TO ANALYZE THE IMPACT OF REFACTORİNG ON THE QUALİTY OF OPEN-SOURCE SOFTWARE

Manpreet Kaur[1], and Dhavleesh Rattan[2]
[1]Department of Computer Science and Engineering, Baba Banda Singh Bahadur Engineering College, Fatehgarh Sahib, Punjab, India
[2] Department of Computer Science and Engineering, Punjabi University, Patiala, Punjab, India

Abstract. Refactoring is a process of changing the source code of a program so that the internal structure of the code improves without changing the external behavior of the program. It cleans the software by eliminating bad smells from the software and makes the software easy to understand, and maintain, and reduces its maintenance cost. The presence of bad smells indicates some problematic code in the software that needs to be eliminated. Tools are available to eliminate such codes from the software. In the current work, refactoring is performed on six open-source software to eliminate bad smells and then the analysis is performed to know the impact of refactoring on the quality of software in terms of external and internal quality attributes. The complexity of the software has been calculated and compared with the initial complexity before applying refactoring. Similarly, quality attributes are calculated before performing any type of refactoring, and the same are calculated again after applying suitable refactoring based on the type of bad smell. Last, a comparison of changes has been done to know the refactoring impact on the quality of open-source software.

Keywords: Refactoring, bad smells, software quality attributes, open source software.

I. I INTRODUCTİON

Software refactoring is also known as software restructuring because there is not much difference between these two terms. Software refactoring is the super-set of software restructuring because it contains all the methods that can be used in restructuring and also adds new specific techniques related to object-oriented programming [1]. Fowler et al. [2] discussed different bad smells that can be present in the source code of software and gave instructions to eliminate these bad smells. Refactoring is defined as a process of modifying the source code in such a manner that its external activities or behavior does not alter but its internal structure is improved [2]. They defined different occasions when a programmer must perform refactoring. Developers perform refactoring when code is repeated, want to add new functionality, a bug needs to be detected, during a code review, etc. Refactoring is also done if the present code or design is difficult to understand or maintain[4]. Refactoring will generate code that is easy to understand.

Refactoring is the behavior-preserving and structure-transforming process [1]. The main aim of refactoring is not to add new functionality to the software but to increase the code understandability and maintainability by removing the bad smells from the software. Bad smells in the software source code are dissatisfactory structures that decrease the software quality and make it less maintainable [2]. The software maintainability is essential because it affects the cost of future improvement events. If the programmer does not understand when refactoring needs to be applied then refactoring will not give full benefits. To make it easy for the software programmer, the authors define a list of bad smells to decide whether software needs to be refactored or not.

Roberts[3] discussed that refactoring is a program transformation process that contains two conditions or steps (pre-conditions and post-condition) that need to be satisfied to easily apply the refactoring techniques. Refactoring contains at least two steps - analysis and execution. The first step is investigation, where the code required to be refactored is evaluated to decide whether the pre-conditions are satisfied or not. It means to detect whether a bad smell occurs in the software or not. If the first step satisfies the precondition then the next step is execution, which is to refactor the source code. It means eliminating those bad smells that occur in the first step by applying the refactoring techniques.

Our work aims to detect bad code smells from six open-source software written in Java and eliminate these smells using refactoring techniques. Then we analyzed the effect of refactoring on the quality of software using various external and internal quality attributes[5]. The rest of the paper presents the work as follows. Section II presents the research methodology to eliminate bad smells and evaluate refactoring impact on software quality. Results are discussed in Section III. Section IV provides an overview of related work and Section V concludes the current work and gives future directions.

II. RESEARCH METHODOLOGY

Current experimentation has used the JDeodorant tool[11] to detect and refactor bad smells from open-source software. JDeodorant is available as a plugin in Eclipse[10]. Quality attributes are obtained using the Metrics plugin[12] available in Eclipse. Firstly, internal quality metrics (Object Oriented Metrics) of source code are calculated. After that bad smells are detected and then changes are made to the code to remove bad smells using a single refactoring technique[9]. After applying refactoring, the software is tested to check whether its external behavior is the same as desired. If an error comes, undo the last code changes and perform refactoring differently. Once all the refactoring techniques are applied, measure the internal quality metrics of software (Object Oriented Metrics) and then, compare the internal quality metrics of software before and after applying refactoring techniques. Also, calculate and compare the external quality metrics [6] to predict the refactoring impact on software quality.

III. EXPERIMENT RESULTS AND DISCUSSIONS

The effect of refactoring on the quality of software has been analyzed by comparing the initial quality attributes with the value of quality attributes after refactoring. Six open-source software written in Java - JLOC (beta) [13], JNFS (0.1) [14], RabtPad [15], JHotDraw (version-5.2) [16], and JFreeChart (Version-0.5 and 0.6) [17]) are used. The bad smells Long Method[24], God Class, Type Checking[8] and Feature Envy[24], have been detected in these software using the JDeodorant plugin. Metrics Plugin is used to measure the metrics of the software. Before applying any type of refactoring, internal quality attributes are measured. If bad smells are detected in the software, refactoring is applied to remove bad smells. Internal and external quality metrics[6] are calculated again after refactoring and are compared to analyze the refactoring impact on open-source software.

3.1 Number of Bad Smells Detected in Software

Four different types of bad smells are detected in six different open-source software using the JDeodorant plugin in Eclipse. Table 1 provides information about the bad smells detected in each open-source software. The number of long-method bad smells is more than the other three bad smells in all software systems. As the system grows from JFreeChart 0.5 to JFreeChart 0.6, the number of all types of bad smells increases.

Table 1: Number of bad smells detected

Software	Size(LOC)	God Class	Feature Envy	Long Method	Type Checking	Total
RabtPad	5256	10	5	27	6	48
JLOC	303	1	-	6	-	7
JNFS 0.1	423	4	2	-	-	6
JhotDraw 5.2	9424	23	5	135	2	165
JFreeChart 0.5.6	6978	21	19	81	4	125
JFreechart 0.6.0	8666	26	24	135	5	190

3.2 Internal Quality Attributes

Before removing any bad smells from the software, the value of internal quality attributes is calculated and given in Table 2 After removing bad smells using a suitable refactoring technique, the value of internal quality attributes is again calculated and given in Table 3. Table 3 gives the information about effect of refactoring on metrics value. For better understandability value of cohesion and encapsulation metrics should be high for good quality software and all other metrics' value should be low. '-ve' shows the negative effect of refactoring on internal quality attributes that can deteriorate the quality of software and increase the maintenance cost by making code more complex. Also, the software may not be easy to understand by the programmer. '+ve' shows the positive result of refactoring on internal quality attributes and it can help to decrease the maintenance cost. This depicts that if the complexity of software is reduced then the programmer can easily understand the software and maintain the software. If the software is not easy to understand then we cannot easily maintain or modify the software because we do not know how the software will work.

TABLE 2: Internal Quality Metrics before Refactoring (Mean Value)

Software Metrics	RabtPad	JLOC	JNFS	JHotDraw	JFreeChart0.5	JFreeChart0.6
Efferent Coupling	0.647	0	1.5	12	6.667	12.2
Cohesion	0.985	0.947	0.892	0.991	0.987	0.991
WMC	18.871	9.2	5.875	13.541	13.771	18.64
Hierarchies	2.774	1.2	2.625	2.811	2.711	2.547
Design Size	1.824	5	4	13.455	13.833	15
Abstraction	0.044	0.167	0	0.131	0.187	0.2
Polymorphism	0.226	0.2	0.5	1.277	0.181	0.307
Messaging	7.581	5	3.875	8.311	7.723	9.76
Inheritance	2.981	4	12.903	15.365	2.343	3.145

Composition	8.581	4.6	5.25	1.831	2.542	3.187
Encapsulation	0.481	1	0.953	0.501	0.826	0.435

TABLE 3: Internal Quality Metrics after Refactoring

Software Metrics	RabtPad	JLOC	JNFS	JHotDraw	JFreeChart0.5	JFreeChart0.6
Efferent Coupling	0.882(-ve)	0(-)	2(-ve)	12.909(-ve)	8.333(-ve)	15.6(-ve)
Cohesion	0.987(+ve)	0.963(+ve)	0.922(+ve)	0.992(+ve)	0.989(+ve)	0.993(+ve)
WMC	16.80(+ve)	11(-ve)	4.867(+ve)	13.908(-ve)	14.115(-ve)	17.649(+ve)
Hierarchies	2.39(+ve)	1.167(+ve)	1.867(+ve)	2.681(+ve)	2.51(+ve)	2.32(+ve)
Design Size	2.412(-ve)	6(-ve)	7.5(-ve)	14.818(-ve)	16(-ve)	19.4(-ve)
Abstraction	0.062(-ve)	0.143(+ve)	0(-)	0.127(+ve)	0.187(-)	0.195(+ve)
Polymorphism	0.171(+ve)	0.167(+ve)	0.267(+ve)	1.153(+ve)	0.177(+ve)	0.289(+ve)
Messaging	7.707(-ve)	6(-ve)	3.733(+ve)	8.607(-ve)	8(-ve)	10.124(-ve)
Inheritance	2.218(+ve)	2.783(+ve)	7.152(+ve)	13.396(+ve)	2.2125(+ve)	2.854(+ve)
Composition	6.634(+ve)	4(+ve)	3.267(+ve)	1.718(+ve)	2.281(+ve)	2.526(+ve)
Encapsulation	.490(+ve)	1(-)	0.959(+ve)	0.5136(+ve)	0.832(+ve)	0.445(+ve)

3.3 Impact on Complexity of software

The complexity of software indicates how much time and effort is required to understand and maintain the class [6]. It is used to measure the difficulties related to incorporating changes in source code during the maintenance phase. Weighted methods per class are the amount of effort required to understand the class [18]. '-ve' shows the negative effect of refactoring, which means more effort is required to maintain the software because of high complexity, and '+ve' shows the positive result of refactoring on complexity it means low complexity and less efforts are required to maintain the software. The values of McCabe Cyclomatic Complexity and Weighted methods per Class Complexity are given in Table 4 to detect the impact of refactoring on software complexity.

TABLE 4: Complexity before and after Refactoring

Software	McCabe Cyclomatic Complexity		Weighted methods per class	
	Before Refactoring	After Refactoring	Before Refactoring	After Refactoring
RabtPad	2.321	2.069 (+ve)	18.871	16.805 (+ve)
JLOC	1.769	1.784 (-ve)	9.2	11 (-ve)
JNFS	1.382	1.237 (+ve)	5.875	4.86 (+ve)
JHotDraw	1.549	1.533 (+ve)	13.541	13.90 (-ve)
JFreeChart0.5	1.709	1.656 (+ve)	13.771	14.11 (-ve)
JFreeChart0.6	1.866	1.714(+ve)	18.64	17.64 (+ve)

3.4 External Quality Attribute

The external quality attributes are calculated using formulas given by Bansiya and Davis [7]. The values of external quality attributes before refactoring are calculated and shown in Table 5. Table 6 shows the refactoring impact on external quality attributes of open source software. '-ve' shows the negative effect of refactoring because it decreases the value of software quality attributes and '+ve'

shows the positive result of refactoring on external quality attributes because it increases the value of software quality attributes. For good quality software, the value of external quality attributes should be high.

As shown in Table 6 reusability and functionality attributes of all the six open source software increased because coupling, cohesion, and design size metrics value increased when refactoring was applied and other attributes(hierarchies and polymorphism) decreased in all the software. After refactoring if the value of the reusability quality attribute is increased then duplication of components is minimized, implementation time is decreased and the functionality attribute is increased the capability of the software system to do its work for which it is intended. Understandability is increased in only RabtPad software. From the above result, we can say that a group of refactoring techniques has a positive impact on reusability and functionality but a negative impact on flexibility, extendibility, understandability, and effectiveness.

TABLE 5: External Quality Attributes Before Refactoring

External Quality Attributes	RabtPad	JLOC	JNFS	JHotDraw	JFreeChart 0.5	JFreeChart 0.6
Reusability	4.787	5.236	3.785	8.131	9.358	9.58
Flexibility	4.362	2.65	2.738	-1.320	-0.098	-1.194
Understandability	-6.64	-4.16	-3.30	-12.84	-10.83	-14.82
Functionality	2.84	2.621	2.52	5.805	5.498	6.193
Extendibility	1.302	2.183	5.951	2.386	-1.978	-4.274
Effectiveness	2.462	1.993	3.921	3.821	1.215	1.454

TABLE 6: External Quality Attributes after Refactoring

External Quality Attributes	RabtPad	JLOC	JNFS	JHotDraw	JFreeChart0.5	JFreeChart0.6
Reusability	5.086(+ve)	6.24(+ve)	5.348(+ve)	8.733(+ve)	10.164(+ve)	11.112(+ve)
Flexibility	3.304(-ve)	2.333(-ve)	1.509(-ve)	-1.663 (-ve)	-0.646(-ve)	-2.381(-ve)
Understandability	-6.222(+ve)	-5.064(-ve)	-4.208(-ve)	-13.665(-ve)	-12.21(-ve)	-17.059(-ve)
Functionality	2.908(+ve)	3.048(+ve)	3.050 (+ve)	6.115(+ve)	5.989(+ve)	7.188(+ve)
Extendibility	0.784(-ve)	1.546(-ve)	2.709(-ve)	0.883(-ve)	-2.878(-ve)	-6.131(-ve)
Effectiveness	1.915(-ve)	1.618(-ve)	2.329(-ve)	3.381(-ve)	1.138(-ve)	1.261(-ve)

Fig. 1 to 6 show the refactoring impact on external quality attributes of open-source software before and after refactoring.

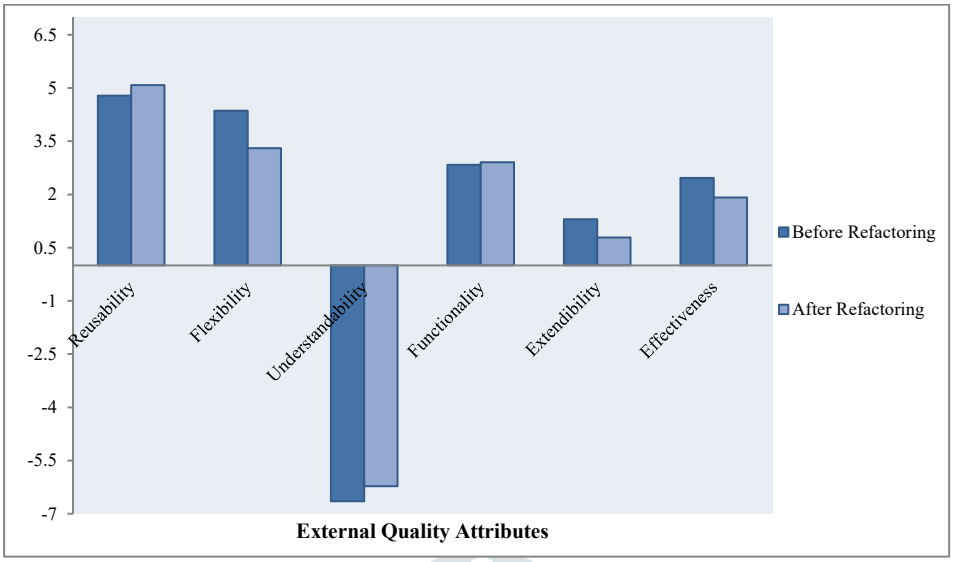

Fig. 1 Show the impact of Refactoring on RabtPad

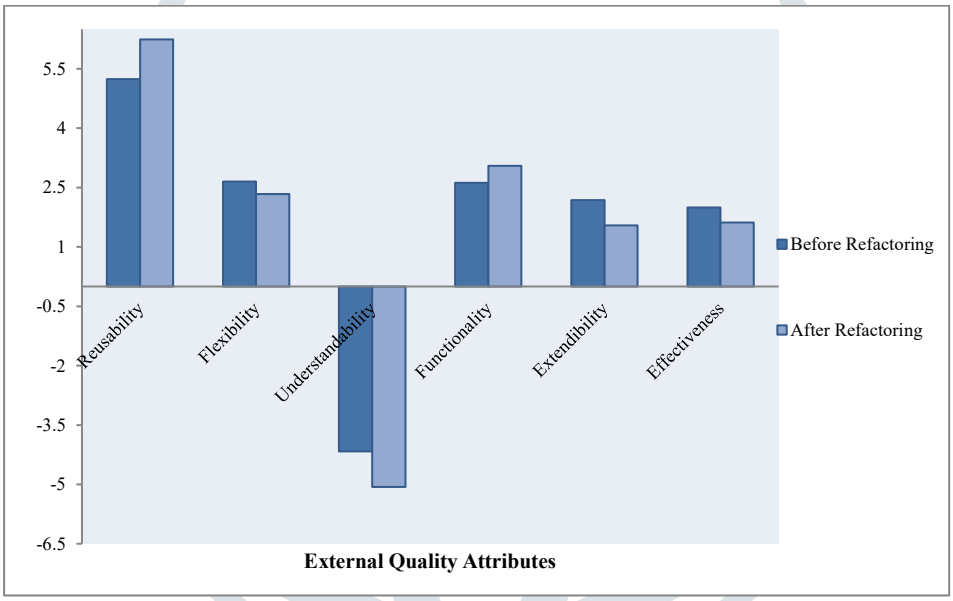

Fig. 2 Show the impact of Refactoring on JLOC

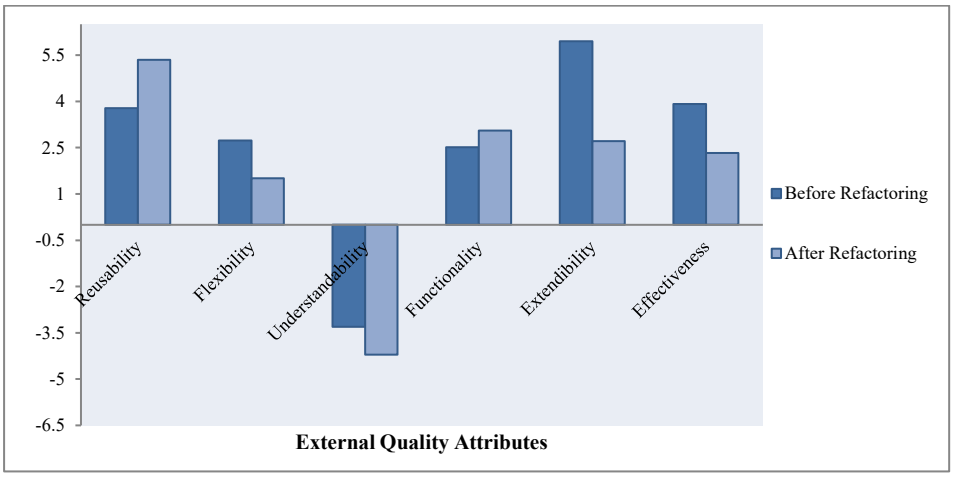

Fig. 3 Show the impact of Refactoring on JNFS

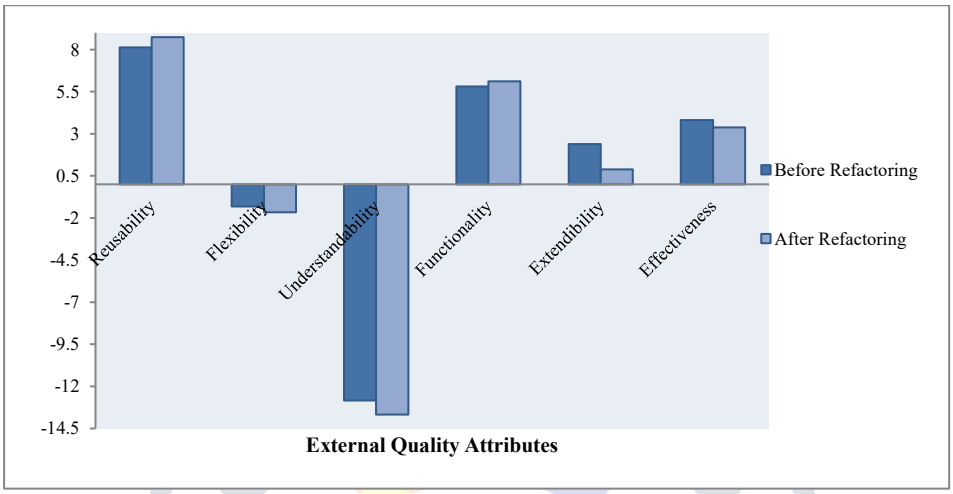

Fig. 4 Show the impact of Refactoring on JHotDraw

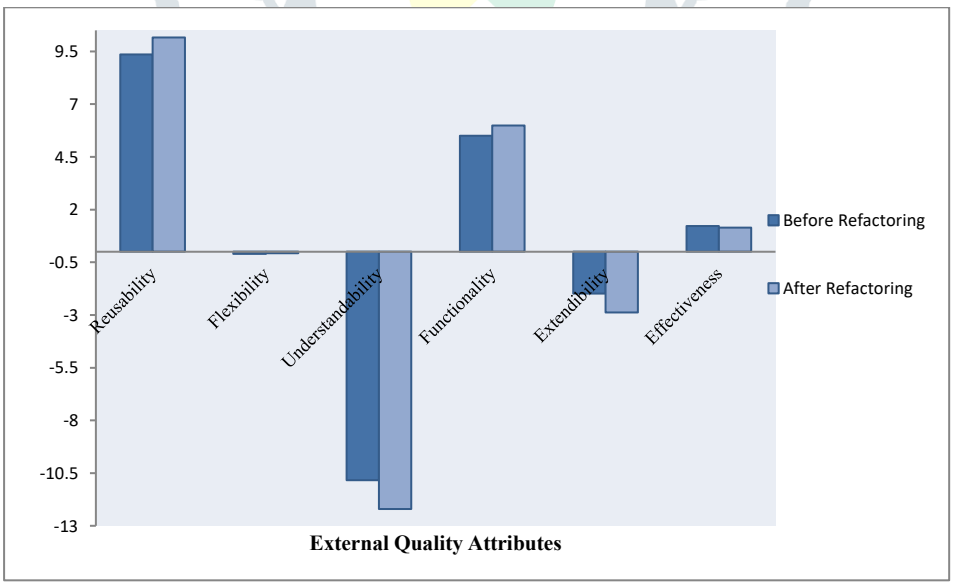

Fig. 5 Show the impact of Refactoring on JFreeChart 0.5

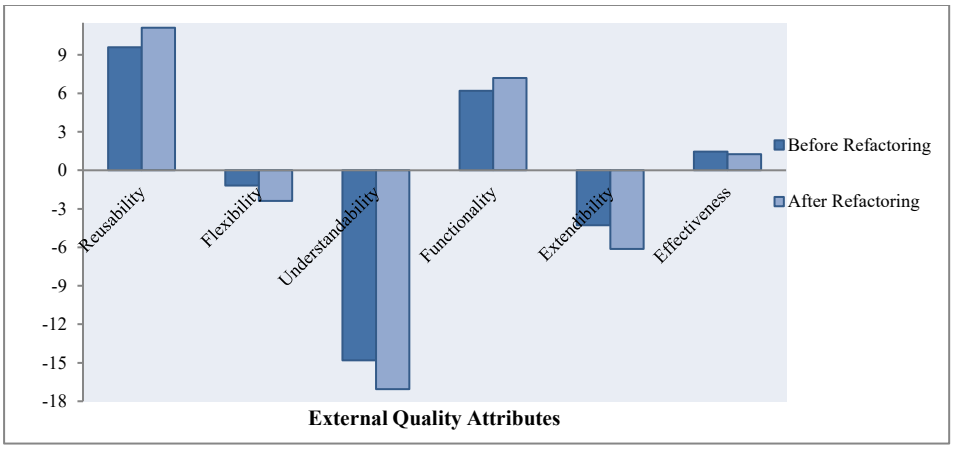

Fig. 6 Show the impact of Refactoring on JFreeChart 0.6

IV. RELATED WORK

Elish and Alshayeb [19] classified refactoring techniques based on software quality attributes. They used three student projects and three open-source software- written in Java and performed refactoring on these software only at the class level. After applying the refactoring, they measure the refactoring impact on internal quality metrics and then map the changes through internal quality attributes to the external quality attributes.

Alshayeb [20] investigated the refactoring effect on software quality using three different software systems, UML tool and two open-source software RabtPad and TerpPaint. They apply different refactoring techniques at the class level and analyze the changes in internal quality attributes and then map these changes to external quality attributes. They failed to prove that refactoring improves software quality.

Elish and Alshayeb [21] examined the refactoring effect on software testing efforts. They select five refactoring techniques: Encapsulate field, Extract method, Extract class, Hide method, and Consolidate conditional expression. To analyze the impact of selected refactoring techniques on testing effort they use internal quality metrics and then map the changes to testing effort using the correlation between quality attributes and testing effort.

Mens and Tourwe [22] provide a survey of software refactoring. They discussed the type of software artifacts that are being refactored, identified issues for building refactoring tools, and the effect of refactoring on software reengineering processes. They identified the need for processes, methods, and tools that perform refactoring in a more consistent, flexible, and scalable way.

Ilyas et.al [23] presented a comparative study of code smell detection tools. They studied different code smell detection tools by comparing their features and working scenarios. They also extracted some suggestions based on variations in the result of detection tools. They performed a comparative study using JDeodorant and InCode. The tools detection methodologies are discussed and variations in the result are noticed. They used two versions of open-source software and found God Class and Feature Envy bad smells using JDeodorant and InCode. They analyzed that available tools are not mature enough to detect bad smells and refactoring.

Chatzigeorgiou and Manakos [24] investigated the evolution of different bad smells in two object-oriented and open-source codes. The consecutive versions of JFreeChart and Jflex were used to analyze three bad smells namely state checking, long method, and feature envy. JDeodorant was used to detect these bad smells. The result revealed that bad smells increase as the software evolves and few bad smells are eliminated using refactoring.

Kannangara and Wijayanake [25] studied the impact on code quality. The objective of this study is to test the statement that refactoring improves software quality. To achieve this objective, the study was carried out using two measurements- internal and external measures. Refactoring techniques were evaluated using four external quality attributes(Resource utilization, Analyzability, Time behavior, and Changeability) and five internal quality metrics(Cyclomatic complexity, Maintainability index, Coupling, Lines of code, and Depth of inheritance tree). Refactoring techniques were applied to the source code of a small-scale project on an online document repository. The results of external quality and internal quality metrics did not show any positive effect of refactoring techniques on software quality except for the maintainability index. According to the result of internal and external quality factors, this study indicates that code quality is not improved after refactoring.

From the above papers, we can notice the limitations of the existing research work. The researchers in [19-21] investigate the effect of refactoring on external quality attributes by applying changes in the internal quality metrics [26]. They did not validate refactoring impact on quality attributes empirically. In addition [25] studied the impact of refactoring using only one small-scale project with different quality attributes. Our work analyzes the impact of refactoring on the quality of six different open-source software in terms of external and internal quality attributes. The values of these quality attributes are calculated and compared before and after applying the suitable refactoring techniques. The values of quality attributes are calculated using the formula given by Bansiya [9]. Also, the effect of refactoring on quality attributes of the software source code is analyzed in terms of nature(positive and negative impact) as well as amount(metric value).

V. CONCLUSION AND FUTURE SCOPE

Refactoring is a simple and important activity to improve the source code of the software. Refactoring makes code easy to use by reducing the complexity of the software. In this work, six different open-source software are used to analyze the impact of refactoring on software quality. Bad smells from these softwares are detected and removed by applying refactoring techniques using the JDeodorant plugin in Eclipse. External and internal quality attributes are calculated and compared to analyze the result of refactoring on software.

It is concluded from the experimental results that in open source software JFreeChart (0.5 and 0.6), as the size of the software increases, the number of bad smells also increases and the understandability of the software decreases. The McCabe cyclomatic complexity of the software is reduced in five software systems except JLOC which shows that refactoring has a positive impact on software in terms of complexity. Weighted methods per class complexity is reduced in three subject systems (RabtPad, JNFS, JFreeChart 0.6), after refactoring which shows positive as well as negative impact on software in terms of complexity.

We analyze the impact of multiple refactoring techniques together rather than a single refactoring technique on the software quality attributes. By applying refactoring to software, the reusability and functionality of all the software are improved in all open-source software systems, and other quality attributes like effectiveness, flexibility, and extendibility are decreased in all open-source software systems. Understandability is only increased in RabtPad and decreases in all other software, which means understandability is increased in 16.67% of open-source software. This result shows that groups of refactoring techniques may have negative effects on software quality.

In future work, more types of bad smells can be detected in open source software and different refactoring techniques can be applied to these bad smells. The impact of refactoring on different sets of (i) internal quality attributes such as MOOD metrics and (ii) external quality attributes like correctness, performance, testability, etc. can also be investigated.

REFERENCES

[1] Opdyke, W.: Refactoring: A Program Restructuring Aid in Designing Object-Oriented Application Frameworks. PhD thesis, University of Illinois at Urbana-Champaign. USA (1992)
[2] Fowler, M., Back, K., Brant, J., Opdyke, W., Roberts, D.: Refactoring: Improving the Design of Existing Code., Addison-Wesley. New York (1999)
[3] Roberts, D.B.: Practical Analysis for Refactoring. PhD thesis, University of Illinois at Urbana-Champaign. USA (1999)
[4] Rattan, D., Bhatia, R., Singh, M.: Software clone detection: A systematic review. Information and Software Technology. ELSEVIER. 55, 1165-1199(2013)
[5] ISO/IEC, 9126 Standard, Information technology-Software product evaluation-Quality characteristics and guidelines for their use, Switzerland, International Organization For Standardization, 1991. (1991)
[6] Sharma, A.K., Kalia, A., Singh, H.: An Analysis of Optimum Software Quality Factors. IOSR Journal of Engineering. 2, 663-669 (2012)
[7] Bansiya, J., Davis, C.G.: A Hierarchical Model for Object-Oriented Design Quality Assessment. IEEE Trans. Softw. Eng. 28, 4-17 (2002)
[8] Tsantalis, N., Chaikalis, T., Chatzigeorgiou, A.: JDeodorant-Identification and Removal of Type-Checking Bad Smells. 12th European Conference on Software Maintenance and Reengineering, 329–331 (2008)
[9] Tsantalis, N., Chatzigeorgiou, A.: Identification of move method refactoring opportunities. IEEE Transactions on Software Engineering. 35, 347–367 (2009)
[10] Eclipse. http://www.eclipse.org/ . Accessed: 16 Jan 2018
[11] JDeodorant. https://marketplace.eclipse.org/content/jdeodorant. Accessed: Jan 2018
[12] Eclipse Metrics Plugin. http://sourceforge.net/projects/metrics/. Accessed: Jan 2018
[13] JLOC beta. https://sourceforge.net/projects/jloc/. Accessed: Jan 2018
[14] JNFS 0.1. https://sourceforge.net/projects/jnfs/files/. Accessed: Jan 2018
[15] RabtPad. https://sourceforge.net/projects/rabtpad/files/. Accessed: Jan 2018
[16] JHotDraw5.2. https://sourceforge.net/projects/jhotdraw/files/JHotDraw/. Accessed: Jan 2018
[17] JfreeChart. https://sourceforge.net/projects/jfreechart/. Accessed: Jan 2018
[18] Michura, J., Capretz, M.A.M., Wang S.: Extension of Object-Oriented Metrics Suite for Software Maintenance. ISRN Software Engineering,2013, 1-14,(2013)
[19] Elish, K.O., Alshayeb, M.: A Classification of Refactoring Methods Based on Software Quality Attributes. Arabian Journal for Science and Engineering. Springer. 36, 1253-1267 (2011)
[20] Alshayeb, M.: Empirical Investigation of Refactoring Effect on Software Quality. Inform. Softw. Technol. J. 51, 1319-1326 (2009)
[21] Elish, K.O., Alshayeb, M.: Investigating the Effect of Refactoring on Software Testing Effort. 16th Asia-Pacific Software Engineering Conference, 29-34 (2009)
[22] Mens, T., Tourwe, T.: A survey of Software Refactoring. IEEE Trans. Softw. Eng. 30,126-139 (2004)
[23] Hamid, A., Ilyas, M., Hammayun, M., Nawaz, A.: A comparative Study on Code Smell Detection Tools. International Journal of Advanced Science and Technology. 60, 25-32 (2013)
[24] Chatzigeorgiou, A., Manakos, A.: Investigating the Evolution of Bad Smells in Object-Oriented Code. 2010 Seventh International Conference on the Quality of Information and Communications Technology, pp. 106-115 (2010)
[25] Kannangara, S.H., Wijayanaka, W.M.J.I: An Empirical Evaluation of Impact of Refactoring on Internal and External Measures of Code Quality. International Journal of Software Engineering and Applications. 6, 51-67 (2015)
[26] Dandashi, F.: A method for assessing the reusability of object-oriented code using a validated set of automated measurements. Proceedings of 2002 ACM Symposium on Applied Computing. 997-1003 (2002)