Advanced Science and Technology Letters
Vol.46 (Games and Graphics 2014), pp.279-284
http://dx.doi.org/10.14257/astl.2014.46.59

Classification and Summarization of Software Refactoring Researches: A Literature Review Approach

Mesfin Abebe and Cheol-Jung Yoo

Chonbuk National University, South Korea
567 Baekje-daero, Deokjin-gu, Jeonju-si,
Jeollabuk-do, Republic of Korea
mesfinabha@gmail.com, cjyoo@jbnu.ac.kr

Abstract. Software refactoring is a technique that transforms a program to improve its internal structure, design, simplicity, understandability or other features without affecting its external behavior. Researchers have studied the different angles of the refactoring activity to develop knowledge. Currently, there is an accumulation of knowledge which needs classification and summarization. We collected and studied refactoring research papers published since 1999 to classify and summarize. This can is help to reveal the research pattern, common concerns, and statistics to formulate better research topics.

Keywords: Software refactoring, Code smell, Design patterns, refactoring tool.

1 Introduction

In the real-world, it is obvious for software to evolve over time to adapt the dynamic environment, therefore software enhancement, modification and adaptation becomes more complex [1, 2, 3, 4]. As the Lehman's stated, the functionality increment in a system always brings a decrease in the quality and an increase in the internal complexity [5, 6]. The branch of software engineering that addresses these problems is called software refactoring [7, 8, 9,10]. This study examined software refactoring research papers from digital libraries such as *IEEE Xplore*, *ScienceDirectory*, *ACM*, *Springer* and *Web of knowledge* and classified based on their content to identify how far each group studied. The reminder of the paper structured as follows: *Section 2* describes the background. S*ection 3*, explains the research method. *Section 4*; provides the results and *Section 5* previews future trends and conclusion.

2 Research Background

Tom Mens provided an overview of existing research area in the field of software refactoring such as: *refactoring activities*, *techniques and formalism*, *types of software artifacts* and *refactoring and software process* [11]. Karim O. Elish et al. proposed a classification of refactoring methods based on their measurable effect on software quality attributes [12]. Another study done by Bart Du Bois et al. use research questions: *what technique to use*, *how to apply in a scalable way*, *dependency between refactorings*, *refactoring at high level of abstraction*, and *how to compare refactoring tools and techniques* [13]. This study investigates research papers published since 1999 and classify and summarize.

3 Research Methodology

3.1 Data Collection Procedure

A total of 467 papers are downloaded from IEEE Xplore (76), ScienceDirect (81), Springer Link (96), ACM DL Digital (156) and Web of Science (58 papers). To narrow down the result the query filtered to: Journal or Conference, English papers, papers since 1999 and match only the Title or/and Abstract. All *Titles* and *Abstracts* are comprehensively analysed using *Jabref 2.9 reference manager*.

3.2 Analyzing the Papers

Through careful reading of the title and abstract, we remained with 169 papers by removing the redundant and unrelated papers. 37 papers (shown in the Annex) selected using systematic sampling for examine the entire paper and for the remaining the title, abstract and conclusion to conduct the classification and the summarization.

4 Conducing the Study

4. 1 Classification of the Software Refactoring Research Papers

Since 1999, researchers have studied software refactoring in various areas. Classification and structuring the findings is important to benefit from these studies. We analyzed all the 169 papers and classified into the following groups:

– *Survey of software refactoring*: review refactoring papers (2, 3, 4, 9, 11)[1].
– *Software refactoring tools*: papers written on refactoring tools (5, 10, 14, 20, 22).
– *Bad smell and Refactoring*: discusses bad smells and refactoring (6, 21, 24, 33).
– *Software artifacts and Refactoring*: study software artifacts such as database, design document and refactoring (1, 34, 37).
– *Agile development and Refactoring*: investigates agile and refactoring (19, 29).

[1] The reference numbers in the curly brace refer to *Annex*.

- *Design pattern and Refactoring*: studies design pattern, micro pattern, anti-pattern and refactoring and their association (43, 12, 27, 29, 30, 36).
- *Test driven development and Refactoring*: considers papers study test driven development and refactoring (15, 26, 28, 31, 35).
- *Software refactoring and System Evolution*: study papers that talk about software refactoring and system evolution (13, 23).
- *Software metrics and Refactoring*: studies software attributes and how they are affected by software refactoring (8, 16, 17, 25, 32).

Table 1. The paper classification statistics.

No	Digital Database	No. of papers
1	Survey of software refactoring	10
2	Software refactoring tools	18
3	Bad smell and Refactoring	23
4	Software artifacts and Refactoring	14
5	Agile development and Refactoring	29
6	Design pattern and Refactoring	17
7	Test driven development and Refactoring	15
8	Software refactoring and system evolution	7
9	Software metrics and refactoring	21
	Total	***169***

4.2 Finding of the Detailed Analysis Process

The following are the analysis process outputs of the *169 papers* where refactoring studies are contribute a significant amount of knowledge.

- Researchers effectively showed database and HTML document refactoring.
- Though it is not an exhaustive list, the driving forces of system refactoring are identified: cost, profits, suppliers, information, technical or futures.
- Design pattern, micro pattern, anti-pattern, code smell and software refactoring association and dependency are investigated to a certain level.
- Test driven development (TDD) and how and when to apply refactoring with unit testing to make the TDD more effective and efficient.
- How formalisms help to guarantee program correctness and preservation.
- General and specific characteristics of the refactoring tools are provided.
- Identification of which refactoring to apply is dependent on the particular application domain type e.g. Web based.
- How software external qualities affected after refactoring is studied theoretically & empirically.
- Some researchers show how to use code complexity analysis to detect whether classes need a refactoring or not.
- Studies indicate that unit testing is the commonly used efficient technique to validate the preservation of the internal behavior after a refactoring.
- The difficulty of merging and integration is the main cause not to do refactoring.
- A better validation technique is more important than having best refactoring tool.

5 Conclusion and Future Work

In software development activity, refactoring is highly desirable to assure the quality of the software process and product. In this paper, we have classified and examined 169 software refactoring related research papers that are published over the past fifteen years. The following are areas which need further investigation:

- How to do refactoring of testing artifacts; requirement and design model.
- How refactoring shall be used in small, medium and big project and/or company.
- Rectify the confusion with different researchers about refactoring and the external or internal quality of software and how they are affected.
- Establishing guidelines are needed which contain and support system refactoring.
- How refactoring can fit well in the linear waterfall or incremental spiral models.
- Studying what fraction of code modification is refactoring need further investigation.
- Identifying the most frequent refactoring to narrow down the refactoring option.
- Classifying refactoring method based on software quality attributes needs theoretical and practical research.
- Determining which code smell is effective to indicating the need of refactoring.
- Currently there a tremendous interest to apply refactoring related concurrency, parallelism, mobile platforms, web-based, cloud computing and GPU etc.
- How can we improve programmers experience in using refactoring tools?
- How can we improve programmers' knowledge and skill of refactoring?

Finally, the above directions can be a candidate research area for the researchers and practitioners to overcome the limitations of the software refactoring activities.

References

1. Coleman D. M., Ash D., Lowther B., Oman P. W.: Using metrics to evaluate software system maintainability. IEEE Computer, vol. 27, no. 8, pp. 44-49 (1994).
2. Guimaraes T.: Managing application program maintenance expenditure. Comm. ACM, vol. 26, no. 10, pp. 739-746 (1983).
3. Rugaber S.: Program comprehension. Encyclopedia of Computer Science and Technology, vol. 35, no. 20, pp. 341-368 (1995).
4. Introduction to Refactoring, http://sourcemaking.com/refactoring/defining-refactoring.
5. Lehman M.: Laws of program evolution - rules and tools for programming management. InProceedings Infotech State of the Art Conference, Why Software Projects Fail, vol. 11, pp. 1–25 (1978).
6. Lehman M., Ramil J.: Rules and tools for software evolution planning and management. Annals of Software Engineering, vol. 11, no. 1, pp. 15-44 (2001).
7. Opdyke W. F.: Refactoring: A Program Restructuring Aid in Designing Object-Oriented Application Frameworks. PhD thesis, University of Illinois at Urbana-Champaign, (1992).
8. Martin Fowler. Refactoring: Improving the Design of Existing Programs. Addison-Wesley, (1999).
9. Murphy-Hill E.: Improving usability of refactoring tools. In OOPSLA '06: Companion to the 21st ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications, pages 746–747. ACM Press (2006).

10. Elish K. O., Alshayeb M.: Investigating the Effect of Refactoring on Software Testing Effort. IEEE Computer Society (2009).
11. Mens T.: A Survey of Software Refactoring. IEEE, vol. 30, no. 2, February (2004).
12. Elish K., Alshayeb M., Karim O.: A Classification of Refactoring Methods Based on Software Quality Attributes. Arab J Sci Eng 36: 1253-1267, (2010).
13. Bois B. D., Van Gorp P., Amsel A., Van Eetvelde N., Stenten H., Demeye S.: A Discussion of Refactoring in Research and Practice, (2006).

Annex

Table 2. List of papers selected for detailed analysis.

No	Digital Database	Author	Year
1	Refactoring of a Database	Ayeesha etal	2009
2	A Field Study of Refactoring Challenges and Benefits	Miryung etal	2012
3	A Survey of Software Refactoring	Tom Men al.	2004
4	A Survey of software testing in refactoring based software models	Pandimurug	2011
5	An expert system for determining candidate software classes for refactoring	Yasemin etal	2008
6	Bad Smelling Concept in Software Refactoring	Ganesh B.	2009
7	Design Patterns in Software Development	MU Huaxin	2011
8	Empirical investigation of refactoring effect on software quality	Mohammad	2009
9	Empirical Support for Two Refactoring Studies Using Commercial C# Software	M. Gatrell, ,	2011
10	Evaluating software refactoring tool support	Erica Mealy	2006
11	Formal specification of extended refactoring guidelines	Wafa Basit,	2012
12	From Software Architecture to Design Patterns	Jing Wang,	2005
13	Identifying Refactoring Sequences for Improving Software Maintainability	Panita	2012
14	Improving Usability of Software Refactoring Tools	Erica Mealy.	2007
15	Investigating the Effect of Refactoring on Software Testing Effort	Karim O.	2009
16	Quantifying Quality of Software Design to Measure the Impact of Refactoring	Tushar S	2012
17	Refactoring – Does it improve software quality	Konstantinos	2007
18	Refactoring : Emerging Trends and Open Problems	Tom Mens	2003
19	Refactoring Practice: How it is and How it should be Supported	Zhenchang	2006
20	Refactoring Tools: Fitness for Purpose	Emerson	2008
21	Schedule of Bad Smell Detection and Resolution: A New Way to Save Effort	Hui Liu,	2012
22	Software refactoring at the package level using clustering techniques	A. Alkhalid,	2010
23	Strengthening Refactoring: Towards Software Evolution with Quantitative	Sérgio Bryto	2009
24	Using Software Metrics to Select Refactoring for Long Method	Panita	2011
25	A Classification of Refactoring Methods Based on Software Quality Attributes	Karim O.	2011

26	A test case refactoring approach for pattern-based software development	Peng-Hua	2012
27	Anti-pattern Based Model Refactoring for Software Performance	Davide	2012
28	Automated Acceptance Test Refactoring	RodrickBorg	2011
29	A Role for Refactoring in Software Engineering?	Tony Clear	2005
30	Drivers for Software Refactoring Decisions	Mika V.	2006
31	Testing During Refactoring: Adding Aspects to Legacy Systems	Panita	2006
32	Impact of Refactoring on Quality Code Evaluation	Francesca	2011
33	Perspectives on Automated Correction of Bad Smells	Javier Pérez	2009
34	Rank-based refactoring decision support: two studies	Liming Zhao	2011
35	Refactoring with Unit Testing: A Match Made in Heaven?	Frens	2012
36	The Impact of Refactoring to Patterns on Software Quality Attributes	Mohammad	2011
37	UML model refactoring: a systematic literature review	Mohammed	2013