International Journal of Scientific Research & Engineering Trends
Volume 4, Issue 6, Nov-Dec-2018, ISSN (Online): 2395-566X

Analysing of Impact of Code Refactoring on Software Quality Attributes

Himanshi Vashisht
Dept. of Computer Science &Engg.
Haryana Engineering college
Jagadhri, Haryana , India

Associate Prof. Sanjay Bharadwaj
Dept. of Computer Science &Engg.
D.A.V College for Girls
Yamuna nagar, Haryana, India

Sushma Sharma
Dept. of Computer Science &Engg.
D.A.V College for Girls
Yamuna nagar, Haryana, India

***Abstract*-Code refactoring is a "Process of restructuring an existing source code.". It also helps in improving the internal structure of the code without really affecting its external behavior". It changes a source code in such a way that it does not alter the external behavior yet still it improves its internal structure. It is a way to clean up code that minimizes the chances of introducing bugs. Refactoring is a change made to the internal structure of a software component to make it easier to understand and cheaper to modify, without changing the observable behavior of that software component. Bad smells indicate that there is something wrong in the code that have to refactor. There are different tools that are available to identify and emove these bad smells. Software has two types of quality attributes- Internal and external. In this paper we will study the effect of clone refactoring on software quality attributes.**

***Keywords* – Code Cloning, Code Refactoring, External Quality attributes, Internal Quality attributes. etc.**

I.INTRODUCTION

Software Refactoring is also known as Software Restructuring because there are not many differences in these two terms and they are quite similar. Software refactoring is actually a super-set of software restructuring. It contains all the techniques that can be used in restructuring. It also adds some new specific techniques related to object oriented programming [1].

Fowler[2] describes 22 different bad smells in a code and instructions on how to remove them. This is called Refactoring. So, Software Refactoring is the technique of changing the software system in such a way that its external behavior does not change but its internal structure is improved. Only the internal structure of software is changed during Refactoring. This makes it understandable and less expensive to modify in future, without changing the external behavior. Refactoring the code because the existing code is hard to understand or the addition is not easy to make because of the existing design. It produces code that everyone understands.

1. Steps in Software Refactoring

Roberts said, "Refactoring is a program transformation process which contain pre-condition and post-condition that must be satisfied for the refactoring to be easily applied".
Refactoring has at least two steps- analysis and execution.
1.1 Analysis- First step in software refactoring is analysis. In this step the program is analyzed to determine whether the desired pre-conditions are satisfied or not. It means to detect whether bad smell occur in the software or not.

1.2 Execution- If first step satisfy the precondition then second step is execution. The actual transformation of the source code takes place in this step. Execution is removal of these bad smells that occured in first step by applying refactoring techniques.

II. LITERATURE REVIEW

Fowler et al. [1] described the 22 bad smells and their 72 respective techniques to refactor bad smells. They also associated refactoring rules with these bad smells, suggesting how to resolve these bad smells. They declared duplicate code as a serious kind of bad smell. It increased maintenance cost of software. Due to increasing use of open source software and its variants, there is also increased use of code reuse. Due to code reuse, it results in duplication of code.

The International Organization for Standardization (ISO/IEC9126) et al. [2] published a set of international standards related to the software engineering, such as ISO 12207 and ISO 9126. However, there is a set of cross-references between the two standards. The ISO 9126 on software product quality and ISO 12207 on software life cycle processes had been analyzed to investigate the relationships between them and to make a mapping from the ISO 9126 quality characteristics to the ISO 12207 activities and vers versa. This study presented a set of comments and suggestions to improve the ISO 9126. The weaknesses of the cross references between the two ISO standards had been highlighted. In addition, this study provided a number

of comments and suggestions to be taken into acc**1.** on the next version of the
ISO 9126 international standard.

Kamiya et al. [3] proposed a clone detection CCFinder (Code Clone Finder).This tool incorpor the use of a lexical analyzer which removes white spaces, comments from source code generate token sequence of code, Then after, token sequence is transformed using certain rules. This transformation regularizes the identifiers by partially removing the context information. A special token replaces the identifiers so that code portions with different variable names could be returned as clone pairs by the matching algorithm.

Garg and Tekchandani [4] introduce an approach to re factor the clones on the basis of their essentiality. The approach measures the maintenance overhead in terms of repetitiveness, size of clones and complexity. They find clones using CC Finder clone detection tool. After detection of clones, calculate efforts required in maintaining clones. They arrange clones according to their value of maintenance overhead. The clones which having high value should be re factor first.

Tstanalis et al. [5] propose an approach to check the re factor ability of clones. They defined pre-condition which are checked during re factor ability. If these pre-condition are satisfied, then we can remove clones easily. If these are violated, then refactorability of that clone is not possible. They used four clone detector tools- CCFinder, Deckard, CloneDR, Nicad.They found that clone with a close distance tends to be more refactorable than more distant. Type 1 clones are more refactorable than other types of clones.

Bansiya and Davis et al. [4] presented a QMOOD (Quality Model for Object Oriented Designed) that access quality attributes like reusability, functionality, extendibility, flexibility, understand ability, effectiveness. QMOOD relates low level design properties such as encapsulation, coupling and cohesion to high level quality attributes. They weighted quality attributes accordance to their influence and importance in the system.

Fontana et al. [7] investigates the impact of clone refactoring on quality attributes internal quality attributes like complexity, coupling and cohesion. They used three clone detection tools PMD, Bahumas and Code Pro on two open source software– Ant and Ghantt Project. Intellij IDEA tool is used for refactoring. They analyze that, after refactoring there is improvement in cohesion, decrement in coupling, complexity and lines of code.

Alshayed et al. [8] investigates the effect of refactoring on software quality attributes. He focused on quality attributes like adaptability, maintainability, reusability, understand ability and testability. They apply refactoring on three open source software- terp Paint, UML tool and Rabtpad. But after refactoring, he concludes that it does not necessary that after refactoring there is increase in quality of software.

III. CLONE REFACTORING TECHNIQUES

Refactoring Techniques remove the bad smells from software and make it clean. Refactoring is a set of techniques, procedures and steps to keep software as clean as possible. There are some basic techniques for duplicated code proposed by Fowler [2]:

1. **Extract Method-** When the clone fragments are located in the methods that belong to the same class, extract method is applied. In this scenario, the unified code is extracted in a new private method within the same class.
2. **Extract and Pull Up Method-** When the clone fragments are located in the methods that belong to different subclasses of the same super class, Extract and Pull Up Method is applied. The unified code is placed in a new protected method within the super class.
3. **Introduce Template Method-** Introduce Template Method is a special case of the previous refactoring. If the methods being commonly accessed in the clone fragments do not belong to the before mentioned clone types, but have an identical signature and the same return type, then an abstract method is created in the super class and the unified code is pulled up.
4. **Introduce Utility Method-** Introduce Utility Method is applied when the clone fragments are located in methods of unrelated classes (not being part of the same inheritance hierarchy), and the fragments do not access any instance variables or methods. Then, the unified code can be extracted into a static method placed in a utility class.

IV. SOFTWARE QUALITY ATTRIBUTES

Software quality is to defining a set of software quality attributes for the system. The ISO/EIC 9126 standards [3] define software quality characteristics as “A set of software system attributes by which its quality is evaluated and described ".
The attributes that affect the software quality is classified into two groups- internal quality attributes and external quality attributes.

1. **Internal quality attributes-** These are that factor which can be measured directly e.g. lines of code and

number of classes etc. The value of software internal quality attributes is predicted using software metrics.

2. **External quality attributes-** are that factors which can be measured indirectly e.g. maintainability and understand ability etc. The external quality attributes are dependent on the software metrics.

V. INTERNAL QUALITY ATTRIBUTES

In this paper, software metrics are used to predict the value of internal quality attributes to find the impact of refactoring on software quality. Internal Quality attributes are calculated by Eclipse Metrics Plugin. These are that factor which can be measured directly e.g. lines of code and number of classes etc. The value of software internal quality attributes is predicted using software metrics.

They interpret these values to calculate metrics used by Bansiya [4]. Therefore the software metrics (OO Metrics) are -

1. **Weighted Methods per Class (WMC) -** WMC metric is used to measure the complexity of a class by calculating the sum of complexity of the methods of a class. It is used to count the methods implemented within a class. The number of methods and complexities involved as predictors, how many time and effort is required to develop and maintain the class.

$$\boldsymbol{WMC = \sum_{i=1}^{m} Ci}$$

Where, Ci= complexity of method i in a class,
m= number of methods.

2. **Hierarchies-** Hierarchies defined as the length of the longest path from a given class to the root class in the inheritance hierarchy. Hierarchies are used to represent different generalization-specialization concepts in a design. It is count of of the number of non-inherited classes that have children in design.

$$\text{Hierarchies} = \text{DIT}$$

Where, DIT = Depth of inheritance tree.

3. **Messaging -** Messaging count the number of methods implemented in a given class. It is count of the number of public methods that are available as services to another class. Measure of services that a class provides.

$$\text{Messaging} = \sum_{i=1}^{n} \text{NOM}$$

where, NOM = the total number of public methods in a class,
n = number of classes.

4. **Design size** – Design size count the total number of classes in software.

$$\text{DS} = \sum_{i=1}^{p} NOC$$

Where, NOC = Total number of classes in a package,
p = number of packages.

5. **Composition -** The count of the number of data declarations whose types are user defined classes. Measures the "part-of ", "has", "consist-of", which are aggregation relationships in an object oriented design.

$$\text{Composition} = \sum_{i=1}^{n} \text{NOA}$$

Where, NOA = Total number of Attributes in a class,
n = number of classes.

6. **Coupling between Objects (CBO) -** A class is coupled to another if it uses methods or attributes of the coupled class. It count the number of other classes to which it is coupled or directly related. It defines the interdependency of an object on other object in a design.

$$\text{CBO} = \text{Ce}$$

Where, Ce = Efferent Coupling.

7. **Lack of Cohesion in Methods (LCOM)-** LCOM measures the number of pairs of methods in the class that have no attributes in common. Cohesion means functional strength of a class Or no. of methods in a class that refers a instance variable. Cohesion is a degree of methods through which all the methods of the class are inter-related with one another. If method refer more variable then it is more complex and less cohesive.

8. **Encapsulation -** is the process of hiding all the details of an object that do not contribute to its essential characteristics. Encapsulation is the technique of making the fields in a class private and providing access to the fields via public methods. If a field is declared private, it cannot be accessed by anyone outside the class, thereby hiding the fields within the class. For this reason, encapsulation is also referred to as data hiding. The ratio of the number of private/protected entities to the total number of entities declared in the class.

$$\text{Enc} = \frac{\text{a (P)}}{\text{a}}$$

Where, a (P) = number of private attributes in a class,
a = total number of attributes in a class.

9. **Inheritance -** The ratio of the number of entities inherited by a class to the total number of entities accessibly by member methods of the class. A measure of "is-a" relationship between classes. Relationship is related to the level of nesting of classes in an inheritance hierarchy.

$$\text{Inheritance} = \left(\sum_{i=1}^{n} \frac{\text{NORM}}{\text{NOM}}\right) \times 100$$

Where, NORM= number of overridden method in class,
NOM = total number of methods in a class,
n = number of classes.

10. **Polymorphism** -Polymorphism count of the methods that exhibit polymorphic behaviour. It is the ability to substitute objects whose interfaces match for one another at run-time. It is a measure of services that are dynamically determined at run-time in an object.

$$\text{Poly} = \sum_{i=1}^{n} \text{NORM}$$

Where, NORM = number of overridden methods in a class,
n = number of classes.

11. **Abstraction** - Abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user. In other words user will have the information on what the object does instead of how it does it.

$$\text{Abs} = \frac{\sum_{i=1}^{n} \text{NoI}}{n}$$

Where, NoI = total number of interfaces in a package,
n= total number of classes in a package.

VI. EXTERNAL QUALITY ATTRIBUTES

The external quality attributes are dependent on the internal quality attributes. Therefore, attributes can be calculated by using these formulas given by Bansiya and Davis [5]. External quality attributes (ISO 9126 (QMOOD)) are [13] are reusability, flexibility; understand ability, functionality, extendibility and effectiveness.

These are that factors which can be measured indirectly e.g. maintainability and understand ability etc. The external quality attributes are dependent on the software metrics.

Table 1 Formula to Calculate External Quality Attributes

External Quality Attributes	Formulas Used for Calculation
Reusability	-0.25*Coupling+0.25*Cohesion+0.5*Messaging+0.5*Design Size.
Flexibility	0.25*Encapsulation - 0.25*Coupling + 0.5*Composition + 0.5* Polymorphism.
Understand ability	-0.33*Abstraction+0.33*Encapsulation-0.33*Coupling+0.33*Cohesion-0.33*Polymorphism-0.33*Complexity-0.33*Design Size.
Functionality	0.12*Cohesion + 0.22*Polymorphism + 0.22*Messaging + 0.22*Design Size +0.22*Hierarchies.
Extendibility	0.5*Abstraction - 0.5*Coupling + 0.5*Inheritance +0.5* Polymorphism.
Effectiveness	0.2*Abstraction + 0.2*Encapsulation + 0.2*Composition+ 0.2* Inheritance+ 0.2*Polymorphism.

VII. CONCLUSIONS AND FUTURE WORK

Refactoring is an important and simple activity to re factor the source code of the software's. Refactoring makes code easy to use, by reducing the complexity of the software. In this work, four different software's are used to analyze the impact of clones' refactoring on quality of softwares. Clones are detected from the software's by using CCFinder tool and these clones are removed by applying refactoring techniques using JDeodorant plugin in eclipse. After that quality attributes are calculated and compared to analyze the result of refactoring on software's. In future work, we shall see the impact of software refactoring on external quality attributes of open source software's in detail.

REFERENCE

[1] M. Fowler, K. Back, J. Brant, W. Opdyke and D.B. Roberts, "Refactoring: improving the design of existing code," Addison-Wesley, New York, 1992.

[2] ISO/IEC9126, 1991. "Software Product Evaluation-Quality Characteristics and Guidelines for their Use", 9126 Standard, Information technology.

[3] T. Kamiya, S. Kusumotoand K. Inoue, "CCFinder: a multilinguistic token based code clone detection system for large scale source code," IEEE Transactions on Software Engineering, Vol. 28, No. 7, pp. 654-670, 2002.

[4] R. Gargand and R. Tekchandani, "Enhancing code clone management by prioritizing code clones," Master's Thesis, Thapar University, Patiala, 2014.

[5] N. Tsantalis, M. Mazinanian and G.P. Krishnan, "Assessing the refactorability of software clones, " IEEE Transactions on Software Engineering, Vol. 41, No. 11, 2016.

[6] J. Bansiya and C.G. Davis, "A hierarchical model for object-oriented design quality assessment", IEEE Transactions on Software Engineering, Vol. 28, No. 1, pp. 4-17, 2002.

[7] F.A. Fontana, M. Zanoni, A. Ranchetti and D. Ranchetti, "Software Clone Detection and Refactoring," ISRN Software Engineering, 2013.

[8] M. Alshayeb, "Empirical investigation of refactoring effect on software quality, " Information and Software Technology, ELSEVIER, 2009.

[9] JDeodorant, URL - Retrieved from https://marketplace.eclipse.org/content/jdeodorant.

[10] JChart2D Retrieved from https://sourceforge.net/projects/jchart2d/.

[11] Apache-ant Retrieved from http://ant.apache.org/.

[12] JMeter Retrieved from http://jmeter.apache.org.

[13] JEdit Retrieved from http://www.jedit.org/.

[14] Metrics Plugin, URL – Retrieved from http://sourceforge.net/projects/metrics/.