Impact of Refactoring on Code Quality by Using Graph Theory: An Empirical Evaluation

Anam Shahjahan, Aisha Zafar Ahmad, WasiHaider Butt, Usman Qamar
College of electrical and mechanical engineering, NUST
Pakistan
anamshahjahan@hotmail.com, ash.zafar90@gmail.com
wasi@ce.ceme.edu.pk, usmanq@ceme.nust.edu.pk

ABSTRACT: *Refactoring is the process of improving code quality without affecting its external behaviour and by changing its internal structure. Refactoring is done to improve code quality and structure.*

In this research, we have proposed a new method of code refactoring by using graph theory techniques. Previously manual methods were used to identify the classes with high impact in refactoring. The proposed method has been implemented and applied on selected software projects developed in java. Validation has been done by surveying software professionals to measure improve in code quality.

Keywords: Graph Theory, Code Refactoring, Refactoring Techniques, Analysability, Changeability, Time Behaviour, Resource Utilization

Received: 10 March 2015, Revised 9 April 2015, Accepted 15 April 2015

1. Introduction

Refactoring definition of fowler's [1] is to make changes in the internal structure while preserving its external behaviour. This makes the software easy to understand and easy to maintain. Refactoring process consists of three steps: refactoring participant's identification, refactoring effect validation and refactoring application. Figure 1 shows the process of refactoring [2].

Constant changes and evaluation are required by any using software. Software codes become more difficult and die away from its design when that system is changed and redesign to new essentials. Due to this maintenance cost of software system is crucial part of development [3]. 90% cost of any software is of maintenance [4]. So there is a need of a methodology that minimize this cost and lessen the complexity of software and that make better software quality [3]. Source code of any software system that is a large system is modified and changed with the passage of time. There are many reasons for this like: changing of requirements, modification in design, and due to some errors.

Because of these reasons maintenance is difficult and its cost is also high. So to simplify these activities, refactoring is the good technique that upgrade internal structure while maintaining external operations [5]. For openly and commercially available software system refactoring has been used. For example for rewriting code Microsoft preserves 20% of overall cost [6]. Software code quality attract customer towards adapting that software. There is a relationship between customer interest and matrices of source code. Developer should supervise matrices that can improve code quality [7].

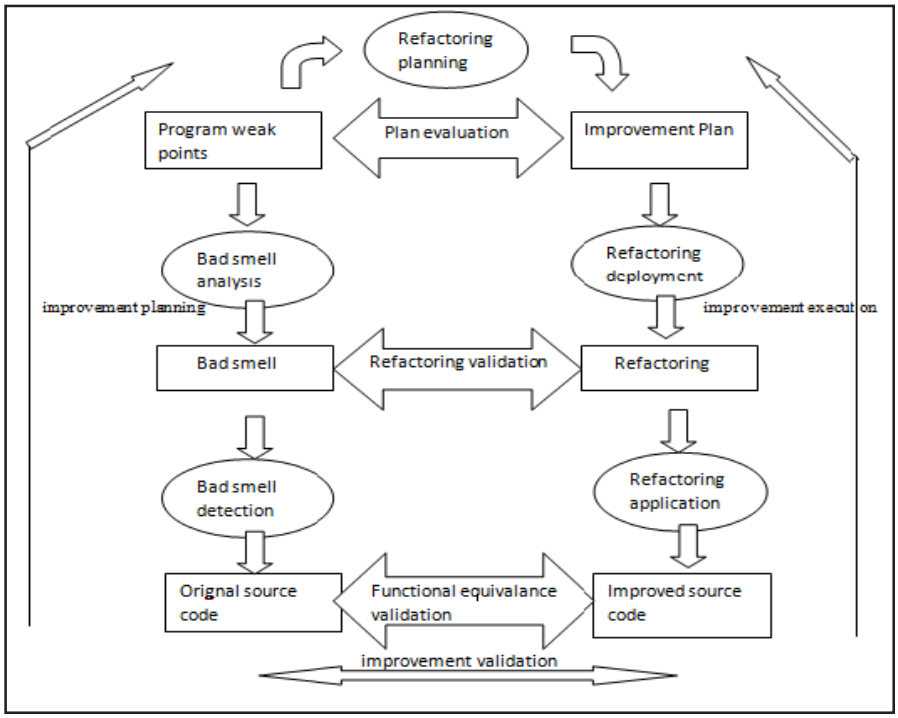

Figure 1. Process of Refactoring [2]

To assess the effect of refactoring on quality of software various studies have been performed [2], [8]. Refactoring improves code quality that is declared by various studies. But these studies didn't provide any evidence. So Conformation of these studies are hardly to be found[9]. All refactoring methods that used earlier did not improve software quality. So there is a need of study which can improve software quality and can decrease maintenance cost of software. The purpose of this study is to improve software quality by conducting set of experiment on code refactoring. This helps developers to improve quality of software.

Structure of remaining paper is as follows: Section 2 is of related work. Purposed method is described in section 3.Experimentation and analysis of this study is described in section 4. Results of these experiments are in section 5. Finally section 6 is of conclusion and future work.

2. Related Work

Several studies have been conducted to improve software quality and to evaluate the impact of refactoring on software code. These studies claim that refactoring improve software quality. But some researchers provide quantitative measures and some not.

Teng Long and Lin Liu[10] proposed a high quality code using refactoring and obfuscation technique. Their study analyzes how refactoring and obfuscation use reverse transformations to improve quality and security of software code. They proposes a systematic modeling approach based on i* model to support the selection of refactoring techniques and obfuscation methods under different social, environmental and operational situations. i* model is applied to help us link the high-level designer/ attacker intentions with the low-level refactoring/obfuscation transformations. According to them refactoring can not only improve the design of existing code to enhance software quality, but can also bring much convenience and efficiency to programmers' work. After applying refactoring and obfuscation techniques, it can be seen that although both use opposite action but goal is same to improve code quality and security. But they does not provide quantitative measures.

To improve software quality S.A.M.Rizvi and Zeba Khanam [11] defines activities which is to be followed for refactoring. First is preparation phase and the second is searching phase. Both of these phases consists of defining motive of refactoring and selection of refactoring pattern. Their technique is used by developers to refactor and to improve quality of legacy code. But thisstudy also not provide quantitative measures to prove their work.

Code clones need to be removed to enhance maintainability. For this Minhaz Fahim Zibran and Chanchal Kumar Roy [12] proposed a refactoring effort model, and constraint programming approach for conflict-aware optimal scheduling of code clone refactoring. To calculate refactoring effort they proposed effort model in OO. They implement model using CP technique and modelled the clone refactoring problem as a CSOP by considering different categories of refactoring constraints. Their work is unique because this did not applied and used previously in any research. But their work is not evaluated experimentally.

Previous research used static analysis for suggesting which candidate could be use for refactoring that cannot get runtime information. Shuhei Kimura, Yoshiki Higo[13] proposed a technique to find refactoring candidates by analyzing method traces. Their proposed technique has following steps: Obtain a method trace and divided method trace, Count invocations of each method in the same phase, Visualize phase count data, Detect refactoring candidates. For evaluation this technique has been applied on two software system and this increased code quality.

Refactoring improves code quality but there are no quantitative measures that proves this. S.H. Kannangara, W.M.J.I. Wijayanayake[4] proved this assumption by experimental approach. According to their experiment quality of software code improved by refactoring. They prove their research with quantitative measures.

Raed Shatnawi, Wei Li[6] proved that software code quality improves by refactoring and they used quality model for their research. They provide quality heuristics and validate it with two open source systems. Developers can use those heuristics to considering that which refactoring improves software code quality.

After analysis of all above studies, there are many concerns:

• All studies did not give same result of refactoring.

• Internal quality factors and quality models are used by most of studies to check or to validate the external quality attributes.

• Most of studies used less number of techniques to refactoring.

• Large number of techniques are used by only one study[4]. They used ten techniques.

• But the results of study[4] are not good. Results of this study are not as needed. Those can be further improved.

After analyzing all above studies it is noted that software code quality can improved with refactoring. Good quality code decreases maintenance cost. But maintenance cost decreases with all above techniques is lower. So there is a need of technique that improve code quality and decreases maintenance cost higher than above techniques. This study use graph theory techniques to solve all above problem.

3. Proposed Methodology

Software code quality is very important for a running software. Code refactoring is a technique to increase quality of a software. Objective of this study is to introducing a new technique that can improve code quality. This research depends upon graph theory techniques, code refactoring and external quality factors.

Graph theory is consists of graph that have vertices and edges. Relation between edges an vertices are shown by a graph. For this research we used centrality measures from graph theory techniques. Those centrality measures are degree centrality, betweeness centrality, closeness centrality, eigenvector centrality and clustering coefficient. Degree centrality describe how effectively nodes and edges are connected. Closeness centrality find the distance between a vertices and all other vertices that can be reached from that particular node [14]. Betweeness centrality is the measure of how effectively a vertics act as a mediator between the interaction to other vertices [14].Eigenvector centrality is the measure of effect that on vertices has on other vertices[15]. Clustering coefficient measures the degree to which a graph is connected[16].

This research uses quality factors to check whether code quality improves or not. For this research we used 8 java projects. Tool used for this research is eclipse with STAN plug-in. we took jre file of projects and run them with STAN on eclipse This generate class diagram. We created dependency metric sheet which shows dependency between classes of project. After that apply graph theory techniques on dependency metric sheet. For this we use NodeXL. By applying graph theory techniques on metric we calculate centrality measures. These centrality measures consists of degree centrality, Betweens centrality, Closeness

centrality, Eigenvector centrality and Clustering coefficient. We calculate these centrality measures and find that which class has greater value. Then refactoring applied on those classes which has greater value. Flow diagram of this procedure is shown in Figure. 2.

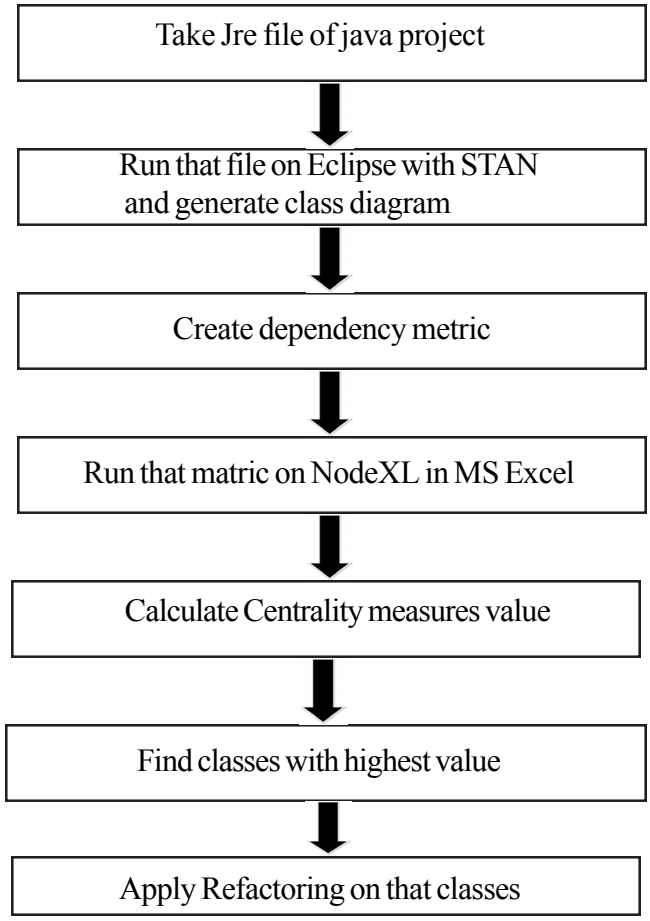

Figure 2. Flow diagram of proposed methodology

Fowler [17] has proposed 72 refactoring techniques. 43 refactoring techniques among fowler [17] has been evaluated by recent study [6]. These techniques were ranked.

For this study 10 refactoring techniques are used from them. These techniques applied in refactoring process. Selected techniques are:

- Introduce Local Extension
- Duplicate Observed Data
- Replace Type Code with Subclasses
- Replace Type Code with State/Strategy
- Replace Conditional with Polymorphism
- Introduce Null Object
- Extract Subclass
- Extract Interface
- Form Template Method

• Push Down Method[4]

We evaluated refactoring by using four quality factors that are:

• Changeability

• Analysability

• Time behaviour

• Resource utilization [4].

We validated this research by surveying and for this we design questionnaire. For evaluation we used variables and those are:

• Memory consumption

• Time to fix bugs

• Execution time

• Obtained marks for question paper [4].

The hypothesis used by this study was on above 4 quality factors. Four quality factors was measured by lower and higher impact.

This experiment was done by 100 experts. Those experts have programming skills. CASE & CARE programming experts was selected for this research. All experiment was done in testing environment.

General procedure of this research was very simple. Designed questionnaire was distributed between experts. 1-2 hours was given to them. After that time questionnaire was collected by them and results evaluated.

• Firstly results was evaluated for analysability. Results was found on the base of marks obtained for paper.

• Secondly changeability results was evaluated. That evaluation was base on time to fix bugs.

• Thirdly quality factor (time behaviour) was evaluated and that was on the base of execution time.

• Quality factor (resource utilization) was evaluated fourthly and that was on the base of memory consumption.

4. Experiment and Analysis

For each refactoring technique we did experiment and to this process 8 java projects are used. Experiment will be done with testing staff. Firstly we presented techniques that are used for refactoring. Secondly we presented applications. Thirdly question paper was distributed and after that results are gathered and calculated.

Finally data was analysed according to impact of each refactoring technique on quality factors. For each quality factor analysis was done and data was provided.

4.1 Analysis for Changeability

Results of quality factor (changeability) was evaluated on the base of time to fix bugs(variable). Results of that evaluation are given in the following table 1.

Refactored code	Yes	No
Difficult	18.03%	81.96%
Easier	77.04%	22.95%

Table 1. Impact of refactoring on changeability

By hypothesis testing under changeability it is not concluded that refactored code could be easily changed than non-refactored code.

4.2 Analysis for Analysability

Results of quality factor (analysability) were evaluated on the base of marks obtained for questionnaire (variable). Results of that evaluation are given in the following table 2.

Refactored code	Yes	No
Difficult	24.59%	75.40%
Easier	75.40%	24.59%

Table 2. Impact of refactoring on Analysability

By hypothesis testing under analysability it is not concluded that analysability of refactored code is higher than non-refactored code.

4.3 Analysis for Time behaviour

Results of quality factor (time behaviour) were evaluated on the base of execution time (variable). Results of that evaluation are given in the following table 3.

Refactored code	Yes	No
Difficult	72.13%	27.86 %
Easier	4.91%	93.44%

Table 3. Impact of refactoring on Time behaviour

By hypothesis testing under time behaviour it is not concluded that response time of refactored code is shorter than non-refactored code.

4.4 Analysis for Resource utilization

Results of quality factor (resource utilization) were evaluated on the base of memory consumption (variable). Results of that evaluation are given in the following table 4.

Refactored code	Yes	No
Difficult	26.22%	68.85%
Easier	67.21%	32.78%

Table 4. Impact of refactoring on Resource utilization

By hypothesis testing under resource utilization it is not concluded that efficient utilization of system resources is higher for refactored code than non-refactored code. There were not enough results to claim that all four quality factors gave better results.

5. Results And Discussion

Results are calculated for each quality factor on the base of refactored and non-refactored code. Results are summarized in table 10. Percentage of unchanged deteriorates and improvement was presented in the table 5.

Quality factors	Deteriorate	Un changed	Improved
Analysability	30%	0%	70%
Changeability	40%	0%	60%
Time behaviour	33%	0%	67%
Resource utilization	50%	0%	50%

Table 5. Summary of effect of refactoring on external measures – Analysis of each refactoring techniques

Results in the above table are analysed. It is concluded from the results that analysability, time behaviour and resource utilization have contrary impact. But the impact of this study is lesser than the study[4].

6. Conclusion and Future Work

This study was conducted to assess the impact of refactoring on code quality. this was done to find the improvement in maintenance of software. To achieve the objective quality factors are used to assess the impact of refactoring. Those quality factors that are used in this study are: analysability, changeability, time, behaviour and resource utilization. Four variables (marks obtained from questionnaire, time to fix bugs, execution time, memory consumption) and ten refactoring techniques are used to assessing the impact. Approach of research was experimental in this study.

Hypothesis techniques was used to compare the results that were got. Results were tested against hypothesis. Those results shows that analysability of refactored code is not higher than non-refactored code but those results was improved than the study [4]. Changeability of code is not easier for refactored but that was improved than the study [4]. Response time is also higher for refactored code but results of response time was improved than the study [4]. Moreover the resource utilization is also not lower for refactored code but result of resource utilization is also improved than study [4].

The results given in table 10 was much better than study[4]. Experiment for this study is done from experts and testers. So there was improvement in the results of this study.

This study improved results but it is needed more to address the effect of refactoring. Refactoring techniques selected from the classification that is done in the previous paper [4]. In future there is a need of study to find the techniques of refactoring that are commonly used in the field. The results of refactoring on code quality would be more better by using those techniques that are common. This will decrease maintenance cost and increase code quality. That code would increase outcome and results of that code will be better.

Acknowledgment

We acknowledge the Center For Advance Studies In Engineering(CASE) & Center For Advance Research In Engineering(CARE) for helping us in conducting a survey.

References

[1] Raimund Moser., Pekka Abrahamsson., Witold Pedrycz., Alberto Sillitti., Giancarlo Succi. (2008). A case study on the impact of refactoring on quality and productivity in an agile team, *IFIP International Federation for Information Processing* , LNCS 5082, p. 252–266.

[2] Kataoka, Y., Imai, T. , Andou, H., Fukaya, T. (2002). A quantitative evaluation of maintainability enhancement by refactoring, 576 - 585, 1063-6773, .

[3] Tom Mens., Tom Tourwe. (2004). A Survey of Software Refactoring, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 30[2], FEBRUARY .

[4] S. H. Kannangara., W. M. J. I. Wijayanayake. (2013). Impact of Refactoring on External Code Quality Improvement: An Empirical Evaluation, International Conference on Advances in ICT for Emerging Regions, (ICTer): 060 - 067,.

[5] Ali Ouni1., Marouane Kessentini2., Houari Sahraoui1. (2013). Search-based Refactoring Using Recorded Code Changes, 17th *European Conference on Software Maintenance and Reengineering*,1534-5351/13 $26.00 © 2013 IEEE.

[6] Raed Shatnawi., Wei Li. (2011). An Empirical Assessment of Refactoring Impact on Software Quality Using a Hierarchical Quality Model, *International Journal of Software Engineering and Its Applications*. 5 [4], October, .

[7] Paulo Meirelles., Carlos Santos Jr., João Miranda., Fabio Kon., Antonio Terceiro.,Christina Chavez. (2010). A Study of the Relationships between Source Code Metrics and Attractiveness in Free Software Projects, 11-20, 978-0-7695-4273-7.

[8] Dirk Wilking., Umar Farooq Khan., Stefan Kowalewski. (2007). An empirical evaluation of refactoring, *e-Informatica Software Engineering Journal*, 1 [1].

[9] Tom Mens., Serge Demeyer., Bart Du Bois., Hans Stenten., Pieter Van Gorp. (2003). Refactoring: Current Research and Future Trends, *Electronic Notes in Theoretical Computer Science* 82 [3] .

[10] Teng Long., Lin Liu.,Yijun Yu., Zhiguo Wan. (2010). Assure High Quality Code using Refactoring and Obfuscation Techniques, *Fifth International Conference on Frontier of Computer Science and Technology*, 978-0-7695-4139-6/10 $26.00 © 2010 IEEE

[11] Rizvi, S. A. M., Zeba Khanam. (2011). A Methodology for Refactoring Legacy Code, 978-1-4244-8679-3/11/$26.00 ©2011 IEEE.

[12] Minhaz Fahim Zibran., Chanchal Kumar Roy. (2013). Conflict-aware Optimal Scheduling of Code Clone Refactoring: A Constraint Programming Approach, 11th IEEE International Working Conference on Source Code Analysis and Manipulation, IET Softw., 7 [3], p. 167–186.

[13] Shuhei Kimura.,Yoshiki Higo., Hiroshi Igaki., Shinji Kusumoto. (2012). Move Code Refactoring with Dynamic Analysis, 28th *IEEE International Conference on Software Maintenance* (ICSM), 978-1-4673-2312-3/12/$31.00 c 2012 IEEE.

[14] Dil Muhammad and Akbar Hussain. Destabilization of Terrorist Networks through Argument Driven Hypothesis Model, *JOURNAL OF SOFTWARE*, 2 (6), DECEMBER 2007.Ulrich Eberle. Terrorist Networks, //// thesis

[15] Nisha Chaurasia., Akhilesh Tiwari. (2013). Efficient Algorithm for Destabilization of Terrorist Networks, *I.J. Information Technology and Computer Science*,12, 21-30

[16] Emilio Ferrara., Pasquale De Meo., Salvatore Catanese.,Giacomo Fiumara. (2014). Detecting criminal organizations in mobile phone networks, arXiv:1404.1295v1 [cs.SI] 3 Apr .

[17] International Standards. (2001). ISO/IEC 9126-1 Standard[Online]Available: http://webstore.iec.ch/preview/infoisoiec91261%7Bed1. 0%7Den.pd