IJITKM 0973-4414 Volume-9, Number-1 Jun-Dec 2015 pp. 87-94 Impact Factor 2.8 available online at www.csjournalss.com

Refactoring: Risks and its Effects on Software Quality Attribute

Ramesh Kumar, Dr.Rajesh Verma
Research Scholar, Department of Computer Science, Singhania University, Rajasthan.
Asst. Professor, Department of Computer Science, Govt. College, Indri, Karnal

Abstract: Refactorings are used to improve the internal structure of software without changing its external behavior. Designed poorly software systems are difficult to understand. Refactoring technique has been applied to improve the quality of software and productivity of developer. Refactoring is mainly applied to improve the software quality after some features are added. Software development team and user of software always require quality software. This paper presents a study of refactoring risks and its effects on software quality attribute. The extracted information from this paper should help the researchers to prepare superior research topic and can save the researchers effort and time.

Keywords: Refactoring, risks, improve, software quality, attribute, productivity.

1. Introduction

Software refactoring was cast by William F. Opdyke in his Ph.D. dissertation [1], after the publication of book which is written by Martin Flower Refactoring: improve the design of existing code, it is mostly used. Refactoring preserves all of system actual functionality to modifying the structure of a program. It is believed that refactoring improves developer productivity and software quality [2]. The purpose of refactoring is to modify a program structure into a good quality after fixing quality fault [3]. Such types of modifications decrease the cost and effort of software maintainability for the long time by keeping software difficulties within reasonable levels. In the past researcher favor several techniques to solve the difficulties and software refactoring is which one. In the context of software evolution it is used to improve the quality of the software. Extracting some code into a method, renaming a class, changing the method signature are ways of refactoring.

The word smell in the software program means prospective problem in the program code. Refactoring methods are applied when the smell is found. It continues till we find the maximum effective code [4]. In any programming language refactoring can be applied, but the maximum of refactoring current tools have been developed for the Java language. Ratzinger et at. found that, the number of faults decreases, if we use the number of refactoring grow increases in the time period of preceding [5]. Diehl and WeiBgerber found that a high part of refactoring edits is often followed by an increasing ratio of fault reports [6, 7]. Refactoring is used to those programs which are not coded poorly. In this paper we study in section as follow: Section 2: explains the related work of various researchers on software refactoring. Section 3. Refactoring risk. Section 4: describes the methods of refactoring. Section 5: Quality model for object oriented design and software measure. Section 6: analyses of refactoring 7: Conclusion.

2. Related Work

The purpose of this paper is to find the risks of refactoring and its effects on software quality. Now we study the review of various researchers on software refactoring. Katoka assesses the effect of refactoring on the maintainability. He used coupling metrics to measure the maintainability. He prefers a quantitative estimation method. Mohammed Alshayeb assesses the effect of refactoring on the software quality external attribute. The quality attribute taken were Maintainability, Reusability, Understandability. In the relationship of refactoring method and external quality attribute author found the inconsistent trend. Karim O. Elish [8] proposed a classification of refactoring methods based on their measurable effects on

software quality attributes. Kolb et al. found that refactoring improves maintainability and reusability of software and case study on the implementation and design [9]. Moser et al. found that refactoring increase quality related measure by conducted a case study in an industrial, agile environment [10]. Flower has given various types of refactoring methods. These methods linked with software quality attribute.

3. Refactoring Risk: - Refactoring is frequently hard because the refactorer often is not the same person as the original designer and they do not have the same background. It is true that when a new team member who is not fully experienced with this system, decides to inject new ideas into an otherwise stable system. He may begin forcing the project in the direction unintended by the whole team when the team is new and is not given sufficient guidance. It is just a risk, however, there is also a possibility that the team is wrong and the new team member, if put a charge and was allowed to do his thing would actually make a serious enhancement. These problem occur between a team when they working on legacy system. Often there are no world-recast improvements planned, so the team is stable with their design. Their purpose is to block new bugs from being introduced and fix old ones with a couple extra features thrown in. Member of new team might come along and distress the apple cart by insisting that, he rewrites certain subsystem program. New bugs are created and users of a justly safe product are distract because the software, from their outlook is getting defeat. If you have larger functionality changes in the software however, have a user base that expects your product to not be fully backed quit yet then it's a much better condition to consider major refactoring because the long time benefits of the superior design will be effective and you're less likely to disrupt a large user-base.

4. Methods of Refactoring:

In Flower's catalog various software refactoring methods are defined. Here we use the some of the methods from this catalog.

1. **Pull Up Method**
   Pull Up refactoring method involves moving a member of a class, such as procedure, from a Subclass into a Super class.

```
 public abstract class Vehicle
{
 // methods
 }
 public class Bus : Vehicle
{
  public void Turn(Direction direction)
  {
     //write code here
  }
}
 public class Bike : Vehicle
{
}
 public enum Direction
{
    Right,
    Left,
}
```

Turn method is currently only available to the Bus class, we also want to use it in the other Bike class so we create a base class. Only place methods that need to be used by more than one derived class. After refactoring our code is:

```
public abstract class Vehicle
{
  public void Turn(Direction direction)
  {
     //write code here
    }
}
public class Bus : Vehicle
{
 }
public class Bike : Vehicle
{
}
public enum Direction
{
  Right,
  Left,
}
```

2. **Add Parameter:** it is a refactoring operation that needs more information from its caller. This method create a new parameter to pass the necessary data.

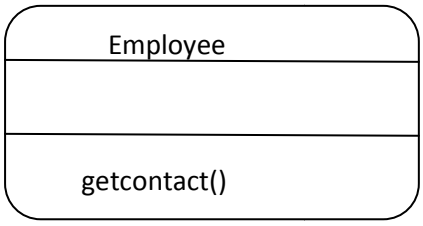

Fig. 1

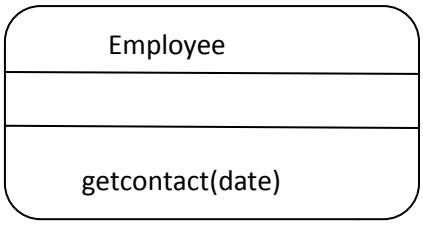

Fig. 2

Add a parameter for an object that can pass on this information.

3. **Extract Parameter**: This refactoring method allows selecting a set of parameters to a method. When the number of parameter in a method is too large then need refactoring. Process of refactoring is also done by delegate via overloading method.
4. **Replace Inheritance with Delegation:** A subclass uses only part of a super classes interface. This method allows removing a class from inheritance hierarchy. Through the new inner class selected methods of the parent class are invoked
5. **Rename Method:** When method name does not reveal its purpose then we use rename method refactoring. It changes the name of the method.

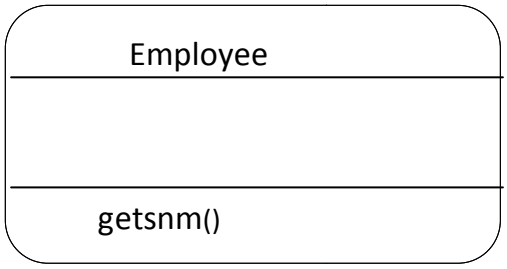

Fig- 3

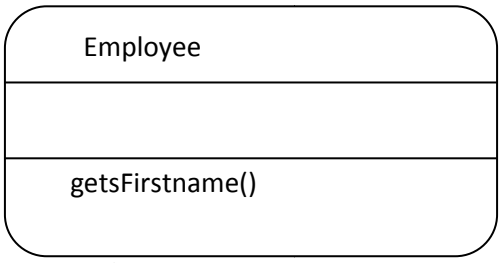

Fig-4

If method does not explain its purpose then we apply rename method. This method changes the name of function so it reveals its purpose. The function getnm( ) not display its purpose but function getFirstname ( ) get reveal it purpose enter employee first name. So refactoring rename method is used to rename the name of function.

6. **Form template method:** This refactoring method is producing standard piece of code by extracting different piece into methods with the same signature. In illustration, the compare and swap operations of collection-like objects. In a subclass common block of code can be pulled up. The client code won't have to change because this method is based on inheritance. Add new kinds of the algorithm beginning from the Template Method will be easily adding a new subclass.

7. **Replace method with method object:** This method isolating a long method in its own class allows stopping a method from swell out in size. This also allows splitting it into sub methods within the class, without infect the native class with usefulness methods.
8. **Introduce foreign method:** This method removes the code delicacy. If our code is repeated in several places, we can replace these code bit with a method call. Smooth examine that the foreign method is discover in a suboptimal place it is better than duplication.

5. The Quality Model for Object-Oriented Design

To achieve the goal of our research we need a quality model that can be easily used to evaluate system evolution when refactoring. To find the effect of refactoring activities on software quality, the model measure the feature of internal and external quality. This model and can be used to define metrics and the relationships between the both internal and external quality features. This model can be used the system and component levels and it is easy to assess, because it provides a quantitative assessment of external quality factors from the measurements of the internal design properties. Quality model gives developers an opportunity to fix problems, eliminate unwanted complexity early in the development cycle [13]. We use the following quality attribute in our study.

1. Effectiveness: Software should be effective.
2. Reusability: Software reliability is a broad term and it is easy to get confused about it.
3. Understandability: Software may be understandable
4. Extendibility: It is the capability of software to add extra functionality and it is a subset of flexibility. It allows required changing at the proper locations to be made without undesirable side effects.
5. Functionality: It is observance of software with actual requirement and identification. Software may have good functionality.
6. Flexibility: It is the capability of software to change, add, and remove functionality with changing current system.. The flexibility of the system may be superb and could be adjust to meet our processes of business.

Table 1. Quality Index Calculation

Software Quality Attribute	Computation formula Equation
Effectiveness	0.2 * Abstraction + 0.2 * Encapsulation + 0.2 * Composition + 0.2 * Inheritance + 0.2 * Polymorphism
Reusability	–0.25 * Coupling + 0.25 * Cohesion + 0.5 * Messaging + 0.5 * Design Size
Understandability	-0.33* Abstraction +0.33* Encapsulation-0.33* Coupling + 0.33* Cohesion -0.33* Polymorphism -0.33* Complexity -0.33* Design Size
Extensibility	0.5 * Abstraction – 0.5 * Coupling + 0.5 * Inheritance + 0.5 * Polymorphism
Functionality	0.12* Cohesion + 0.22* Polymorphism + 0.22* Messaging + 0.22* Design Size + 0.22* Hierarchies
Flexibility	0.25 * Encapsulation – 0.25 * Coupling + 0.5 * Composition + 0.5 * Polymorphism

Bansiya and Davis conducted an empirical study to determine the weights and the model was validated on the evolution of two real systems. [13].

Refactoring activities change the internal design of a software system; therefore we expect refactoring to impact external quality factors consequently.

5.1. Software Measures

Now we describe how we evaluate the effect of refactoring activities. Refactoring method changes the structure of software. To collect the following measures we use a quality assurance tool.

1. Average Number of Ancestors (ANA)-Abstraction of a system is measure by ANA. ANA is calculated by controlling the average of the depth of the inheritance tree. Depth of the inheritance tree is the length of the inheritance series from the root of the inheritance tree to the measured class.
2. The Design Size in Classes measures the number of classes in a system. DSC is very simple measure and on this measure the effect of refactoring is simple to assess. Refactoring affect this measures such as Replace Method with Method Object. Inline class decreases this measure but Replace Method with Method Object refactoring increase the number of classes in a design.
3. Direct class coupling (DCC) –it counts the number of classes that a class is directly connected to. Couplings include classes that are related by attribute declarations and message passing. When the system is highly coupled then the large value of Direct Class Coupling used. Refactoring high coupled classes can improve quality. This refactoring decreases coupling in the system.
4. Measure of aggregation (MOA) in the system is a count of the number of data declarations that are user defined classes. Measure of aggregation measures the extent of the part-whole relationships (composition), realized by using attributes. Objects can encapsulate data attributes as well as other objects. The refactoring activity, Replace Array with Object, replaces data stored in an array into an object. This refactoring introduces a part-whole relationship and therefore increases the use of composition.
5. Measure of functional abstraction (MFA) is the ratio of the number of inherited methods by a class to the total number of local methods in the class. Utilization of inheritance in a design measures by MFA. The number of inherited methods increases when inheritance is used effectively. This refactoring increases the functional abstraction in a design. To increase the reusability via inheritance refactoring can be used. A sign of a functional abstraction is expanding the number of inherited methods[14]. For illustration, the Form Template Method uses inheritance to pull-up the identical methods into a superclass.

6. Class interface size (CIS) is a count of public methods in a class, which is the size of the response set for the class. A class that has large number of responsibilities has many interactions with other classes.

6. Analysis of Refactroring

We face with many problems when we refactor any software. Sometimes it decreases the software quality. With refactoring the processing of software may be slow. Now we analyze the effect of software refactorings on measurements of the internal design properties, and then we assess the effect of refactoring on the quality factors that are defined by QMOOD.

Now we evaluate the effect of the refactoring mechanics on design properties and assess the changes in the relevant measures. For represent to increase we use +, - to represent a decrease and 0 to show no changes on measures.

Table 2. Effect of refactoring method on measure of software quality

Refactoring	ANA	DSC	DCC	MOA	MFA	CIS
Pull Up Method	0	0	-	0	-	-
Add Parameter	-	0	+	-	-	+
Extract Parameter	0	0	-	0	+	0
Replace Inheritance with delegation	-	0	+	+	-	+
Rename Method	0	0	0	0	0	0
From Template Method	0	0	0	0	+	+
Replace Method with Method Object	0	+	+	+	0	0
Introduce Foreign Method	0	0	+	0	0	+

From the results of refactorings analysis, we can notice that some measures are impacted more than others. To characterize the effect of refactoring on software measures, for each category, we count the number of times a measure is impacted by the refactorings in that category. If a measure is impacted more than 50% of the times then it is considered highly correlated with such a group, otherwise it is loosely correlated

6.1. Refactoring impact analysis on quality Attribute

Now we evaluate the effect of refactorings on quality attribute. Evaluation tells us that refactoring activity improves or deteriorates quality attribute. Evaluations that are presented in the following table result from using the impact of refactoring on software measures to calculate the impact on quality attribute. Table 3. consists of refactoring activities that are considered safe and unsafe. Pull Up Method, Add Parameter, Extract Parameter and Introduce Foreign Method are unsafe refactoring and rest of all are safe refactoring. Safe refactorings have improvements more than deteriorations but unsafe have more deterioration than improvement. In the table we show that Pull Up Method totally deteriorated the attribute not any improvement but From Template Method totally improvement the attribute. Replace Method with Method Object improves Effectiveness, Reusability, Understandability, Functionality and Flexibility. When we use safe refactoring then there is no more precautions need. In the case of Pull Up Method we need more precaution. Software developers can use these refactoring heuristics to make decisions when to use refactoring efficiently and to conduct goal-driven refactoring. If we want to improve the functionality of a system then we consider all refactoring activity which increase the functionality of system and avoid which deteriorate functionality. In summary, our research findings

revealed more useful information on refactoring effect on software quality attribute. In addition, the results in Table 3 show the effect of safe and unsafe refactoring on software quality attribute.

Table 3. Effects of safe and unsafe refactoring on software quality attribute

Refactoring	Effectiveness	Reusability	Understandability	Extensibility	Functionality	Flexibility	Deteriorated	Improvements
Pull Up Method	-	-	-	-	-	-	6	0
Add Parameter	-	-	+	-	+	-	4	2
Extract Parameter	-	-	-	-	+	+	4	2
Rename Method	+	-	+	-	-	+	3	3
Replace inheritance with Delegation	-	+	-	-	-	-	5	1
From Template Method	+	+	+	+	+	+	0	6
Replace Method With Method Object	+	+	+	-	+	+	1	5
Introduce Foreign Method	-	+	-	-	-	-	5	1

We show in the Table 4 summary of the relationships between refactorings and the six quality attribute and show refactoring heuristics that deteriorate, keep unchanged, and improve quality attribute. The table shows that refactoring activity has more positive impact on reusability. We show in the table that flexibility improved 53.5%, Understandability improved 47%, extendibility and effectiveness improved 42%and Functionality improved 40% of the refactoring. Effectiveness deteriorated only 9%. Applying these refactoring does not always deteriorate quality sometime be may improve the quality.

Table 4: Conclusion Effect of Refactoring on Software Quality

Quality attribute	Deteriorated	Unchanged	Improved
Reusability	16%	19%	65%
Flexibility	21%	25.5%	53.5%
Understandability	19%	28%	47%
Extendibility	30%	28%	42%
Effectiveness	9%	49%	42%
Functionality	25%	35%	40%

7. Conclusion

Refactoring is highly sensible to assure the quality of the software process and product in software development. Refactorings is used to improve the software quality attribute. In this study we analyses the software risks and effects of refactoring on attribute of software quality attribute. There are many risks when we refactor any software. But if we use safe refactoring then we save the quality of software. We analyses the effect of many refactoring activities on six software quality attribute. In this paper we also define of eight methods of refactoring and six software measure notice that they effect on different

software quality attributes. We found that some refactorings are safe and some refactorings are unsafe. Safe refactoring improves the attribute and unsafe refactorings are not improving the attribute they only deteriorate the attribute. This paper concludes that refactoring improves the quality of software but developer needs to look for the particular refactoring method for desirable attribute.
Software Refactoring is an important area of research that promises substantial benefits to software maintenance. Refactoring allows developers to redress code without starting again. We can return with a well structured and well designed code after proper application of refactoring techniques.

References

[1] W. F. Opdyke, "Refactoring: A Program Restructuring Aid in Designing Object-Oriented Application Frameworks", Ph. D. Thesis, Univ. of Illinois at Urbana-Champaign, (1992).

[2] M. Fowler. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional, 2000

[3]. J. Rech, Context-sensitive Diagnosis of Quality Defects in Object-Oriented Software Systems, Ph. D. Thesis. Hildesheim: University of Hildesheim, Department IV, 2009

[4] W.C Wake, (2003), "Refactoring Workbook", Addison Wesley.

[5] J. Ratzinger, T. Sigmund, and H. C. Gall. On the relation of refactorings and software defect prediction. In MSR '08: Proceedings of the 2008 international working conference on Mining software repositories, pages 35–38, New York, NY, USA, 2008. ACM.

[6] P. Weißgerber and S. Diehl. Are refactorings less error-prone than other changes? In MSR '06: Proceedings of the international workshop on Mining software repositories, pages 112–118. ACM, 2006.

[7] P. Weißgerber and S. Diehl. Identifying refactorings from source-code changes. In ASE '06: Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering, pages 231–240, Washington, DC, USA, 2006. IEEE Computer Society.

[8] K. Elish, M. Alshayeb and O. Karim, "A Classification of Refactoring Methods Based on Software Quality Attributes", Arab J Sci Eng., vol. 36, (2010) May, pp. 1253-1267.

[9] P. Meananeatra, "Identifying Refactoring Sequences for Improving Software Maintainability", ACM 978-1-4503, (2012) September.

[10] R. Moser, A. Sillitti, P. Abrahamsson, and G. Succi. Does refactoring improve reusability? In ICSR, pages 287–297, 2006.

[11] AspectJ homepage: http://www.aspectj.org

[12] Subramanian, Nary, and Lawrence Chung,(2001) "Metrics for software adptability", Proc. Software Quality Management(SQM 2001).

[13] J. Bansiya, C. Davis, ―A Hierarchical Model for Object-Oriented Design Quality Assessment□, IEEE Transactions on Software Engineering, 28 (1), (2002) pp. 4–17

[14]. T. Mayer, T. Hall, ―A Critical Analysis of Current OO Design Metrics□, Software Quality Journal, 8 (2), (1999) pp 97–110.

[15] S. Chidamber, C. Kemerer, ―A Metrics Suite for Object Oriented Design□, IEEE Transactions on Software Engineering, 20(6), (1994) pp. 476–493