Software design enhancement using refactoring

A Dissertation

Submitted

By

Rekha Rani

11306293

To

Department of Computer Science and Engineering

In partial fulfilment of the Requirement for the

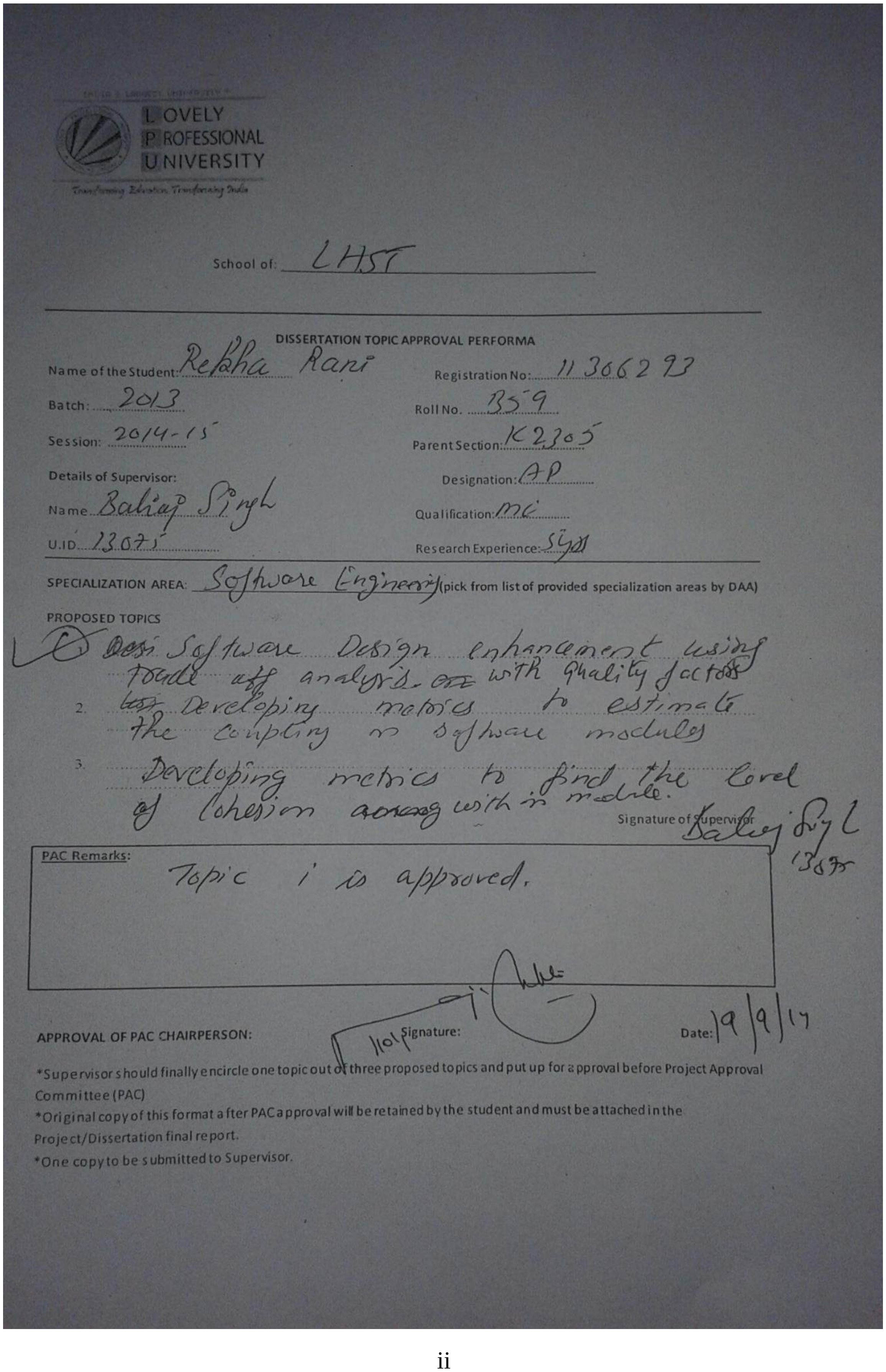
Award of the Degree of

Master of Technology in Computer Science and Engineering

Under the guidance of

Mr. Balraj Singh

(May 2015)



LOVELY PROFESSIONAL UNIVERSITY

Transforming Education Transforming India

School of: LHST

DISSERTATION TOPIC APPROVAL PERFORMA

Name of the Student: Rekha Rani	Registration No: 11306293
Batch: 2013	Roll No. B59
Session: 2014-15	Parent Section: K2305
Details of Supervisor:	Designation: AP
Name: Baljeet Singh	Qualification: MC
U.ID. 13075	Research Experience: 5yrs

SPECIALIZATION AREA: Software Engineering (pick from list of provided specialization areas by DAA)

PROPOSED TOPICS

[✓ 1.] ~~Desi~~ Software Design enhancement using trade off analysis ~~on~~ with quality factors
2. ~~for~~ Developing metrics to estimate the coupling in software modules
3. Developing metrics to find the level of cohesion ~~among~~ with in module.

Signature of Supervisor: [illegible] 13075

PAC Remarks:

Topic 1 is approved.

APPROVAL OF PAC CHAIRPERSON: Signature: [illegible] Date: 19/9/14

*Supervisor should finally encircle one topic out of three proposed topics and put up for approval before Project Approval Committee (PAC)

*Original copy of this format after PAC approval will be retained by the student and must be attached in the Project/Dissertation final report.

*One copy to be submitted to Supervisor.

ABSTRACT

Refactoring is the process of improving the design of code by changing its internal structure without affecting its external behavior. Refactoring method is useful for easily readability, reduce bugs and for improving software quality attributes. There are many refactoring methods that are apply on internal attributes such as OOCK metrics, each method has a particular purpose and effect. The effects of refactoring method on software quality attributes vary a lot. Refactoring is able to reduce the complexity of the software. In this work, we are applying the refactoring on the OOCK metrics and have measureable effects on quality attributes.

Keywords: Software metrics, Software quality attributes, Refactoring methods.

CERTIFICATE

This is to certify that Rekha Rani has completed M.Tech dissertation proposal titled "**Software design enhancement using refactoring**" and its effects on Quality Attributes" under my guidance and supervision. To the best of my knowledge, the present work is the result of her original investigation and study. No part of the dissertation proposal has ever been submitted for any other degree or diploma. The dissertation proposal is fit for the submission and the partial fulfillment of the condition for the award of M.Tech in Information Technology.

Date: **Signature of Advisor:**

Name: Balraj Singh

ACKNOWLEDGEMENT

First of all, I would thank to Almighty god, who has always blessed me and for giving me strength to do this work. I wish to express my deep gratitude to my guide, Mr. Balraj Singh for his generous guidance. His guidance and support enabled me to conduct the research. Without his support, I would not be possible to complete this program. I owe my loving thanks to my friends and colleagues, without their cooperation, encouragement and understanding it would have been impossible for me to finish this work. Lastly, and most importantly, I remain indebted to my parents, well-wishers and Almighty for always having faith in me and for their endless blessings.

Rekha Rani

11306293

DECLARATION

I hereby declare that the dissertation proposal entitled, "**Software design enhancement using refactoring**" submitted for the M.Tech Degree is entirely my original work and all ideas and references have been duly acknowledged. It does not contain any work for the award of any other degree or diploma.

Date: Rekha Rani

Registration no.11306293

TABLE OF CONTENTS

LIST OF FIGURES:

List of tables:

CHAPTER 1
INTRODUCTION

A software handles the real world problems and so it adapts to make the changes that occur in problem domain. A software developed by the developer is passed through the number of phases. Different models are used for development of software. Some examples of the software models are Waterfall model, Evolutionary model, Prototype model, Incremental model, Spiral model and the Object-Oriented model. Software development phases include the requirement gathering phase, designing phase, programming phase, testing phase and the maintenance phase. During the development and the maintenance phase, we also change and improve the code. Changing and Improving of code is done by the fixing of bugs and by the addition of new requirements

1.1 Refactoring

One basic process is used to improve the code of software is called "Refactoring". Refactoring is the approach that is used for refining the internal part (code) of software without affecting its external activities. Refactoring is performed while improving the internal structure of the software by adding some new features in the software. Refactoring is a technique that is used to reduce the complexity of the software by improving its internal structure by fixing bugs or by adding new features. We used the software metrics – Object Oriented chidamber and kemerer metrics, in this six metrics are include that are used as internal attributes of the software. The examples of quality attributes are Adaptability, Completeness, Maintainability, Understandability and Testability. These quality attributes can act as external attribute of the software. This research work mainly concerns with three factors that are included for research first one is Refactoring Methods, second one is software metrics and third one is software quality attributes. Refactoring is used for improving the code is directly measured the software metrics and indirectly measured the software quality attributes.

One common concern with refactoring is that it also effects on the performance of the software. Refactoring is the one method to achieve the better quality. Refactoring is mainly concerned with the reengineering process. Reengineering is also a process that is related to the changes to improve the software quality. Refactoring is generally accepted for the purpose of maintainability of software system. The concept of refactoring is also considered as the transformations of the source code, without changing the visible behavior, to make the software stress-free to understand and reuse. Refactoring is used for the incremental improvement in the software quality through the different refactoring methods that are used in the code at right place. Different refactoring methods are applicable for particular purpose and effect. Refactoring is manual process that is used to remove or weaken quality defects for the improvement of the software quality. Refactoring is an active approach to extend the software life-time. The impact of refactoring on the software system varies. The poorly designed code is too harder to maintain, test and implement. The basic goal is related to refactoring is the safe transformation of the code to improve the quality. In software program, the word "Smell" means potential problem in the code. In the refactoring process, as the smell is found, refactoring methods is applied and improved the code. The classification of refactoring methods is prepared for the particular desired quality attributes and metric set. Sometimes, the unclear for the software developers exactly how to use refactoring methods to increase the software quality.

The refactoring applied on the software code is reduces the cost of software maintainability for the long time. Refactoring used in real world practice is improve the code for software maintenance and the improvement of the existing software. Refactoring method is also efficient for the solving problems and maintenance of existing software. The transformation in the software is done by some changes in the code. The process of restructuring or refactoring is transform the internal arrangement with the objective of accepting of its processes. In software development process firstly we designed the software system then we write the code for the implementation purpose. Refactoring have some positive effects on software quality attributes but also have some negative aspects. They also considered higher power consumption, longer execution time, additional memory used, less suitable for the safety critical applications.

Refactoring process is used to remove the weak defects and adding some new features to the code so that the software quality should be improved. Since the activities involved in the software, result of the product maintenance for the cost overruns in SDLC, helping to improve the maintainability. The tools and methods to support the refactoring process becomes important.

The present research in the field of software refactoring is active and has identified the usage of metrics for refactoring as a main research attribute. The previous examination in this field had resulted in behavior preserving methods for refactoring of the object-oriented software structures, tool supports to automatically refactor or reorganize the applications or methods for code based refactoring. Software restructuring or refactoring is used in the field of the software engineering for the upgrading of the structure and the understandability of the software parts of the development. The main use of the refactoring is to transform the program structure into better quality after setting the quality faults. Software refactoring recovers the basic Object Oriented strategy internal assets such as encapsulation, abstraction, message passing, inheritance and polymorphism. These assets are quantifiable and can be measured. The significant clue behind the refactoring is to restructure aspects and procedures through the class hierarchy in order to arrange for the future allowances. The motive of refactoring provide the techniques and tools for the software developers so that it make the program faster, remove bugs and improve the software quality and place the refactor methods at right place.

Sometimes the task of selecting the refactoring methods is time consuming and form conflict between the program writers. Refactoring is applied on the internal code changes the value of software metrics and hence the software quality attributes. It is not compulsory, all the refactoring methods improve the software quality, so we need to find those refactoring methods that are used to improve the software quality attributes. Refactoring is also helpful in minimizing the code duplication.

Refactoring has two explanations dependent on condition. Refactoring a change in the interior structure of software to make it stress-free, understandable and inexpensive to adapt without varying its external activities. Refactor to rearrange software by applying a sequence of refactoring without altering its recognizable behavior.

Refactoring is generally inspired by observing by code smells. For example the method at hand may be long, or it may be a matched with another close method. After recognized, such difficulties can be addressed by refactoring of the source code, or transforming it into a new form that implement same work as before but they have no bad "smells".

Benefits of Refactoring:

1. Without refactoring, the design of the program decays.
2. Refactoring helps to develop code more quickly.

3. Refactoring helps to find bugs.
4. Refactoring makes software easier to understand.
5. Refactoring improves the internal structure of software.

Refactoring is the procedure used for changing a software structure in such a manner that it does not modify the external activities of the code yet improves its design. It's a well-organized way to clear the code that decreases the chances of offering bugs. When you refactor you are refining the actual form of the code after it has been written.

In understanding of software improvement have faith in that design and then code. A best design derives when the software coding is complete. Over time the code will be form with better-quality and the trustworthiness of the system, its structure according to that design, slowly but surely fade away. The code gradually sinks from engineering to hacking.

With refactoring you can take a design that have bad code smells, confusion even, and modify it into well-organized code. Refactoring interchange a field from one class to another, pull some code out of a method to make into its new method, and push some code up or down order.

The refactoring process include the following activities:

a) Identify the software where refactoring is to be applied.
b) Determine which refactoring method should be applied on the software.
c) Guarantee about preserve behavior.
d) Apply the refactoring technique to the software.
e) Identify the effects of applying refactoring to the software.

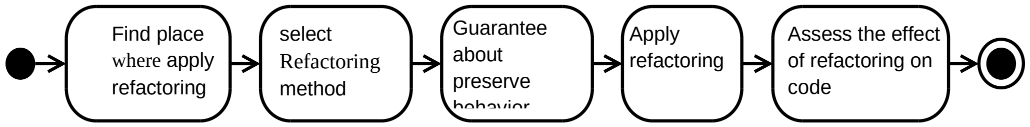

Find place where apply refactoring
select Refactoring method
Guarantee about preserve behavior
Apply refactoring
Assess the effect of refactoring on code

Fig.1.1 refactoring process

1.1.1 Refactoring Methods

There are many numbers of refactoring methods. The list mentions some refactoring methods that redistribute responsibilities within or between the classes and some refactoring that operate at the field level. The following table shows those re-factoring methods:

Refactoring Patterns	**Description**
Chain Constructor	Chains a set of constructors together to obtain a low amount of duplicated code.
Collapse hierarchy	Merges a super class and a subclass together.
Delete Attribute	Delete an attribute that is not being referenced by any class.
Delete Class	Delete a class and all references to it.
Delete Method	Deletes a method that is not being referenced by any class.
Encapsulate Attribute	Creates assessors for attribute and replace all the read and write operations to the attribute by calls to the assessors.
Extract Class	Create a new class and moves to it fields and methods from an existing class.
Extract Interface	Extracts method signatures and creates an interface for a class.
Extract Method	Extracts a piece of code to a new method.
Inline Class	Moves all the features of a chosen class into another class. Deletes the chosen class.
Inline Method	Replaces all the method calls of a specific method by the contents of that method.
Inline Singleton	Moves the features of a singleton to a class that stores and provides access to the object. Deletes the Singleton.
Introduce Ex-planning Variable	Extracts an expression to a local variable.
Move Attribute	Moves an Attribute to another class.
Move Embellishment to Decorator	Moves an embellishment code to a Decorator.
Move Method	Moves a method from one class to another.
Pull Up Attribute	Moves an attribute to a super-class or super-aspect of the current class or aspect.
Pull Up Method	Moves a method to a super-class of the current class.
Push down Method	Moves a method to one or more of its subclasses.
Rename Class	Changes the name of a class and in all the places that it is referenced.
Rename method	Changes the name of a method and all the method calls to a new name.
Replace Method with Method Object	Extracts a method from a set of selected statements to a new class, containing the extracted statements as a new method and the local variables of the method as fields of new class.

Table 1.1 Refactoring methods for the Object Oriented Software

Refactoring methods applied on source code after the calculation of internal metrics to improve the code. These method directly affect the software internal metrics and indirectly affect the software quality attributes. Applying refactoring tool for make refactoring efficient and can avoid the possibility of the presenting bugs in the code.

1.2 Software Metrics

Software metrics are used as internal quality attributes. A software metrics are the good events of measuring the software quality. Software metrics offer a means to extract valuable and quantifiable data about the structure of the software system. The software code has a very large list of metrics in order to understanding the structure and quality of the system. The software metrics help us to establish the value of class. Measuring the complexity measures of the software system is the basic method to estimate the maintainability of the software. If the calculated result has higher value then its mean is program is too complex and very hard to maintain it.

To classify the refactoring method based on the internal quality attributes, we choose metrics based on the objected oriented concepts. We consider the OOCK (Object Oriented Chidamber and Kemerer) metric suite which consist of six metrics that are measured: Depth of Inheritance Tree(DIT), Weighted Methods per Class(WMC), Number of Children(NOC), Response for a Class(RFC), Coupling Between Object Classes(CBO), Lack of Cohesion on Methods(LCOM). The metrics we investigate [1] that are the following:

a) DIT (Depth of Inheritance) is defined as the length of the longest path from a given class to the root class in the hierarchy.
b) Weighted Methods per Class also termed as (WMC) is defined as a no. of methods defined in a given class. Traditionally, it is used to measure the complexity of an individual class.
c) Number of Children (NOC) is termed as the no. of classes that inherit directly from the given class.
d) Response of a Class abbreviated as (RFC) is defined as the number of methods that can be executed in response to a message being received by an object of that class.
e) Coupling between Objects or (CBO) is explained as a number of distinct and non-inheritance related classes for which a given class is coupled.
f) Lack of Cohesion on Methods also called as (LCOM) is defined as the number of pairs of member functions who do not share an instance variable, minus the number of pairs of member with the shared instance variables but, the metric is set to zero when the subtraction produces a negative value.

1.3 Software Quality Attributes

Quality of software is a multidimensional concept. The term quality has a different meanings. It all depend on the users, customers and developers of the software system. Developers develop the product that satisfy the customer, what they want. The customer buy the software system according to requirement of the organization and user must support these software system. The user want the beneficial features that they easily work with the software system. Quality can be defined in different way, one party could be denoting to it in its broadest sense, where as another might be denoting to its specific meaning. The term quality is a portion of our daily language and the popular and professional uses of it may be very different. Based on the quality attributes, the refactoring methods [1] have been classified as:

a) Adaptability is defined as the ease with which a can dwell itself to the applications or environments other than those for which it was designed.
b) Completeness is defined as the degree to which the selected component implements all the required functionalities.
c) Maintainability can be defined as the ease with which a component can be modified to correct its faults, or improve performance of other attributes.
d) Understandability is termed as the degree up-to which the meaning of a software component is clear to the user.
e) Testability can be said as a set of attributes of software that bear on effort needed to validate the software product.

There is no direct method to measure the external software quality attributes. The internal metrics can be used as pointers for external software quality attributes. Some existing research study shows the correlation between the external quality attributes and the internal quality metrics. The outcome of the effect of the restructuring on the software quality has an extensive scope. Different refactoring methods are linked to the different quality attributes. By applying the different refactoring method enhances the software quality attribute.

1.4 Relation of Refactoring to metrics and software quality

Refactoring is taken as the process of transforming a piece of code or program of a software to improve the software quality by reducing the bugs and by adding the new features. Refactoring or restructuring is also important for decrease the complexity of the software at the all levels of SDLC and it also assure that the cost of product is not reach at higher level. Some problems are faced by some developers at the time of applying refactoring. Basically these problems are related with the human awareness and because of bad smells in the code.

The software metrics are used to find the place in the code where the refactoring should be applied in better way for the profit. Tool support is required to contribution the human awareness in the decision making process in an effective fashion.

Software quality attributes can be defined as the conformance of functional and non-functional requirements. Refactoring provide the definition that are related to the maintenance of the quality of the software. Refactoring is directly affected the internal metrics but indirectly effected the quality attributes. Firstly we calculate the internal metrics then we take the result about how these Calculation affect the quality attributes of software. Refactoring is useful to find the progress of the software system. Significant budget and improvement is participated in refactoring.

Refactoring classification for code is altering the code to make it improved in particular way. Altering code to make it stress-free to maintain. This code transformation by using the refactoring method is reduce the cost and effort of the software. Refactoring enhanced the quality by decreasing the size of the code and increasing the number of the procedures. Refactoring applied on the code will also increases the quality of the code. Refactoring can also be classified into – combining methods, data organizing, to make the conditional expressions simpler, make the method call simpler and dealing with inheritance, encapsulation, abstraction and generalization. Today, in world the various number of tool is present to support the refactoring and restructuring. A refactoring is applied on existing software implementation so that result can be formed in the form of improved software implementation.

Refactoring is also used for adding new features in the software so that the software will be easily reusable in other systems. Refactoring is important but also it is complex. There are number of approaches are used to evaluate the effects of refactoring on the software quality attribute. There are number of approaches are used to find the places where refactoring must be applied in better way. Refactoring is applied on some software also support the minimizing coupling and maximizing the cohesion.

One issue is regarding how different refactoring definitions are compared. Some refactoring issues are regarding with design, scope, procedures and actions that re performed by developer as a starting point to start the development of the program. The aim of refactoring is to make the code more readable form. Refactoring is also used for improve the extensibility of the software code. Refactoring also boost up the code reuse. Refactoring is the vital part of the software development process. Developers firstly write the test and then write the code to pass the test and then apply the refactoring to improve the quality of the software. The refactoring techniques also help to improve the internal consistency and clarity of the code. Mostly the useful software systems required the continuous progress and change. As the software system is boosted, altered and improved by using new requirements, the code become more complex. The major part of cost of software development process is devoted to maintenance process. The purpose of apply refactoring on code is better evolution of software while protecting the quality of software system. The best solution for reduce the maintenance effort is software code refactoring applied on the code.

In software development industry, from the opinion of project managers and developers quantitatively calculate the effect of refactoring and reconstructing on the software quality. If we doesn't know about the refactoring techniques then we cannot apply in beneficial way to the source code for improved the quality. Refactoring applied on code is also helpful to reduce the size of code. The size of code is reduced then it make the code less complex. The refactoring is fight with complexity and coupling in the source code. Refactoring is used to make the code make reusable by organizing the code in useful fragments. Refactoring applied on the code increase the performance of the software system.   Refactoring is applied on code is also helpful to make the code more understandable so that user and developers easily be familiar with the code.

1.5 **Reasons for Refactoring**

Why the people want to refactor the code or the program of the software. Some reasons are like that they want to remove duplication, reduce the size of code, breakup the long methods and also present the design patterns in the software system. People mainly attention on that how to make the software more effective and valuable and they also think about how we improved the programs for better performance. Mostly the refactoring is use for the duplication of code. Sometimes the similar code is defined in the program then refactoring applied on this type of code to move this code in the method. It also relate to gathering what belongs together and pushing it all in one place. So that it make easier to maintain.

Another problem encountered in the programming is some methods are very large so refactoring applied to make the larger method into smaller ones. Bugs found in the program is also removed by applying refactoring methods. Addition of new requirements for improving the quality of the software system. Refactoring support to facilitate to make the changes, understandability and for increased the performance of the software product.

1.5.1 Why refactoring is done?

It is valuable tool that helps you keep a good control on your code. Refactoring is a tool that can, and should, be used for a different purposes.

a. Refactoring Improves the Design of Software

Without refactoring, the design of the program will decay. As person change code modifications to understand short-term objectives or variations made without a complete information of the design of the code—the code lose its procedure. It becomes very hard to look the design by study the code. Refactoring is somewhat like clean up the code. Work is completed to eliminate bits that aren't actually in the correct place. It is very hard to look the design in the code, it very hard to preserve it, and the more quickly it falloffs. Even refactoring supports code keep its shape.

The poorly developed code generally takes more code to do the similar things, repeatedly since the code relatively accurate the same thing in a number of places. Thus main feature of refining design is to reduce the duplication part of the code. The importance of refactoring is also help in future adaptations to the code. Decreasing the size of code won't make the system to run fast for that reason the result on the footmark of the programs very hard to significant. Decreasing the size of code does, still made a large variations at the time of changing the code. The new code there is very hard to modify properly and there is a more code to understand. The refactoring help to removing the duplicates, you confirm that the code declares the whole things once and only once, which is the principle of good design.

b. Refactoring Makes Software Easier to Understand

Refactoring makes the [1] code in more readable form. Refactoring is important for future development. Refactoring helps to understand unfamiliar code. When user see at unfamiliar code he will try to understand what code does. Refactoring transforms the code to make it better which can be easily understood.

c. Refactoring Helps Find Bugs

Refactoring helps to understand the code easily and find the bugs. Refactoring technique is used to find if the refactored code is working properly. It provides deep understanding of what code does. Refactoring technique helps for writing robust code.

d. Refactoring Helps Program Faster

Refactoring is useful to improve code and the quality of the code.

When talk about the restructuring and refactoring then the people also see that it exactly work for improving the quality. Successful design, educating readability, decreasing the several bugs, all these factors improve the software quality.

A best design is important for quick software development. Without a good design code will work better for some time but because of poor design the system will slow down. Refactoring is done by fixing bugs and reducing a number of errors more helpful to improve code than adding the new functions. Modification take the more time to understanding the code and for find the duplicate code. New features are added to the code also difficult to handle.

A good design is important to continuing speed in software development. Refactoring helpful for developing software more quickly and it stops the design of system from decay.

1.5.2 When to refactor the code?

When we talk about refactoring, the frequently examined how to be scheduled it. Must allocate the two weeks to perform the refactoring. In most of cases, set the time for perform the refactoring activity. Don't decide to refactor, refactoring is applied want to do somewhat else, and refactoring supports to do other thing.

a. Refactor When Add Function

Main reason for refactoring is to understand the code that want to change. The code may be been written by other peoples or by owns. At any time think about what the code is doing, can refactor the code to make that understanding further straightaway. Then refactor it. This is partially for the next time, but mostly it's for understanding and clarify the code.

Another driver of refactoring is sometimes does not help to adding a new feature. To look at the design the design should be like that adding of new features is easily done by programmer and developer. In some cases, also fix the bugs. Do this partially to make future developments easy, but generally do it for find the fastest way. Refactoring is a fast and easy process. Once code is refactored, addition of feature can go much more rapidly and easily.

b. Refactor When Need to Fix a Bug

Fixing of bugs applied in refactoring approach also help to make the more understandable code. Developers and designers find that the refactoring activities work well with code for finding the bugs. The presence of bug how that the code is not clear for customers and the users of the software.

c. Refactor is As Do a Code Review

Code reviews help to spread the information and knowledge through the development team members. Reviews helpful for the less experienced people they get the knowledge from the more experienced developers. They also support to understanding more characteristics of a large software system. They are also very significant to write clear code. It’s very difficult for people to put themselves in the shoes of somebody unfamiliar with the effects they are working on. Reviews give the chance to people they also suggest some useful ideas. So in this way lot of ideas will got in a weeks.in this way the contribution of many peoples help to make easier the life, so most of people also follow the reviews for many purpose.

Refactoring also help to review the code of the software. Before start working with refactoring, read the code then understand some point of it, and make some ideas. Now when derive some ideas, study whether it can be easily implemented with the refactoring methods and approaches. After that performed refactor. Then check refactoring applied on code increased the performance. As a result, then applied the refactoring on second level of ideas, does not know about they may be refactored or not.

Refactoring also helps the code review to provide the actual results. Not only related to think about suggestion also to implement those suggestion.

To make this method work, work with small review groups. The reviewer advises changes, and they also decide whether the variations can be simply refactored in and then they make the changes.

CHAPTER 2

LITERAURE REVIEW

A number of research studies are included for the refactoring purpose. These describe the co-relation exist between the internal quality metrics and the external software attributes. A number research works explain the no. of different criteria: the refactoring activities are explained that are refining the features of the software.

Mohammad Alshayeb et.al Refactoring [1] used in educating design code of software using its internal metrics without making any effect on its external performance. Sometimes, it is often undecided for the software inventers how to use the refactoring methods to recover software quality attributes. In this paper, author planned the arrangement of refactoring based on the software quality attributes. This, in turn, help software inventers choose suitable refactoring methods that will progress the quality of their design, bas on their design ideas. It also enables them to calculate the quality drift caused by the specific refactoring methods. The arrangement is done by recording the changes in the internal quality metrics, produced by put on the refactoring methods, to the external quality attributes. This recording is based on research work that shows the relationship exist between the internal quality metrics and the external quality attributes. This research work was imperfect to examining the result of refactoring. (**Mohammad Alshayeb et.al, 2011**)

Tom Mens et.al this paper [2] offer the wide overview of current research works in the field of software refactoring. This research work is matched and debated based up no. of altered principles, the significant subject that are needed to be taken into the account when structure refactoring tool support, and the result of refactoring on the software process. Refactoring is the mostly the object-oriented modified of restructuring "the course of altering a software system in such a way that it does not alter the external performance of the system, yet improve its internal structure. Reformation is defined as "the alteration from one demonstration form to another at some qualified concept level, while conserving the subject system's external actions. A rearrangement alteration is often one of arrival, such as altering code to recover its internal structure in the old-style sense of structure designs. While rearrangement, generate new manners that implement or suggest change to the subject system, it does not usually contain alteration because of new requirements.

Still, it may lead to well explanations of the subject system that instruction changes that would improve features of the system.

In overall, recognized a need for formalisms, procedure and tools that address refactoring in a more consistent, basic, walkable and elastic way. While marketable refactoring tools being to increase, research into software reformation and refactoring stays to be active, and remainders needed to expose and address the limitations of these tools. (**Tom Mens et.al, 2004**)

Konstantinos Stroggylos et.al [3] organization of software suffer from adjustments, developments and enrichments to handle with growing necessities. This maintenance can origin that the quality to decrease. Several Metrics can be used to assess the way the quality is affected. Refactoring is the one of the utmost significant and frequently used practices of exchanging a portion of software is directive to expand its quality. However, even though it would be projected that the rise in excellence accomplished via refactoring is reproduced in the many metrics, quantities on actual life schemes indicate the reverse. To explored source code account switch system logs of popular open source software systems to notice changes manifest as refactoring and survey how the software metrics are pretentious by this process, in order to estimate whether refactoring is effectually used as a means to progress software quality with in the open source community. This research inspects the how the metrics of standard open source project were caused when the development group implemented refactoring, irrespective of the motives that led to that conclusions. This result, specify a substantial change of confident metrics to the inferior. Using a refactoring uncovering technique to classify the refactoring achieved each time, one could also relate each kind of refactoring to a precise drift in the change of several metrics and thus realize which ones or more favorable to the whole quality of the systems. (**Konstantinos Stroggylos et.al, 2007**)

Dirk Wilking et.al this paper [4] offers a practice assessment for the alert technique of refactoring centered on the language C. The basis for the estimation is made up by trial which is directed on the features of increased maintainability and modifiability. Even though the maintainability test show a minor benefit for refactoring. Regarding modifiability, the above of put on refactoring seems to even fail. The examination of subordinate variables offers clues on the benefits of the refactoring practices like compact resource consumption and a reduced existence of complex mechanism. In this research, a precise trial is offered evaluating the outcome of refactoring on the efficient features. However, universal result of refactoring on maintainability or modifiability could not be shown. As an alternative, an overhead aimed at the modifiability feature seems to exist as refactoring itself needs a sure amount of time for its finishing. A constructive characteristics of refactoring might be found in the “once and once only” project source, as these seems to cut the memory necessities of a system. As an accumulation, the three most significant refactoring found during this experimentation look to be “extract method”, “rename Method”, “comments” which might be a preliminary idea for straightforward refactoring maintenance in software tools. In accumulation, a different methodology to consider the status of refactoring is accessible directing on indirect expectations of why refactoring is practical and what challenging might to be solved. (**Drik Wilking et.al, 2007**)

Mohammad Alshayeb this research, examine [5] for practices to enhanced software quality and attain robust, consistent, and supportable software is continued. Refactoring is a method that expands the internal structure of software without disturbing its external behavior, is one technique that attain superior quality. Refactoring pattern is another. The objective of this research is to inspect whether refactoring patterns expands software quality. This is finished experimental by the metrics value of external quality attributes for different software systems earlier and next refactoring pattern is applied. To originate no dependable perfection movements in the software quality attributes. This is because each refactoring patterns performance has a specific determination and effect, and in future moves in software quality attributes are seems to be differently. Additionally, the properties of refactoring patterns on the external qualities exposed inconsistencies and were unclear and unspecified in several cases. In systems where refactoring patterns improved a quality in definite proportion of classes, it reversely fallen that identical quality in a related or even advanced proportion of classes. Based on this research exertion and the earlier education, can see that learning the consequence of refactoring and refactoring to arrangements as entire on software quality indications to unclear drifts in quality enhancement. The outcome and the assumption are equivalent with the outcomes of the research study directed to explore the properties of refactoring on the entire, not exclusively to the patterns, on the designated groups of quality attributes. In the later study, could not verify that refactoring in overall recovers the software quality. (**Mohammad Alshayeb, 2011**)

Bart Du Bois The procedure [6] of refactoring – reformation the source code of an object - oriented program without altering its external behavior – has been comprised by many object – oriented software creators as way to provide accommodations fluctuating requirements. The overall objective of refactoring is to acquire the maintainability of software. Inappropriately, it is unclear how exact quality aspects are affected. Consequently, this paper proposes to describe the impression of illustrative number of refactoring on the source code, prolonged with cross – references. How internal program internal quality metrics can be defined on the topmost of the program structure illustration and determine how to project the impression of refactoring on these internal program quality metrics value in the form of implications or improvements. Our techniques for examining the impression of refactoring on internal program metrics permit the interpretation of the implication or improvement of specific internal program quality metrics, as caused by the application exacting refactoring. (**Bart Du Bois, 2003**)

Noble Kumari et.al Software quality [7] is a significant topic in the improvement of effective software application. Many methods have been functional to expand software quality. Refactoring is one of those methods that are used to progress the software quality. But the outcome of refactoring in over-all on all the software quality attributes is ambiguous. The aim of this paper is to invention out the consequence of numerous refactoring methods on quality attributes ant to illuminate them based on their quantifiable conclusion on precise software quality attribute. This paper attention on the learning the reusability, complexity, Maintainability, Testability, Adaptability, understandability, fault proneness, stability and completeness attributes of the software. This, in turn, will support the maker in defining that whether to put on a definite refactoring method to expand a desired quality attributes. Our work contains that refactoring progresses the quality of software but developers need to appearance for the specific refactoring method for looked-for quality attribute. (**Noble Kumari et.al, 2014**)

Sultan Alshehri et.al the analytic [8] hierarchy process (AHP) useful in numerous fields and exclusively to multifaceted engineering difficulties and applications. The AHP is accomplished of constructing design difficulties and conclusion mathematically single-minded decisions build on involvement and information. This recommend that AHP should demonstrate beneficial in responsive software development where composite conclusions occur regularly. In this paper, the AHP is used to rank the refactoring practices built on the internal quality attributes. However, refactoring may consume more time and efforts. So, maximize the benefits of the refactoring in less time and effort, AHP applied to achieve this purpose. Subsequently consuming the AHP to rank the refactoring techniques, it was initiate to be an important tool that delivers a noble idea for developers when they want to apply the refactoring patterns to improve the code. Considering the complexity, cohesion, coupling and code size when ranking the refactoring techniques could take many benefits to the development team such as code enhancing the code in short time and transferring the knowledge to the developers. The Extract Class and Extract Method were the most refactoring techniques have improved the code in our studies. However the other refactoring techniques have added values to internal code qualities as well. (**Sultan Alshehri et.al, 2014**)

Yoshiki Higo Refactoring [9] is set of procedures to increase the maintainability and understandability or additional attributes of software system without altering the external behavior of it and it is receiving a much consideration lately. But it is problematic to achieve suitable refactoring since the impression of refactoring should verify the cost. Consequently, before a refactoring is demonstration achieved, the conclusion and the cost of it should be estimated. The approximation make it potential for us to sufficiently evaluate whether each refactoring should be implemented or not. This paper shows that it is challenging for developers to execute suitable refactoring, and recommends a methods approximation refactoring result. The method has been executed as a software tool, and a case learning presented the approximation of the tool assisted the developer of the mark system to execute a suitable refactoring. In this paper, a scheme for refactoring consequence assessment was proposed. The outcome signifies how the z of program will change by implement the refactoring. Also a tool was established based on the projected method and a case learning was showed to calculate the practicality of the planned method. From this inspected, to achieve that the planned method is valuable for directing developers/maintainers to achieve actual refactoring. **(Yoshiki Higo, 2008)**

Jasmeet Singh, presents software maintenance [10] activities often cause design erosion and lead to increased software complexity and maintenance costs. Extract Class Refactoring attempts to address design erosion by identifying and pulling out extraneous functionalities from a class and distributing them to new classes. This thesis extends previous research in this area but improving a metric known as Structural Similarity between Methods (SSM) used during Extract Class Refactoring. The improved metric, called Variable based Similarity between Methods (VSM), establishes similarity between methods based on the variables they share, and on how they use these variables. Strongly connected methods are then allocated into new classes. This paper also introduces another metric, Cognate Members Metric (CMM), which identifies those members of a class that are only used in combination with each other, and hence, probably belong together in a separate class. Additionally, this work extends and modifies existing refactoring processes for extracting classes. A software prototype that performs Extract Class Refactoring has been developed to substantiate the research. A few case studies are discussed and comparison and analysis of results of refactoring using the new and older approaches of the Extract Class Refactoring process are presented.

Furthermore, an implementation of the proposed research and the previous research is used as the basis for comparative study which verifies the applicability of the research. The implementation tool is further enhanced to provide the developers with the opportunity of selecting the final design and then automatically refactoring the code to that design. Lastly, software metrics are applied against the final refactored classes to substantiate the effectiveness of the research. Our result indicates that our approach leads to improvements in the overall design of the refactored classes and that in general there is scope for further improvements in the process. (**Jasmeet Singh, 2013**)

Serge Demeyer et al in this research work [11] offer an exhaustive impression of obtainable exploration in the field of software reformation and refactoring, form an official as well as a useful point of view. Next to recommend a wide-ranging list of open queries that designate upcoming study directions, and offer some limited answers to these queries. The research in software reformation and refactoring remains to be dynamic. While marketable refactoring tools are being to increase, there are still a percentage of open matters that persist to be solved. In general, there is a need for processes, methods and tools, that address refactoring in a more standard, walkable and elastic way. (**Serge Demeyer et.al, 2003**)

Quinten David Soetens presents the economical aspect of refactoring by investigating what effect refactoring [12] has on the costs of adding new functionality to a system and by looking at the costs of the refactoring activity itself. To expect that the costs of adding functionalities will be lower after the system was refactored. Indeed it is generally accepted that refactoring has a positive inspiration on the maintainability of a system. However, it is purely based on anecdotal evidence and still needs to be reinforced by a through scientific and systematic investigation. In this research, the conducting a series of case studies on some “real-life” systems. On these cases we will determine where and when they were refactored and influence these refactoring had on the development costs of the systems. (**Quinten David Soetens, 2009**)

Marouane Kessentini et.al present techniques and tools that have been developed [13]to support maintenance activities in demand to progress software quality. One of the most efficient ones is software refactoring to eliminate bad-smells. A majority of existing work suggest “typical” refactoring explanations that can be applied by hand for each kind of fault. However, it is hard to demonstrate or guarantee the simplification of these explanations to any kind of bad smells or software codes. In this paper, recommend a method to correct bad smells using well designed code. To use genetic algorithm to generate correction solutions defined as a grouping of refactoring procedures that maximize, as much as possible, the correspondence between the modified bad-smells and examples of well-designed code. To report the results of a progress of our approach using four open-source projects. Our proposal achieved high correction scores by fixing the majority of expected bad-smells.

To accessible a novel approach to the difficulty of fixing a bad-smells. Then used well designed code examples to correct bad-smells by applying refactoring sequences to maximize the similarities between the code to correct and these examples. After applying refactoring, jump by producing some clarifications that signifies a mixture of refactoring processes to apply. A qualification purpose computes, after relating the planned refactoring, the structural comparison between samples and calculate code. The projected method was tested on open-sources systems. Our research work displays that our technique offerings good grades on a precise test quantity. (**Marouane Kessentini et.al, 2013**)

Marjan Hericko et.al the goal of each software artifact is to accomplish an applicable level of software quality. Developers and designers [14] are demanding to produce understandable, trustworthy, workable, reusable and testable code. To support accomplish these objectives, a number of methods have been utilized. In this paper, refactoring procedure was used to estimate software quality with a quality index. It is composed of different metric collections which defines numerous quality aspects. It has been recognized that software metrics return software quality.

They have been generally used in software quality quantities. The outcomes of these calculations specify which parts of software essential to be reengineered. Furthermore, creators and designers attempt to attain higher software quality after creation source code alterations (refactoring). However, a number of quality metrics have been projected that define changed software quality aspects. In this research, a quality key metric has been useful to estimate the refactoring effect on software quality. (**Marjan Hericko et.al, 2010**)

S.H. Kannangara et.al Quality software are robust, [15] trustworthy and stress-free to maintain, and as a result decreases the charge of software maintenance. Subsequently software systems undergo alterations, enlargements and developments to manage with changing requirements, quality of software can be reduced. Although software system is growing, refactoring is one of the approaches which have been useful with the determination of refining the software quality. Refactoring is defined as the process of improving the proposal of the current code by altering its internal structure without disturbing its external activities, with the key goals of improving the quality of software product. For that reason, there is a faith that refactoring improves quality aspects such as understandability, flexibility, and reusability. However, there is restricted practical suggestion to support such expectations.

The motive of this learning is to validate/invalidate the statements that refactoring increases software quality. Experimental study methodology was used to attain the objective and ten particular refactoring procedures were used for the exploration. The whole effect of particular refactoring techniques and the impact of specific refactoring procedure were evaluated based on external actions namely; analyzability, variability, time activities and resource consumption.

After examining the experimental outcomes, whole study finished up with the effect that refactoring weakens the code quality. However, different analysis indications that certain refactoring procedures increase the code quality while rest is weakening the code quality. Furthermore, amongst the verified ten refactoring procedures, “Exchange Conditional with Polymorphism” classified in the highest as having high proportion of improvement in code quality and “Introduce Null Object” was classified as worst which is having highest measurement of deteriorate of code quality among the evaluated ten refactoring procedures. (**S.H. Kannangara et.al, 2014**)

Frank Simon et.al Refactoring [16]is the one of the most important topic that increase the software quality throughout the entire software development. Since classifying structures where refactoring should be valuable every so frequently is described with particular observations like “bad taste” or “bad smells” a programmed refactoring place detector appears hard. To appear that a special kind of metrics can support these particular observations and thus can be used as efficient and competent way to get maintenance for the result where to put on which refactoring. Due to the information that the software designer is the last specialist to offer powerful and metrics established software positive thinking to provision the developers refereeing their products. In this paper, prove this approach for four representative refactoring and existing both a tool associate the documentation and case lessons of its presentation. Our effort indications that metrics can benefit to classify exceptional variances for positive refactoring. Like Fowler, consider that the developer should be the last specialist for the result where to apply which refactoring. On the other hand, tool support is essential to support the human awareness in a very well-organized and actual way. To trust that software conception based on fixed structure analysis and metrics is a main topic for this task. To limit our provision of awareness we think on some characteristic of refactoring that attention on the associates of a class, i.e. methods and attributes. Briefly present both these refactoring and their conforming “bad smell” which should benefit to recognize parts of the system where to put the conforming refactoring. (**Frank Simon et.al, 2001**)

Quinten David Soetens et.al Refactoring [17] is generally familiar as a way to increase the internal structure of a software system in demand to confirm its long-term maintainability. Subsequently, software projects which accept refactoring practices should realize decreases in the complexity of their code base. Statement is evaluate on an open source system —namely a Java source code analyzer— and exposed that stages of refactoring did not distress the cyclomatic complexity. This paper considers this counterintuitive phenomenon concluded a complete examination of the real source code operations applied on the system under study. . Based on their work with IBM, they expressed the “laws of software development”. The refactoring can be described as:

- *Continuing Modification*: A program that is recycled in an actual world environment must alter, or converted more and more useful in that environment.
- *Increasing Complexity*: As a program changes, it turn into more difficult form, and additional resources are desired to reserve and make simpler its structure.

The law of "Continuing Modification" offers a darwinistic vision on the life-cycle of a software system, mostly testifying that a software system must familiarize to its background in directive to continue. The law of "Aggregate Complexity" on the other hand look like the law of entropy in thermodynamics, saying that a system attempts towards a determined state of disorder except more energy is added. (**Quinten David Soetens et.al, 2010**)

Francisco Zigmund Sokol et.al Refactoring [18] is the act of varying software code, frequently to improve internal code quality, without varying its outer behavior. Soetens and Demeyer (2010) evaluated one software and exposed that code refactoring did not suggest in enhanced outcome for code quality metrics. In this work, extend the mining data from 256 software projects from Apache Software Foundation, using Metric Miner, a web application absorbed on supportive mining software sources studies. The quantitative analysis displayed that refactoring absolutely does not drop Cyclomatic Complexity. On the other hand, the qualitative study indicated that a refactoring be likely to to improve code in expressions of readability and maintainability. Refactoring is the practice of improving code quality without changing the external behavior of a software system. If useful and correctly applied, refactoring recovers the maintainability of a system, without altering its functionality. Thus, it is predictable that refactoring performs central to recovered outcomes in terms of code quality metrics, such as Cyclomatic Complexity, also this metric measures only one code quality attribute, namely, complexity. To understand the effect of refactoring on the quality of a system, an open source system, involving the require messages with the progression of the Cyclomatic Complexity metric.

In this paper, repeat and cover that study, examining the history of 256 different open source Java projects from the Apache Software Foundation. To suggest a more effective approach in mining those repositories using a web presentation planned to support mining software sources trainings, namely Metric Miner. To examined all the 256 java projects that occur in Apache Software Foundation and are obtainable via interface. To originate that refactoring performs do not decrease Cyclomatic Complexity meaningfully, checking the results of the imaginary study in a bigger dataset. However, when examining a small division of requires, to observed that refactoring have a tendency to increase system code in expressions of readability and maintainability. In addition, we present some empirical displays that recommend different patterns of the progression of Cyclomatic Complexity amongst projects. (**Francisco Zigmund Sokol et.al, 2013**)

CHAPTER 3

PRESENT WORK

In this chapter, we are going to present the problem of our research work, its objectives, the methodology that we used for our purposed approach and the introduction of the developed tool. In the 3.1 section we explain how we formulated our problem and what the approach we are going to use. In the 3.2 and 3.3 section, the objectives and the methodology of the work done. In the methodology the flow of our work with the help of flow chart is explained.

3.1 PROBLEM FORMULATION

Refactoring is the approach use for improving the internal structure of software without affecting its external behavior. Refactoring is a technique that is used to reduce the complexity of the software by improving its internal structure by fixing bugs or by adding new features. We use the eclipse tool for the implementation of the purposed work.

In our research we will use refactoring methods in the code and calculate its effect on the OOCK (Object Oriented Chidamber and Kemerer) metric suite and map this effect on quality attributes like adaptability, correctness, testability etc.

3.2 OBJECTIVE

The objective of this proposed work:

1. The consequential effect of refactoring on software external quality attributes.
2. Developing Process for Implement the refactoring on Object Oriented Chidamber and Kemerer metrics and quantify the changes in the context quality attributes.
3. Analyze and compare the adaptability of proposed Process on the Object Oriented Chidamber and Kemerer metrics.

Refactoring is basically related to restructuring process and it does not affect the behavior of the system. Refactoring affects the internal quality software metrics and the external software quality attributes. The overall objective of proposed work is to enhance the quality of the software by performing analysis and refactoring using a proposed process.

3.3 RESEARCH METHODOLOGY

Refactoring is an approach that is used for the improving the internal structure without affecting the external behavior. Software metrics are used as internal quality attributes of software and also called the characteristic of a software process. In order to classify the refactoring method based on the internal quality attributes, we will choose metrics based on the objected oriented concepts.

This proposed work is divided into following steps as under:

1. Develop a Process for refactoring in terms of quality attributes and propose an expected quality benchmark.
2. Select working software.
3. Perform analysis on the software and calculate its internal quality attributes.
4. Perform the refactoring based upon the already existing techniques and record their results.
5. Perform the refactoring as per the proposed process and desired bench mark.
6. Then calculate the internal quality metrics of software after refactoring. Let us say it 2
7. Examine the changes in the internal quality metrics and check the effects of these changes on the software quality attributes.
8. Demonstrate improvement in the architecture as per the proposed process by comparing it differ with the non-refactored (α) software and the results of the other already existing refactoring techniques.
9. Discuss the various parameters which have scored comparatively high on our process.

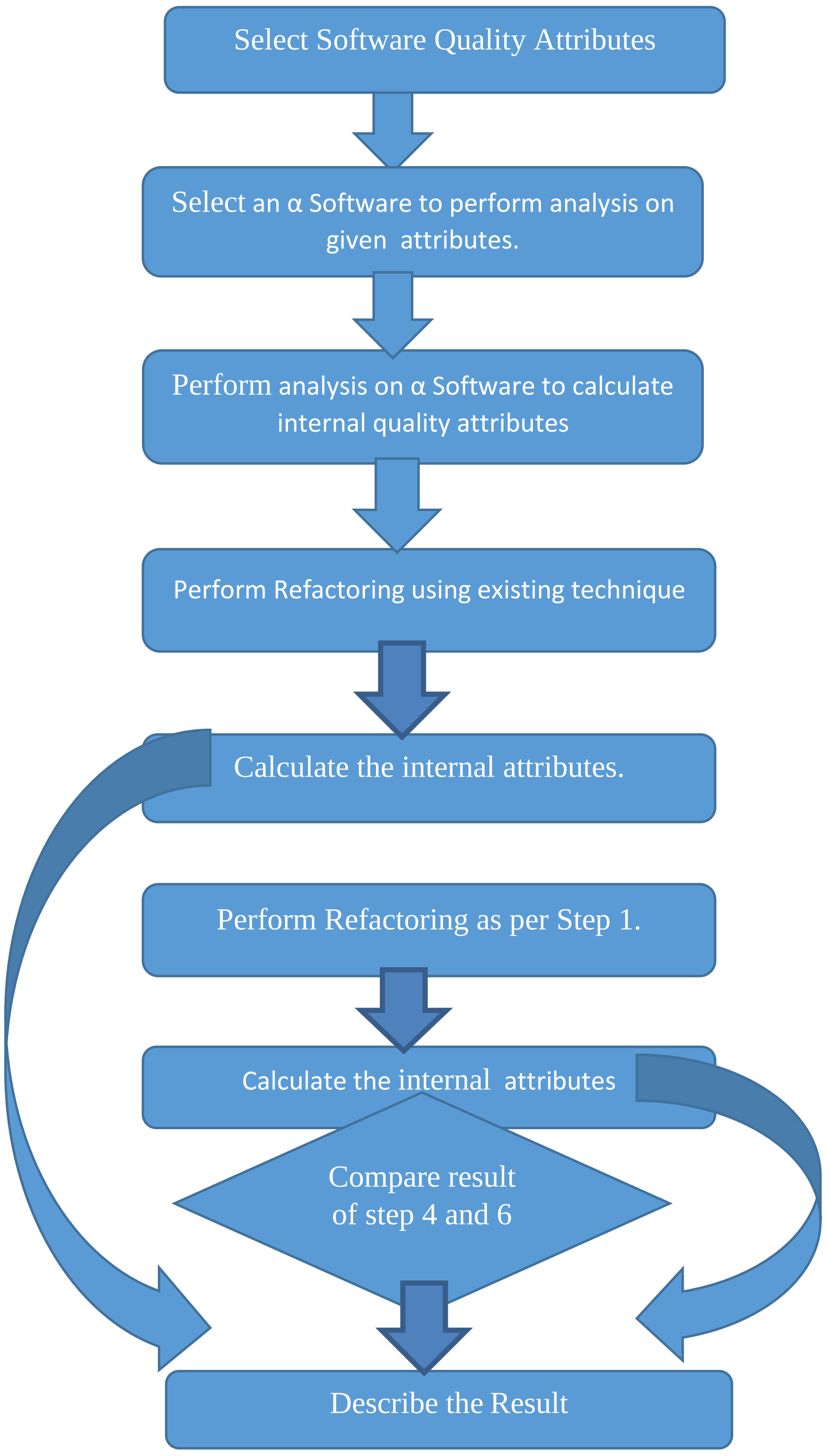

Select Software Quality Attributes
Select an α Software to perform analysis on given attributes.
Perform analysis on α Software to calculate internal quality attributes
Perform Refactoring using existing technique
Calculate the internal attributes.
Perform Refactoring as per Step 1.
Calculate the internal attributes
Compare result of step 4 and 6
Describe the Result

Fig: 3.1 the flow chart of proposed approach

In the first step, we will identify the software quality factor that we want to access by refactoring. Then we take the ckjm.jar file of java project in which we form refactoring. After applying the refactoring firstly calculate the internal metrics. These metrics values is calculate with metrics value that are calculated after applying the refactoring. When refactoring is applied in the code they affect the value of internal metrics. So we calculate the values of metrics for comparison with the actual software code. Then we map the result in the form of quality attribute. This result indicate the how much the software quality attribute effected by the refactoring on the actual code. Refactoring is done by the various refactoring method that improve the quality attributes. We used the eclipse tool for perform the refactoring. Eclipse is a general purpose open platform that provide tool for coding, building, running and debugging the application. Before apply the refactoring identify the place where refactoring apply but also guarantee about they preserve behavior. Does not affect the external behavior of the code. After that access the effect of refactoring applied on the internal metrics of the software.

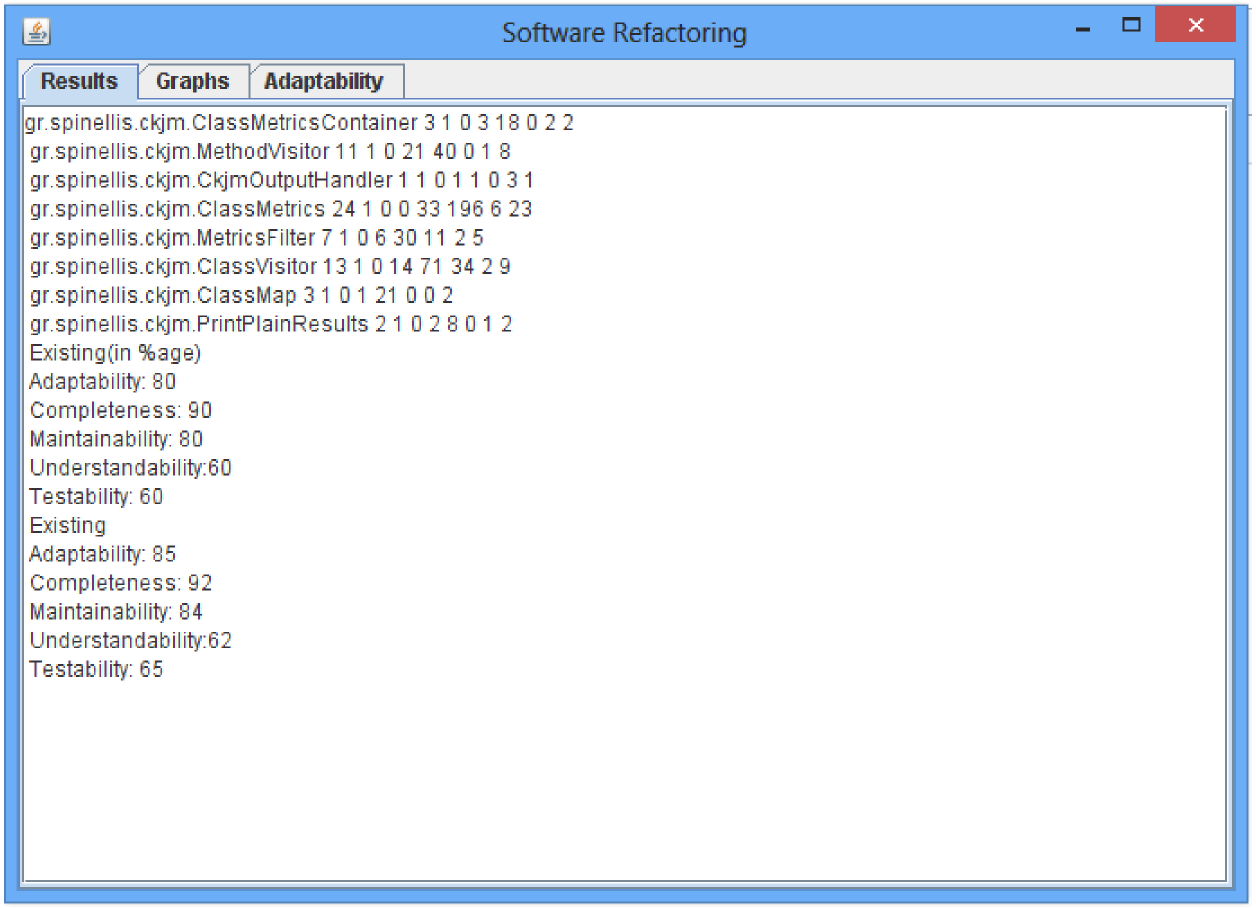
CHAPTER 4

RESULT AND DISCUSSION

In this chapter we have presented experimental result of our purposed approach with snapshots and graphs. In this chapter we will compare the results before applying refactoring and after refactoring. We report the changes in the internal quality metrics caused by applying refactoring methods on the selected course software projects and we also present an effect of refactoring methods on external quality attributes.

The fig. 4.3 contain the graph that indicate the changes between the metrics values before and after applying the refactoring. Fig 4.5 indicates the effect of refactoring on quality attributes.

The result that form by applying refactoring:



```
gr.spinellis.ckjm.ClassMetricsContainer 3 1 0 3 18 0 2 2
gr.spinellis.ckjm.MethodVisitor 11 1 0 21 40 0 1 8
gr.spinellis.ckjm.CkjmOutputHandler 1 1 0 1 1 0 3 1
gr.spinellis.ckjm.ClassMetrics 24 1 0 0 33 196 6 23
gr.spinellis.ckjm.MetricsFilter 7 1 0 6 30 11 2 5
gr.spinellis.ckjm.ClassVisitor 13 1 0 14 71 34 2 9
gr.spinellis.ckjm.ClassMap 3 1 0 1 21 0 0 2
gr.spinellis.ckjm.PrintPlainResults 2 1 0 2 8 0 1 2
Existing(in %age)
Adaptability: 80
Completeness: 90
Maintainability: 80
Understandability:60
Testability: 60
Existing
Adaptability: 85
Completeness: 92
Maintainability: 84
Understandability:62
Testability: 65
```

Fig 4.1 Result of the refactoring approach

Metrics	**Java software 1**	**Java Software 2**	**Variations**
WMC	**24**	**7**	**17**
NOC	**3**	**1**	**2**
RFC	**0**	**2**	**-2**
DIT	**8**	**6**	**2**
CBO	**33**	**4**	**29**
LCOM	**5**	**2**	**3**

Table 4.1 metrics value of software

This graph describe the metric calculation before and after applying the refactoring approach.

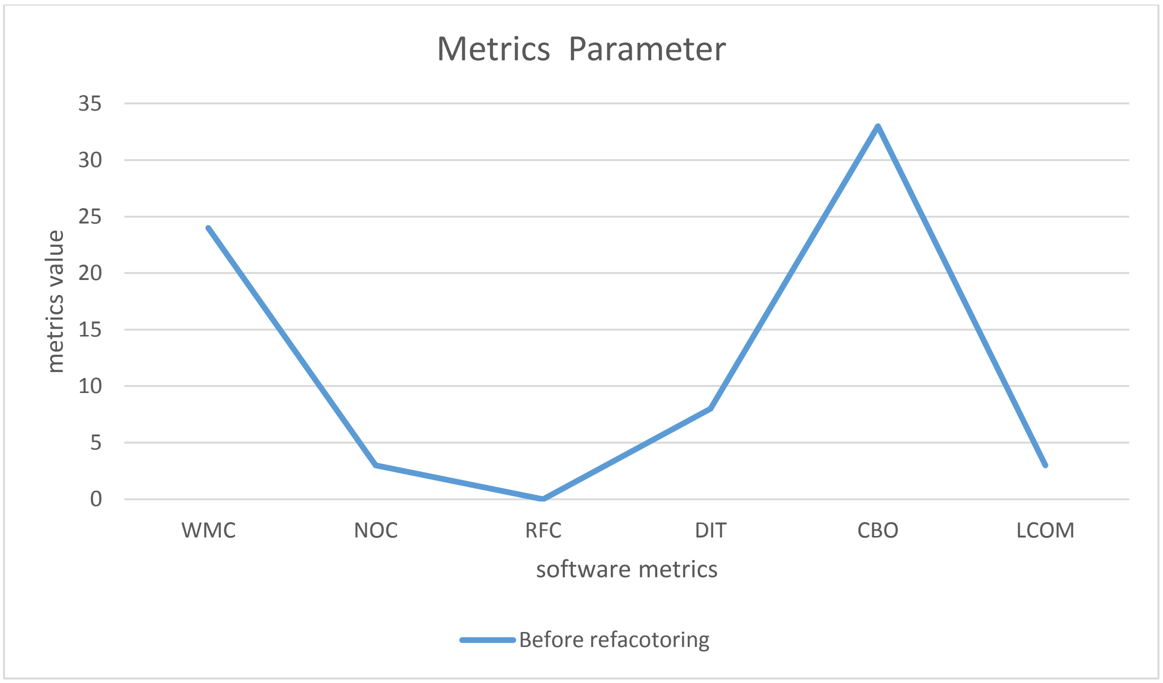


Fig 4.2 metrics value before refactoring

This graph indicates the metric values of Object Oriented Chidamber and Kemerer metric suite before apply the refactoring.

The graph for metrics value contain by applying refactoring methods at right place. This graph describe the metric calculation after applying the refactoring approach.

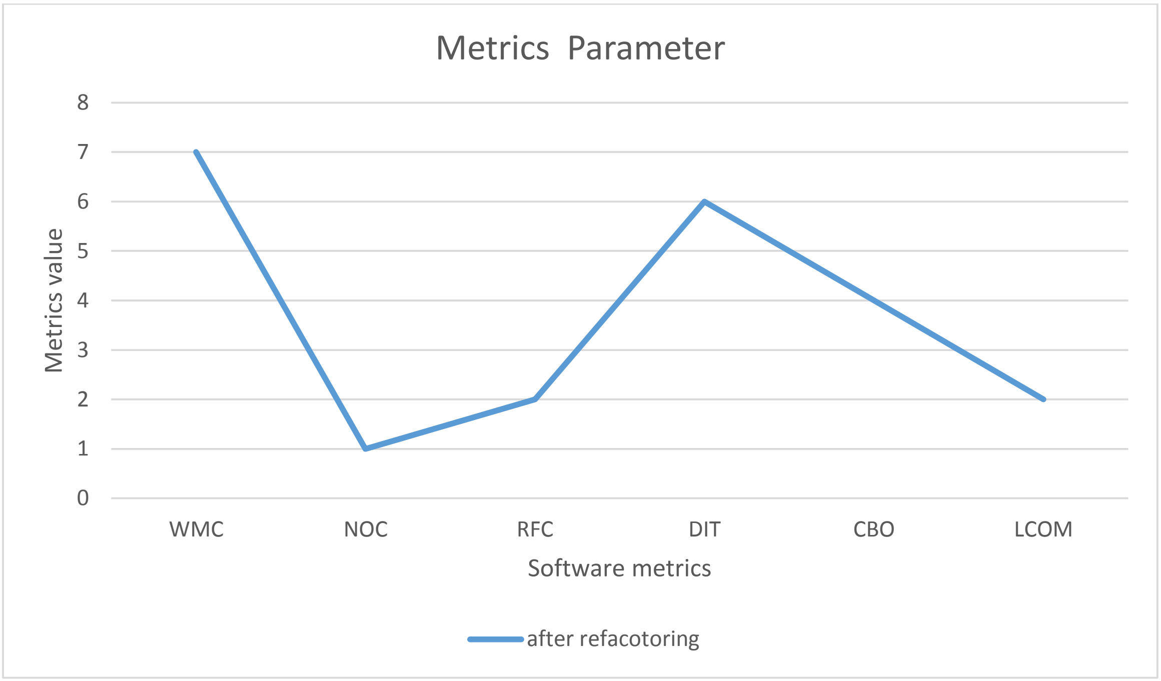


Fig 4.3 metrics value after refactoring

This graph indicates the metric values of Object Oriented Chidamber and Kemerer metric suite after apply the refactoring

Formula to calculate the Adaptability:

$\mathbf{S_a = ((R_p - R_f)/R_t) \times 100}$ where,

S_a: Software Adaptability

R_p: Code executed successfully

R_f: part of code fail to execute

R_t: Total Lines of Code

Formula to calculate the completeness:

Completeness = (no. of requirement full filled/ Total no. of requirement)*100

Formula to calculate the maintainability:

M = (Time Spent to fix a Bug / Total development time)*100

Formula to calculate the Testability:

T = (Time Spent to testing the functionality/ Development Time)*100

The graph of Quality Parameter before applying refactoring. The quality parameter show quality value for the quality attributes before applying refactoring. In fig. 4.4 blue line indicate the how much quality achieved by the actual code of the software in term of quality attributes.

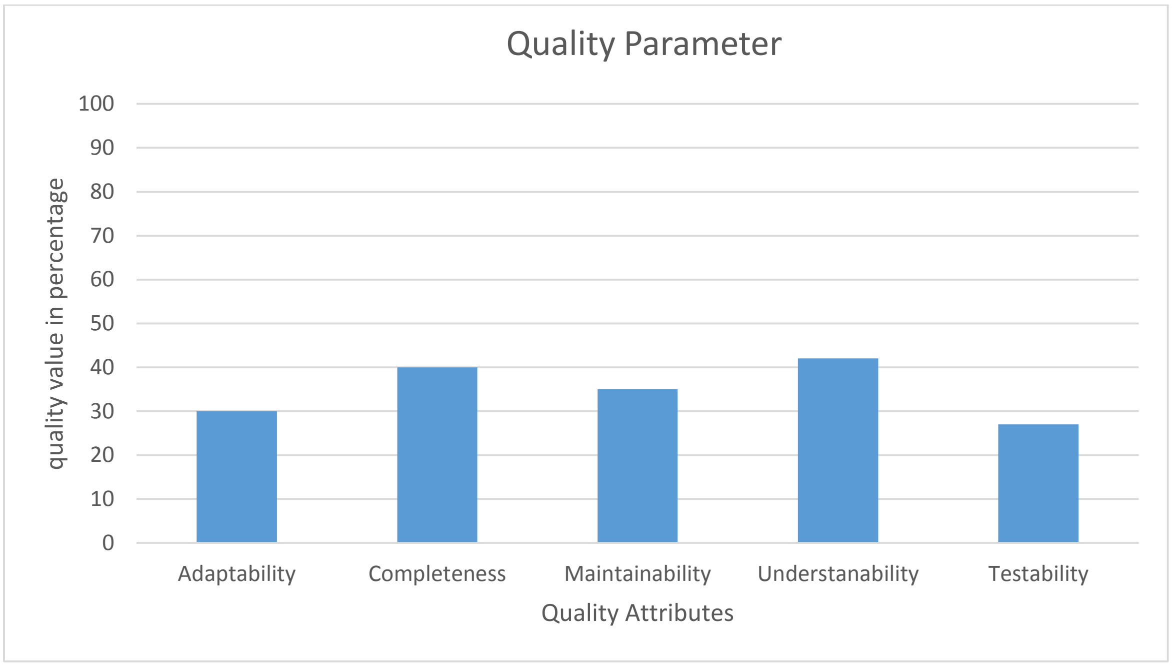


Fig 4.4 Quality attribute before refactoring

The graph of Quality Parameter after applying refactoring. The quality parameter show quality value for the quality attributes after applying refactoring. In fig. 4.5 blue line indicate the how much quality achieved by the refactored code of the software in term of quality attributes

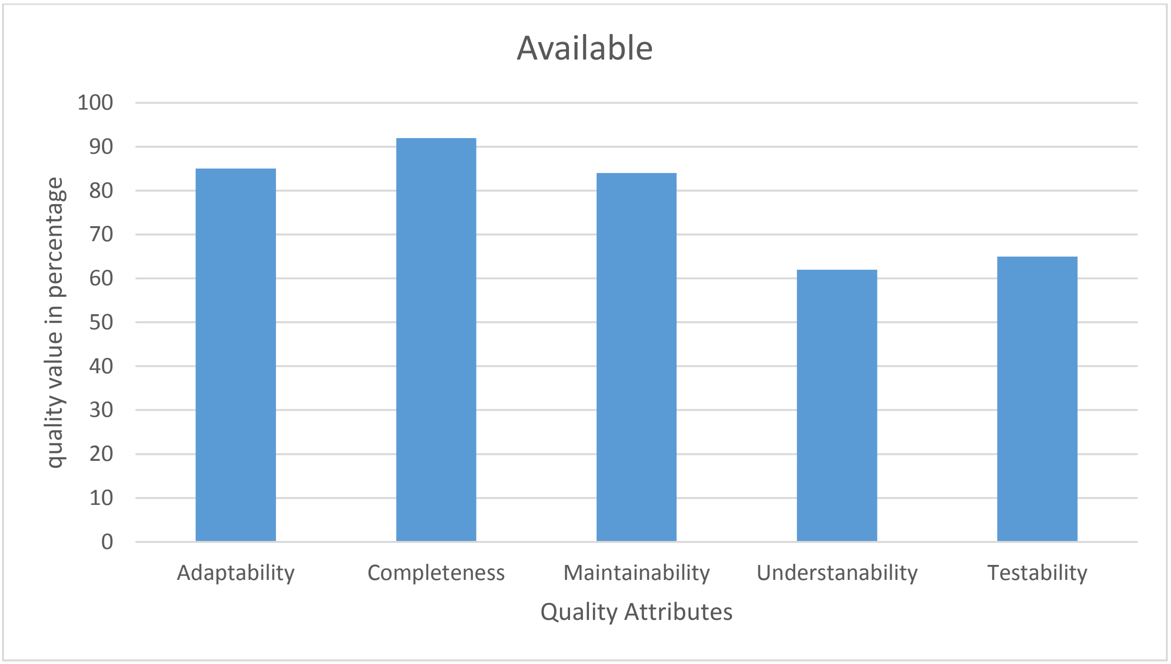


Fig 4.5 Quality attribute after refactoring

In fig 4.4 the x-axis show the quality attributes and the y-axis show the quality value in percentage. This graph show the enhancement in the software quality attributes.

% Improvement	Existing Quality	Enhanced Quality	Difference
Adaptability	30	85	55
Completeness	40	92	52
Maintainability	35	84	49
Understandability	42	62	20
Testability	27	65	38

Table 4.2 Comparison of quality parameter

This graph show the comparison between the existing quality and enhanced quality that are formed the by the applying refactoring on the code.

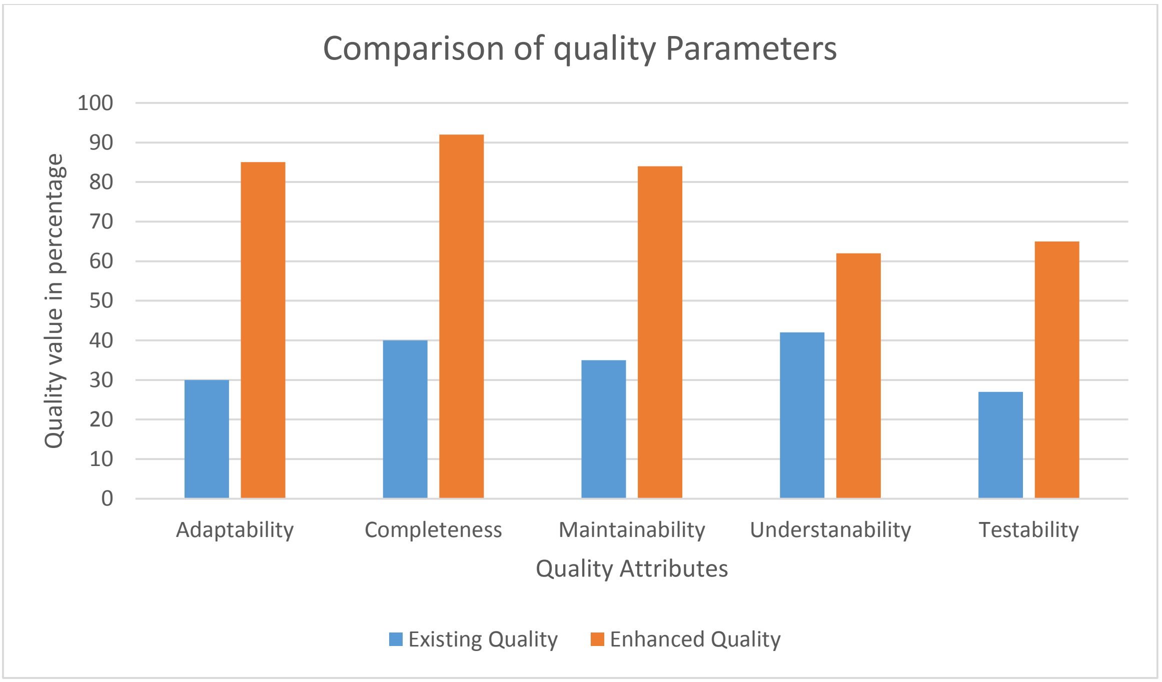


Fig 4.6 Comparison of quality parameters

In fig 4.4 the x-axis show the quality attributes and the y-axis show the quality value in percentage. This graph show the comparison of existing and enhancement software quality.

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

The refactoring is the important technique used to improving the quality of software. The poorly designed code is harder to maintain, test and implement and it degrades the quality of software. So the basic goal of refactoring is the safe transformation of the program to improve the quality. In this paper, we also examine the effect of changes on software quality attributes because of the internal quality metrics. These changes are based on the co-relation exist between the internal quality metrics and the external quality attributes. This research work also help the developer or designer of the software to make the improvement in software. We focused on the Object Oriented chidamber and kemerer metrics and different quality attributes which are Adaptability, Correctness, Maintainability, Understandability and Testability. Our research work achieves the increased quality of software but designers need to look for the specific refactoring method for required quality attribute. Research can also examine and prove the result on larger projects and can come up with common relation between refactoring and quality attributes. The method measures CK metrics from the original program and the revised one without actually performing the refactoring, and compares the metrics values. The comparison result represents how the complexity of the program will change by perform the refactoring. Also, a tool was developed based on the proposed method and a case study was conducted to evaluate the usefulness of the proposed method. Further, research on refactoring methods investigated for applying refactoring on the data set and another possible direction for future work to investigate the side effects of the refactoring apply on code.

CHAPTER 6

REFERENCES

[1] Elish, Karim O., and Mohammad Alshayeb, "A classification of refactoring methods based on software quality attributes," *Arabian Journal for Science and Engineering,* 36.7 (2011): 1253-1267.

[2] Soetens, Quinten David, "Refactoring Economics," (2009).

[3] Kumari, Noble, and Anju Saha, "EFFECT OF REFACTORING ON SOFTWARE QUALITY," *International Journal of Computer Science & Information Technology,* 6.4 (2014).

[4] Mens, Tom, and Tom Tourwé, "A survey of software refactoring," *Software Engineering, IEEE Transactions on* 30.2, (2004): 126-139.

[5] Alshayeb, Mohammad, "The impact of refactoring to patterns on software quality attributes," *Arabian Journal for Science and Engineering,* 36.7 (2011): 1241-1251.

[6] Alshehri, Sultan, and Luigi Benedicenti, "RANKINGTHEREFACTORING TECHNIQUES BASED ON THE INTERNAL QUALITY ATTRIBUTES," *International Journal of Software Engineering & Applications,* 5.1 (2014).

[7] Kessentini, Marouane, Rim Mahaouachi, and Khaled Ghedira, "What you like in design use to correct bad-smells," *Software Quality Journal,* 21.4 (2013): 551-571.

[8] Higo, Yoshiki, et al. "Refactoring effect estimation based on complexity metrics," *Software Engineering, 19th Australian Conference on*. IEEE, 2008.

[9] Du Bois, Bart, and Tom Mens, "Describing the impact of refactoring on internal program quality," *International Workshop on Evolution of Large-scale Industrial Software Applications*, 2003.

[10] Wilking, Dirk, Umar Farooq Kahn, and Stefan Kowalewski, "An Empirical Evaluation of Refactoring," *e-Informatica,* 1.1 (2007): 27-42.

[11] Stroggylos, Konstantinos, and Diomidis Spinellis, "Refactoring--Does It Improve Software Quality?," *Proceedings of the 5th International Workshop on Software Quality*, IEEE Computer Society, 2007.

[12] Mens, Tom, et al. "Refactoring: Current research and future trends," *Electronic Notes in Theoretical Computer Science,* 82.3 (2003): 483-499.

[13] Singh, Jasmeet, "Extract Class Refactoring by analyzing class variables," (2013).

[14] Gerlec, Č., and M. Heričko, "Evaluating refactoring with a quality index," *World Academy of Science, Engineering and Technolog,y* 63 (2010): 76-80.

[15] Kannangara, Sandeepa Harshanganie, and Janaka Wijayanayake, "An Empirical Exploration of Refactoring effect on Software Quality using External Quality Factors," *ICTer* ,7.2 (2014).

[16] Simon, Frank, Frank Steinbruckner, and Claus Lewerentz. "Metrics based refactoring," *Software Maintenance and Reengineering, Fifth European Conference on*, IEEE, 2001.

[17] Soetens, Quinten David, and Serge Demeyer, "Studying the effect of refactorings: a complexity metrics perspective," *Quality of Information and Communications Technology (QUATIC), Seventh International Conference on the*. IEEE, 2010.

[18] Sokol, Francisco Zigmund, Mauricio Finavaro Aniche, and Marco Aurélio Gerosa, "Does the Act of Refactoring Really Make Code Simpler? A Preliminary Study," *4th Brazilian Workshop on Agile Methods*, 2013.