Software Refactoring: A Systematic Literature Review", Journal of Software Engineering and Its Applications, vol. 8, no. 6, **(2014)**, pp. 299-318.
[12] W. F. Opdyke, "Refactoring Object-Oriented Frameworks", Department of Computer Science, University of Illinois at Urbana-Champaign, PhD Thesis, **(1992)**.
[13] "Cincom Systems Inc. Cincom Smalltalk", [Online], http://www.cincomsmalltalk.com/main/products/visualworks/.
[14] "The Eclipse Foundation", Eclipse. [Online]. https://eclipse.org/.
[15] "JetBrains", IntelliJ IDEA, [Online]. https://www.jetbrains.com/idea/.
[16] B. Kitchenham and S. Charters, "Guidelines for Performing Systematic Literature Reviews in Software Engineering," Keele University, UK, Technical Report EBSE, **(2007)**.
[17] "IEEExplore Digital Library", [Online]. http://ieeexplore.ieee.org/Xplore/home.jsp.
[18] "Elsevier Science Direct", [Online]. http://www.sciencedirect.com/.
[19] R. Dijkman, B. Gfellar, J. Kuster and H. Volzer, "Identifying refactoring opportunities in process model repositories," Information and Software Technology, vol. 53, **(2011)**, pp. 937-948.
[20] S. Kolahdouz-Rahimi, K. Lano, S. Pillay, J. Troya and P. VanGorp, "Evaluation of model transformation approaches for model refactoring," Science of Computer Programming, vol. 85, **(2014)**, pp. 5-40.
[21] H. Leopold, S. Smirnov and J. Mendling, "On the refactoring of activity labels in business process models," Informations Systems, vol. 37, **(2012)**, pp. 443-459.

Copyright ⓒ 2015 SERSC

[22] M. Avorniculuia, G. Kovacsa and V. P. Bresfeleana, "Considerations on the Development of a Refactoring-Based Navigation Systems", in Procidia Economics and Finance, vol. 3, **(2012)**, pp. 591-596.
[23] B. Weber, M. Reichert, J. Mendling and H. A. Reijers, "Refactoring large process models repository," Computers in Industry, vol. 62, **(2011)**, pp. 467-486.
[24] R. Stoiber, S. Fricker, M. Jehle and M. Glinz, "Feature Unweaving: Refactoring Software Requirements Specifications into Software Product Lines", in 18th IEEE International Requirements Engineering Conference, **(2010)**, pp. 403-404.
[25] R. E. Lopez-Herrejon, L. Montalvillo-Mendizabal and A. Egyed, "From Requirements to Features: An Exploratory Study of Feature-Oriented Refactoring," in 15th International Software Product Line Conference, **(2011)**, pp. 181-190.
[26] A. Amirat, A. Bouchouk, M. O. Yeslem and N. Gasmallah, "Refactor Software Architechture Using Graph Transformation Approach," in 2nd International Conference on Innovative Computing Technology (INTECH), **(2012)**, pp. 117-122.
[27] F. Losavio, O. Ordaz, N. Levy and A. Baïotto, "Graph Modelling of a Refactoring Process for Product Line Architechture Design," in XXXIX Latin American Computing Conference (CLEI), **(2013)**, pp. 1-12.
[28] I. H. Moghadam and M. O. Cinnéide, "Automated Refactoring using Design Differencing," in 16th European Conference on Software Maintenance and Reengineering, **(2012)**, pp. 43-52.
A. Alkhalid, M. Alshayeb and S. A. Mahmoud, "Software refactoring at the package level using clustering techniques," IET Software, vol. 5, no. 3, **(2011)**, pp. 274-286.
[29] S. A. Vidala and C. A. Marcosa, "Toward automated refactoring of crosscutting concerns into aspects," The Journal of Systems and Software, vol. 86, **(2013)**, pp. 1482-1497.
[30] A. Mahmoud and N. Niu, "Supporting Requirements Traceability through Refactoring," in 21st IEEE International Conference of Requirements Engineering (RE), **(2013)**, pp. 32-41.
[31] N. Niu, T. Bhowmik, H. Liu and Z. Niu, "Traceability-Enabled Refactoring for Managing Just-In-Time Requirements," in 22nd IEEE International Conference on Requirements Engineering (RE), **(2014)**, pp. 133-142.
[32] M. O'Keeffe and M. O. Cinnéide, "Search-based Refactoring for software maintenance," The Journal of Systems and Software, vol. 81, **(2008)**, pp. 502-516.
[33] A. Christopoulou, E. A. Giakoumakis, V. E. Zafeiris and V. Soukara, "Automated refactoring to the Strategy design pattern," Information and Software Technology, vol. 54, **(2012)**, pp. 1202-1214.
[34] J. A. Dallal, "Constructing models for predicting extract subclass refactoring opportunities using object-oriented quality metrics," Information and Software Technology, vol. 54, **(2012)**, pp. 1125-1141.
[35] H. Washizakia and Y. Fukazawab, "A technique for automatic component extraction from object-oriented programs by refactoring," Science of Computer Programming, vol. 56, **(2005)**, pp. 99-116.
[36] G. Bavota, A. De Lucia and R. Oliveto, "Identifying Extract Class refactoring opportunities using structural and semantic cohesion measures," The Journal of Systems and Software, vol. 84, **(2011)**, pp. 397-414.
[37] B. C. Da Silvaa, E. Figueiredob, A. Garcia and D. Nunesa, "Refactoring of Crosscutting Concerns with Metaphor-Based Heuristics," Electronic Notes in Theoretical Computer Science, vol. 233, **(2009)**, pp. 105-125.
[38] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," Journal of Systems and Software, vol. 80, no. 4, **(2007)** April, pp. 571–583.
[39] M. Fowler, "Refactoring: Improving the Design of Existing Code", Boston: Addison-Weasley Longman Publishing Co. Inc., **(1999)**.

Authors

Siti Rochimah, received her PhD degree from Universiti Teknologi Malaysia in 2011. She is currently the Head of Software Engineering Laboratory, Department of Informatics, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia. Her research interest includes software quality and testing, software evolution, and software methodology.

 Copyright ⓒ 2015 SERSC

Siska Arifiani, is currently working toward the master degree at Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia. She received barchelor degree in 2013 from the same institute. Her research interests include software engineering practice and application.

Vika F. Insanittaqwa, is currently working toward the master degree at Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia. She received barchelor degree in 2014 from the same institute. Her research interests include software engineering practice and application.

Copyright ⓒ 2015 SERSC

Copyright ⓒ 2015 SERSC