Improving Reliability of Object Oriented Design: Defect Mitigation Perspective

THESIS

Submitted to
Babasaheb Bhimrao Ambedkar University
(A Central University)
Lucknow

BABASAHEB
BHIMRAO
AMBEDKAR
UNIVERSITY
LUCKNOW
प्रज्ञा शील करुणा
ESTABLISHED 1996

For the Degree of

Doctor of Philosophy

In

INFORMATION TECHNOLOGY

By:

Pawan Kumar Chaurasia

Enrollment No. 960/14

DEPARTMENT OF INFORMATION TECHNOLOGY
BABASAHEB BHIMRAO AMBEDKAR UNIVERSITY
(A CENTRAL UNIVERSITY)
VIDYA VIHAR, RAEBARELI ROAD, LUCKNOW-226 025 (U.P.), INDIA

2017

This thesis is dedicated to my parents

& family

for their love, endless support

and encouragement

DECLARATION

I, Pawan Kumar Chaurasia, solemnly declare that this thesis of research on **'Improving Reliability of Object Oriented Design: Defect Mitigation Perspective'** is my original work. The study has been conducted under the guidance of Professor R. A. Khan, at Department of Information Technology, Babasaheb Bhimrao Ambedkar University (A Central University), Lucknow. It is further declared that to the best of my knowledge and belief it has not been submitted earlier for the award of any degree and also undertaken that the thesis is essentially free from all kinds of plagiarism.

Dated: …..……

(Pawan Kumar Chaurasia)

Research Scholar
Department of Information Technology
Babasaheb Bhimrao Ambedkar University
(A Central University)
Lucknow-226025, India

BABASAHEB BHIMRAO AMBEDKAR UNIVERSITY
LUCKNOW
प्रज्ञा शील करुणा
ESTABLISHED 1996

Babasaheb Bhimrao Ambedkar University
(A Central University)
Vidya Vihar, Rae Bareli Road, Lucknow - 226 025.
बाबासाहेब भीमराव अम्बेडकर विश्वविद्यालय
विद्या विहार, रायबरेली रोड, लखनऊ - 226 025

CERTIFICATE

This is to certify that the thesis titled **"Improving Reliability of Object Oriented Design: Defect Mitigation Perspective"** submitted by **Mr. Pawan Kumar Chaurasia** is an original research work and has not been previously submitted in part or full for the award of any other degree or diploma to this or any other university.

The thesis submitted to Babasaheb Bhimrao Ambedkar Univesrity Lucknow satisfies all the requirements as stipulated in the *Doctor of Philosophy (Ph.D.) regulations-1999 as amended in 2013* and it is fit for submission and evaluation for the award of the degree of Doctor of Philosophy of the University.

Date: Supervisor

Head of the Department

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my guide and supervisor Prof. R.A. Khan for the continuous support of my Ph.D. study and research for his patience, motivation, enthusiasm and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I am short of words to convey my real feelings for his invaluable help and concern together with 'scholarly, insight and critical' guidance. I could not have imagined having a better advisor and mentor for my Ph.D. study.

Besides my advisor, my sincere thanks goes to my colleague Dr. Alka for her encouragement, insightful comments and research questions. I am very thankful to Dr Dhirendra Pandey and Dr Raj Shree for providing moral support, encouragement and consultation during the course of study. I am also thankful to all research scholars and office staff of the Department for their assistance and support.

I am thankful to Dr. R. K. Jaiswal, Director, UPTECH Computer Consultancy Ltd. allowing me the permission for data collection. I also express my thanks to entire staff member, for supporting me in conducting the experiment. I express my sincere thanks to all the expert for rewarding me for their valuable remarks during the expert review.

I would like to express my sincere gratitude for my parents Late Smt. Shakuntala Devi and Sri Shiv Shankar Chaurasia, for giving birth to me at the first place and supporting me spiritually throughout my life.

I owe my loving thanks to my wife Seema Chaurasia. She has lost a lot during my research. Without her encouragement and understanding it would have been impossible for me to finish this work. I am thankful to my loving son Anurag and Anmol for his be loving and support. I have neglected some of the duties towards them and appreciate their tolerance even in the worst condition.

Finally, I would grateful acknowledge to my Vice-Chancellor Pof. R. C. Sobti, Babasaheb Bhimrao Ambedkar University, Lucknow for his invaluable support, encouragement and inspired me.

Pawan Kumar Chaurasia

ABSTRACT

With the development of electronic gadgets, people have become highly dependent on these gadgets in our daily lives. The gadget revolution is accelerated by technological advancement in software as well as hardware. Today, software has fully captured our modern society. It is embedded in mobiles, home appliances, aviation, automobiles, industries and military. Software makes every work easier, faster and accurate etc with less cost. It has become a driver for everything from elementary items to genetic engineering. Every work can be done easily with the help of software based devices. This is the reason software has crept into every industry including telecommunication, petroleum, banking, railway, hospital, pharmaceutical etc. All these are highly dependent on software for their basic functioning.

Today, software industry develops a large volume of software to fulfil customers and users demand. It provides facilities to the users, but on the other side it also increases the possibility of failure/errors/faults/bugs with high complexity, increased size and unreliable software. Due to unreliable software, industries face a lot of problem and customers pay in form of economy or users lost their lives. In November 2000, therapy planning software in a series of accidents, created by Multi data Systems International, USA was not able to calculate the accurate dosage of radiation for patients undergoing radiation therapy. It allowed a radiation therapist to draw Multi data on a computer screen. Due to over dosage, eight patients died. Other 20 patients likely to cause significant health problem. In 2003, New York, Northeast blackout happened due to the programming error. The failures occurred when multiple systems tried to assess the same information at the same time and acquired the busy signals. Actually, the software could allow one system presidency at a time. Unfortunately, due to several requests at a time the software could not perform properly. The several incidences have forced the industry to develop reliable software. Though, it is not possible to develop 100 percent reliable software the developers and researchers have been making an effort to improve reliability of software under process.

Software reliability is one of the attributes of software quality. Software quality is an important factor because software systems are playing a major role in today's software development environment. There are various mechanisms being

used for addressing reliability but these exhibit their own limitations on implementation. Hence, it is difficult to decide whether the existing approaches are adequate for improving software reliability or not. In today era, for improving reliability most of the organization used object oriented programming (OOP) as OOP supports multiprogramming. By using object oriented approach the system could be designed at the high level. Class, method attribute, encapsulation, inheritance, cohesion, coupling and polymorphism are the important constructs of object oriented design. During design phase, the entire software architecture is designed and developed. At the time of software design, the designer may take various decisions for the development of the system. Design phase is the only phase where the programmer must decide about coupling, cohesion and inheritance etc for producing reliable software. High reliable software makes the system less complicated and easy to understand. Therefore, design phase is the most important and suitable phase for reliability improvement of an object oriented design. With the demand and significance of addressing reliability in the design phase, the research work in the thesis has been persuaded on the *Improving Reliability of Object Oriented Design: Defect Mitigation Perspective.*

A framework is proposed for reliability improvement of object oriented design to mitigate the design defects by minimizing/maximizing the usage of object oriented design constructs. The framework reflect that high design defect is one of the major causes for decreasing the reliability of object oriented design. It is an integrated method for improving reliability and mitigating defects of an object oriented design. The methodology not only mitigate the defect but also find the answer to the question like "Up to how much extent the designer has been able to mitigate the defects". The proposed framework consist of four phases namely *Identification Phase, Mapping Phase, Measurement Phase and Improvement Phase.*

The first phase is the identification phase. In this phase, object oriented design constructs, object oriented design defects and reliability factors of the object oriented design are identified. Design defect has been taken as a major factor to decrease reliability. In the second phase i.e. mapping phase, two mappings have been established; the relation between design defects and object oriented design constructs including inheritance, cohesion and coupling etc. as well as the relation between object oriented design defect and reliability of the object oriented design have been

mapped. The phase also analyzes the impact of inheritance, cohesion and coupling on design defects and impact of design defect on reliability. Third phase is the measurement phase. In this phase, design defect and reliability of object oriented design is computed. For mitigation of the defects Reliability Improvement Guidelines of Object Oriented Design (^{RI}GOOD) has been proposed in the improvement phase of the framework.

To measure the design defect, object oriented design metrics are considered. For inheritance, cohesion and coupling, Depth of Inheritance Tree (DIT), Coupling Between Objects (CBO) and Cohesion Among Methods in Class (CAMC) are selected. The research is focused on the fact that rigidity is one of the major factor contributing to design defect. Hence, with the help of the proposed framework, Rigidity Estimation Model (RiEM) has been developed. By using the model, reliability is calculated with the proposed Reliability Estimation Model (ReEM).

The proposed Rigidity Estimation model and reliability estimation model along with Reliability Improvement Guidelines of Object Oriented Design (^{RI}GOOD) are validated. For the purpose, five hypothesis are generated for significance of the proposed work. In addition, designs of 10 realistic projects of B.Tech, MCA and M.Sc. (I.T.) students are collected from UPTECH consultancy Pvt. Ltd., Lucknow. Values of OOD metrics and design defect is computed for these object oriented designs. All these designs are then treated with the help of Reliability Improvement Guidelines to mitigate defects. Comparison of the values of reliability of these designs before and after the treatment showed significant improvement in reliability. It is evident from the validation that reliability improvement model and rigidity estimation model are able to estimate defect and reliability of object oriented design. Hence, all the five null hypothesis are strongly rejected on (0.05) alpha level of significance for two tail test. Therefore, the corresponding alternate hypothesis are accepted and hence the model is accepted.

ABBREVIATIONS

DIT	:	Depth of Inheritance Tree
CBO	:	Coupling Between Objects
CAMC	:	Cohesion among Method in a Class
RiEM	:	Rigidity Estimation Model
ReEM	:	Reliability Estimation Model
OO	:	Object Oriented
OOD	:	Object Oriented Design
SDLC	:	Software Development Life Cycle
TBF	:	Time between Failures
NHPP	:	Non- Homogeneous Poisson Process
WMC	:	Weighted Methods Per Class
NOC	:	Number of Children
LCOM	:	Lack of Cohesion in Methods
RFC	:	Response for a Class
NAC	:	Number of Ancestor Classes
CTI	:	Coupling through Inheritance
CTM	:	Coupling through Message Passing
CTA	:	Coupling through Abstractions
RIGOOD	:	Reliability Improvement Guidelines for Object Oriented Design

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

CHAPTER 1: INTRODUCTION

"Quality is never an accident; it is always the result of intelligent effort."

-- John Ruskin--

1.1 Background

Today, software plays important role in various fields. It is defined as the collection of programs, functions, rules and subroutines with the operation. It contains all the related documentation necessary to understand and use the solution[1]. With the development of electronic items and digital computers, life of people is fully dependent on electronic gadgets in our daily lives. Software is embedded in mobiles, television, home appliances, automobiles, aircraft, medical equipments etc. All the electronic devices are based on software. It is used to assist solving business problems, industries, decision-making, scientific industry, medical field and engineering problems. Science and technology both demand high quality hardware and reliable software for making improvements and throughputs. Increased demands of software have imposed changes in its ability of quality design, test and maintenance.

Modern computer industry and electronic systems typically contains a significant amount of software. With the development of the industry, the size and complexity of software is also grown during the past decades and these trends will continually persuade in the future. Software is a part of system solution that can be executed on a computer in the form of instruction. Due to increased dependency on software, it is the responsibility of the developer to provide quality and reliable software to the society. In view of quality, software reliability is becoming more important. Software is used in many applications in various fields. Software development is badly affected by strict product delivery deadlines. This is one of the reasons for the improper behavior or failure of the software. Software failure affects seriously the field of aviation, telecommunication, industry, handheld devices, home appliances etc. Even a single defect in the software may affect billions of people life. With the increase in the dependency on software, the possibility of defects and hence software failures increases. These failures cause economic damage as well as loss of life. In other words, we can say that software can also kill people and society. Literature survey reveals that failure of software led to serious effects on business,

economy and society[2]. Even the most powerful business organization such as Microsoft is fearful from "killer bugs"[3]. Hence, the reliability of computers systems has become a major interest for our society.

There happened many casualties due to software failure in the field of aviation etc. and many hazards made people lost their lives[4]. L. Lee wrote in his book that various patients have lost their lives in 1985 and 1986 due to software errors occurred in Therac-25 radiation therapy machine[2]. In 1990, a fault in a switching system software caused heavy long distance networks and another series of local phones equipment failure[2]. In October 1992, the Computer Aided Dispatch (CAD) system of London Ambulance service broke down right after its installation, deactivating the capability of world's largest ambulance service, which handles more than 5000 requests daily to transport patients in emergency[2, 142]. In 2003, the cause of North East blackout at New York has been a programming error. The failures occurred when multiple systems tried to assess the same information at the same time and acquired the busy signals. Actually, the software could allow one system presidency at a time. Unfortunately, due to several requests at a time the software could not perform properly. The several incidences have forced the industry to develop reliable software. Though, it is not possible to develop 100 percent reliable software the developers and researchers have been making an effort to improve reliability of software under process.

Identification and fixing of software defects is important for improving the quality of the software. It is the most expensive activity in software development[15]. Software reliability has become an important component for software industry. With the growth in the demand of software, industry also advances in technology development, which makes software easily available to the society in minimum cost with faster speed. Software failures occur due to errors or bugs in the software at initial level of the design phase of the software. With the increased growth of software development and its easy availability to the user, failures have also increased in the same manner. Therefore, strict quality assurance is required for the execution of software to validate whether the software application behaves as required by the user[16]. Software quality is one of the important factors of software reliability. Software quality is affected by the development cost, delivery, schedules and user satisfaction. Software defect's cost billions of dollars for the industry[162]. Therefore,

these defects need to be identified and removed at early stage of software development to improve the reliability of the software[141].

1.2 Reliability

Software reliability plays important role in the development of software. Reliability is a key factor of quality, which quantifies the software failure. It answers the questions like "How well software meets its requirement" and can be defined as "The probability of failure free operation for a specified time in a specified environment"[34]. Reliability has the ability of maintaining a specified level of performance. It is the probability that a given software system has operated for some time without software error on the machine for which it was designed, within design limits[3]. Software and hardware reliability both have different mechanism and parameters to identify the cause of failures. Hardware faults are physical faults while in software it comes from design. Hardware faults are easily detected and corrected while software faults are hidden which are typical to identify, detect, correct and classify[4]. In 1976, Myers defines reliability as the probability that the software will execute for a particular period of time without failure, weighted by the cost to the user of each failure encountered[4].

1.2.1 Object Oriented Software Reliability

Software reliability is about constructing reliable software, software design and ensures that the software is reliable and provide training to software developers, architects and users about the usage of reliable software[7]. Software reliability may also be known as the cause of software failures. A failure is the inability of a system or component to perform a required function according to its specification. To improve the efficiency of software, various attributes are required to measure the reliability of the software. To develop reliable software, various efforts have been made by the researcher, practitioners and industrial professionals[5]. Earlier reliability was just limited to only knowing and understanding of the common faults or bugs. Reliability not only depends on product attributes and number of faults but also on the environment in which software is used during operation. A better way to integrate software reliability is to focus on the same during its development.

Software reliability is defined as "The probability for failure free operations of a program for a specified time under set of operating conditions"[5]. Reliability of software is assessed by testing the software product under simulated conditions. Software reliability covers methods, models and metrics to estimate and predict software reliability. Musa defines software reliability as "The probability of failure free operation of a computer program for a specified time in a specified environment" [6, 7]. Therefore, all the devices are based on software require software that is more reliable.

Design is the backbone of every system. Good software design is a design which can easily understandable, modifiable and testable. Object oriented concept is one of the most demanding and popular concepts for software development and industries. It supports inheritance, cohesion, coupling, polymorphism and encapsulation. The reliability of any system depends on the design defects of the system. These object oriented design constructs have a potential to mitigate defects of OOD. Object oriented design reliability can be maintained by implementing OOD constructs like inheritance, cohesion, coupling, encapsulation and polymorphism.

1.2.2 Need and Importance of Software Reliability

From literature survey it is found that various hazards have occurred claiming lives of people and assets because of error, faults, failure, defects and bugs in software. In 1985 and 1986, a lot of patient lost their lives because of software errors in Therac-25 radiation therapy machine[1]. One of the incidence happened in 1992 in London Ambulance service, provide the largest ambulance service in emergency situation to patient, broken down after its installation[1]. In aviation industry, every year several airline crashes due to misinterpretation and un-compatible work between software and pilot[1]. In 1990, a small fault in a switching system newly released network, and traced to software problems occurred during the summer[1]. All the above hazards occurred because of unreliable software and cause of defect in software. The object oriented design defects happened in Ariane 5 software reuse the specification from the Ariane 4, but the path defined in Ariane 5 was substantially different. Testing is not performed by the tester, which had never been performed on Ariane 5 flight and the result is shown in figure 1.2.2 Ariane 5 Disaster[5]. These hazards occurred because of reuse of code and applicable of OOPs concept without test the design of the software.

Figure 1.2.2: Ariane 5 Disaster

Design defects play an important role in software development lifecycle. High quality and reliable software is difficult to achieve in large and complex object oriented programs. These defects come from internal design of software and their effect on quality factors like understandability, maintainability, reusability and ease of implementation[9, 11]. These defects occurred at the design level are known as code smells[9, 12] or anti-patterns[13]. It is difficult to address every defect for software developer. Even a single defect in software made the whole system crash. It is difficult to maintain software after deliver to the customer. If defects are not exposed and mitigated during software design, can obtain huge cost in terms of time, money and effort after implementation. From the above software failure shows that a single defect may cause the huge lost and people lost their lives. Therefore, it is required that the defects should be mitigate at the design level for improving reliability of object oriented design.

1.3 Defects

The term ‘defect’ refers to something that is wrong with a program. It occurred in a form of misspelling, a punctuation mistake or an incorrect program statement. It is difficult to finding and fixing defects in software. Software defects can cause a range of problems, from minor defects to major defects. Primary responsibility of software engineer’s is to deliver quality products within specified time and cost. It also meet-out the user’s requirement and consistently do the user’s job. While the software functions are most important to the program’s users, these programs are not usable unless the software execute. To make the software reliable, however engineers should

remove all its defects. There are many factors of software reliability, first factor of software reliability referred with its design defects.

Defects are so important for developer that people made lot of mistakes. Even experienced programmers made a mistake in a small program/module. Generally, they find and remove these defects when they compiled and test their programs. They often still lot of defects in the final product. Some programmers refer these defects in form of error, bugs or failures. As we know that software systems are enhance to meet new requirements, potential problems can be display and small defect can become dangerous. While the vast majority of little defect have little effects. All the defects treated as important defects, not as negligible defects.

1.3.1 Design Defects

Design defect plays an important role in the software development life cycle. These defects come from poor internal organization design in object-oriented system and they are negatively effect on quality factors, maintainability, understandability and ease of evolution1[5, 17, 24]. Design defects usually known as code smells or anti-patterns[13, 165]. There is always a possibility about whether any components in a program are defective. Detection of design defects can considerably reduce the cost and time of subsequent activities in the software development and maintenance phases[155]. Several methods suggested to specify and detect defects. Software inspection[18] with software visualization[19] proposed manually detect design defects. However, this methodology leads to other several issues like time expensive, non-repeatable and non-scalable. Mantyla and Lassenius identified that when a certain software system increases, even experience developers capable to perform an objective system evaluation of design defects decreases[20]. Moha proposed to prevent the drawbacks of a purely manual detection approach[21]. According to Moha, "Design defects are bad solution" to recurring design problems in object oriented system[21].

Design defects come from bad design ranging from high-level design problems such as anti patterns[22, 29]. Design defects are bad solutions to recurring problems design, whose origins are from poor design practices and different defects "Difference from specification or requirements"[23, 30]. Software defects have a negative impact on the quality of object oriented systems and make it difficult to address debugging and development.

1.3.2 Object Oriented Design Defect

Object oriented design related with developing an object-oriented module of a software system for identified requirements. Designer used object-oriented to speedup the development process, module-based architecture, containing high reusable features and increase quality of design. *"OOD is a method of design covering the process of OO decomposing and a comment for depicting logical and physical as well as static and dynamic models of the system under design*[2]*"*

Designers can perform a good OO design by applying the OOD principles. If the designers know the cause and reasons of bad design then it assists them to avoid bad design. There are some reasons of bad design like change technology, complexity, lack of design knowledge and design practices. Technology is change on regular interval. Today it is the time of OOD, because these properties are available only in OOD provides flexibility for changing in previous design or existing design. But designers should be careful about the properties of OOD, which can make designing more complex like inheritance hierarchy. Designers are not able to use OOD in such a manner used in future, will not make the program complex. Usage of more method makes the program more complex. Martin[35] proposed four symptoms of bad design make the design decomposing. They are not relevant for the designing and used as rigidity, fragility, immobility and viscosity.

1.4 Relevant Issues

Today, internet, mobile and computer are the important role of every people in daily life. Everyday people perform his task by using computers and internet. Every person has a computer and deals with the use of software and services depend on them[25]. Business of different areas uses the internet for transaction of payment, purchase, sales and transportation of business with partners. From academic to health, business to medical everyone use the internet and computer and they are not alone. All these items run upon which they can belief. More dependency on the system makes the system more error prone and unreliable[36]. This makes the system more typical to understand and increase the no of defects of the system. Due to reliability many other episodes also occurred. Early stage of software development is necessary to detect defect and measure the reliability of the software.

Twenty first century is the revolution of object-oriented technology. Object oriented concepts have changed the viewed that used to assess the reliability of software. Object oriented technology applied various constructs like cohesion, coupling, inheritance, encapsulation and polymorphism[38, 143]. In design phase of the software development the programmer has an authority of using the coupling, cohesion, inheritance and polymorphism for the design of reliable software. High reliable software makes the system defect free, easy to understand.

Based on the discussion of the above problem there may be large set of research challenges that may need to focus. Some of the relevant issues discussed which are as follows:

- What are the factors that directly influence the reliability of an object oriented software design?
- Is design defect a reliability factor?
- Is there any standard framework available to develop reliability improvement of object-oriented design.
- Is there any OO design metric or model available, which can be used in the early phase of software development life cycle to measure its reliability?
- Can we develop an approach that may be used in the design phase to estimate the reliability of an object oriented software?
- Can effort of estimating the design defect of OOD be reduced by using good measurement tools?
- Is the proposed approach to quantify object oriented design defect help to compare the reliability of various alternative designs?
- Can a proposed approach be useful without assuring theoretical and empirical validity?
- How general are the lessons learned in this study? Can they be applied in situations involving other metrics and models, or to organizations, which have different operational contexts?

1.5 Research Problem

A design should be as simple as possible. Design is the main frame of the software. It is easy to incorporate reliability with design during software development life cycle (SDLC). Similarly, if the reliability features ignored, the final design of the software may be complex[35]. Object oriented software frequently modified during development or software evolution. Design defects are highly effect the reliability. If design defect increased, it affects reliability. These design defects can manage through the implementation of object oriented concepts. It has been use as one of the key factor for solving the software crisis problem[36]. Defects of software design can checked by software design constructs like inheritance, encapsulation, polymorphism, cohesion and coupling. High coupling and inheritance, increases the design defects of the software design and when cohesion increases, design defects decreased. Therefore, inheritance and coupling have a positive impact while cohesion has negative impact on software design. Reliability of the object-oriented design can increase by controlling the design defects.

It comes from the discussion that defect mitigation in the software design leads to the development of high reliability of final products[37]. Using metrics, defects may be measure and mitigate. A metrics based approach may assess defects and their expansion for object oriented design. With these approaches, reliability of the object oriented software design at early stage of software development can improved.

There is not any approach used for early stage of software development to measure the defect. It is highly demand to measure the design defects for object-oriented software used in the design phase. In view of that, the researcher has developed a problem in order to detect defect and estimate reliability of object oriented design by controlling the defects of software design. From critical review and found that there is not any framework to measure design defect to improve the reliability of the software design. Therefore, researcher has proposed framework can mitigate the design defects of object- oriented design under this title:

Improving Reliability of Object Oriented Design: Defect Mitigation Perspective

1.6 Objectives of the Research

To achieve the most general objective of working out of an approach for improving reliability of object-oriented design defects, following objectives are set away:

- To critically review and analyze the literature survey on software reliability, object oriented design constructs and object oriented design defects.
- To consider the demand, importance and implications of identifying reliability and design defects in the early stage of software development life cycle.
- To identify the factors which affect the reliability of object-oriented design.
- To study OOD constructs such as inheritance, cohesion and coupling to address design defects.
- To consider the relation between object oriented designs construct with object-oriented design defects.
- To measure which of the object-oriented design constructs has positive and which one has negative impact on design defect.
- To establish the relation between OOD defects and reliability.
- To develop a design defect mitigation methodology to be used in the design phase to estimate the rigidity of object oriented design and provide guidelines to improve the reliability of the object oriented design.
- To create a framework for improving reliability of the software for each of the identified object oriented design constructs.
- To develop design defect and reliability estimation model to validate the proposed framework and model.

1.7 Limitations

Every field have its own advantages and limitations. No work is perfect. In this research, design defects have taken as the main factor of software reliability. When the OOD defects increases reliability of software decreases. Object oriented design is broader area of research, so it is not possible to take all the aspects of object-oriented

design. In addition to successful completion of the research work, the study organized with the following limitations:

- The approach can only be applied to measure and mitigate OOD defects and to improve reliability of object oriented design.
- Due to unavailability of corporate data, the proposed research work is validated with a small set of data.
- The research focuses only on rigidity (a factor of reliability).

1.8 Thesis Outline

A thesis of the research has been prepared to reflect the detailed study on the research problem and previously mentioned research problems.

Chapter -1: Introduction

First chapter is the introduction part of the thesis. The chapter starts with the introduction of reliability and importance of reliability in software development. Some incidents discussed about cause of software failure in introduction; that needs to conduct the research on software reliability. Second part introduced about the design defects. The research question related to software reliability at design level. Then objectives of research are frame. At last, each research have its own limitation is discussed.

Chapter-2: Present Approaches

This chapter is associated with the presently existing approaches for mitigation of software defects and improving reliability of object-oriented software discussed. An elaborated review on reliability estimation model and design defect from last two decades presented. The detail review of some significant existing models from the year 2000 to 2014, introduced.

On the basis of review, it is recommended that reliability is affected inversely when design defect of software increases. Design defect identified as a key factor to reliability through literature survey and review. Therefore, there is requirement of reliability estimation by taking design defect into consideration. In addition, reliability improvement of an object-oriented design with object oriented design constructs is not address to design defect.

Chapter-3: Proposed Framework for Reliability Improvement

This chapter presents a framework for improving reliability of object-oriented design in respect of object oriented design defect. The chapter covers defect mitigation can control by the object-oriented design constructs like inheritance cohesion and coupling. The assumptions of the framework presented. A framework for reliability improvement of an object-oriented design is proposed. This framework consists of four phases including identification phase, mapping phase, measurement phase and improvement phase.

Chapter-4: Execution of the Framework

Implementation of the framework presented in this chapter. According to framework design defect is a key factor of reliability. In first phase of the framework, object oriented design defects, object oriented design constructs and reliability factors identified. In the second phase, mapping between object-oriented designs construct and design defect established. Finally, object oriented design defects and reliability is established.

To establish the relation, Depth of Inheritance Tree (DIT) used for Inheritance, Coupling between Objects (CBO) is used for coupling and Cohesion among Methods in Class (CAMC) used for cohesion. Rigidity Estimation Model (RiEM) and Reliability Estimation Model (ReEM) proposed to measure the design defect and reliability of object-oriented design.

Chapter-5: Validation of Framework

This chapter presents the theoretical and empirical validation of the proposed framework. As an experimental validation, pre tryout carried out on an object-oriented design. On the basis of reliability improvement framework, metrics and models values are computed according to the requirement. On the base of reliability improvement guidelines of object oriented design (^{RI}GOOD), design defect of the object oriented design is mitigate.

After redesign, the metrics and model values recomputed. It is verified that the DIT and CBO values are minimized and CAMC values are maximized. After dealing with ^{RI}GOOD on ReEM, the value of design defect found less than the values of

previous design. For the tryout purpose, ten object-oriented designs are accepted. The same process reproduced and is resolved that the approach performs well in this experiment also. Various statistical studies and hypothesis test carried out for the acceptability of the framework.

Chapter-6: Conclusion and Future Work

It is the last chapter of the thesis. In this chapter, major research findings along with the other findings presented. The research questions of the first chapter are treated one-by-one in this chapter. Significance of the research also discussed in details. Future plan, for extending the study are discussed.

CHAPTER 2: PRESENT APPROACHES

CHAPTER 2: PRESENT APPROACHES

"Without requirements and design, programming is the art of adding bugs to an empty text file."

-- Louis Srygley--

2.1 Background

Software is use by our modern society from last two-three decades. It is the requirement of every people in every aspect of human life. Software has become an essential part of business and life of the modern society. People used equipments in daily life i.e. wristwatch, hand held devices, home appliances, aircraft and automobiles are embedded with software. Nothing can operate without the use of software. It is a way of presentation and processing of human knowledge[25]. Today everything depends on software. When dependency of software increases, software failure are also increases in the same ratio. Due to huge no of people, depend on software so the reliability of software is required to develop. Therefore, reliable software execute only when it defined to suppose to do. It is still challenging problem that affects software producers from industry to customer[26]. Due to changes in design of software, it is difficult to address each problem. Most of the software developers have incomplete knowledge of reliability features and applied immature reliability methods for keeping reliability defects in the whole software development cycle. This may promotes more defects seeking in software. Due to these causes most of the industry applied object oriented concepts in software design.

With the revolution of software development, progress has done in a great height all around the world. Therefore, various other problems started with development. These problems are in form of economic loss and cause some hazards consequences. Software failures occur due to unreliable software. Due to increase of the design defect, software failure also increased with the execution of the program making the system more unreliable. The reliability of a system represented as the failure rate of the execution of the program[27]. Software reliability is important in aviation industry, spacecraft control, telecommunication, online banking and industry[28]. Software defects remain in the system, up to the deliver to the customer. There are many factors of software reliability. Design defect is one of the factors to mitigate the design defect and improve the reliability of the software.

There are many studies related to software maintenance, most of them based on software maintenance level, software coding during its development process or delivery of project, which consumes up to the 90% of the total cost of the project[31]. The objective of maintenance is to adding new functionalities, detecting design defects, correcting them and modifying the code to improve the quality of the project[32]. Sometimes design defects are unavoidable, it can be care by the development team and removed at the code level to avoid the failure of the software. There has been a lot of research work on the study of design defects called anti-patterns[9], smells[3] or anomalies[31]. Design defects made the system less reuse, less maintainability, high complexity and faulty action[33]. Sometimes the numbers of defects are typical to address in available resources. In most of the situation, developed software placed with hidden, known and unknown defects due to deficiency of development resources to deal with every defect. Therefore, it is required to detect defects and improve reliability of object-oriented design.

There are various research carried out on software reliability and design defects. In 2006, Haifeng Li et. al. proposed a framework for choosing software reliability metrics based on analytical hierarchy process and expert judgment[133]. In 2004, Cem Kaner proposed a framework for evaluating proposed metrics and applied it to count bugs. Bugs count a small part of attributes they are being used to measure. In 2001, Alan P. Wood et. al. describes many types of reliability metrics and their interrelationships both in theory and in practice. Reliability concept based on faults and failures. Definition of failure, define on one perspective to another[130].

In 2004, Pankaj Jalote suggested in his research on reliability measurement that once reliability is measure, manufacturer wish to know how to improve the reliability of the software[27]. In 2002, Bansiya et. al. introduced a hierarchical model for assessment of object oriented design quality[43]. In 1996, Alan Wood et. al. predicted software reliability by applying software reliability modeling to a subset of products for four major releases of software products. It is found that the correlation with a simple exponential model was good and model can reasonable predict the number of remainder defects in delivering software[134, 158]. In 1991, A. L. Reibman et. al. examined the role of reliability models in software design and discuss predicting the reliability of the software system method [135]. M. Thangranjan et. al. considers a reliability model which has been properly used for predicting the reliability of

software systems. It is based on Rayleigh equation related to the Weibull family of curves.

In 2007, M. C. Kim et. al introduced that there are some potentials for providing the maximum level of reliability in safety critical system by the use of software reliability growth models where all the essential software faults are identified and properly repaired[163]. The limitations of software reliability growth models are also discussed that cause of software failure for the critical sets[136, 157]. In 2006, I. Karnanta introduced a report in which author analyzes the existing software reliability models on basis of analysis model. It does not take application of complexity or test coverage into consideration[137]. In 1992, M. R. Liu et. al. showed that the use of existing software reliability models are more effectively than a combination models that integrate the results of single or component models[138].

2.2 Reliability Models

Software reliability analysis performed at various stages during the process of software engineering. Software reliability model defines the universal form of the dependency of the failure process[1]. There are various reliability models for evaluating the reliability of a given model. Software reliability is one of the important parameters of software quality and system dependency. Software and hardware reliability both have different mechanism and parameters to identify the cause of failure. Hardware faults are physical faults, while in software it comes from design. Hardware faults are easily identified and corrected while software faults are hidden and typical identify correct, detect and classify[96, 97].

Software reliability played an important role in the quantitative measurement of software quality. For improving reliability of software, various models have developed for software reliability estimation[149]. The first model proposed in 1972 by the Jelinski and Moranda[99].Reliability is the capability of the software product to maintain a specified level of performance when used under specified conditions defined by the ISO 9126. It is not easy to estimate software reliability, because there are number of reliability estimation model that can be used to test and analysis of failure data. Figure 2.2 shows the hierarchy of software reliability models.

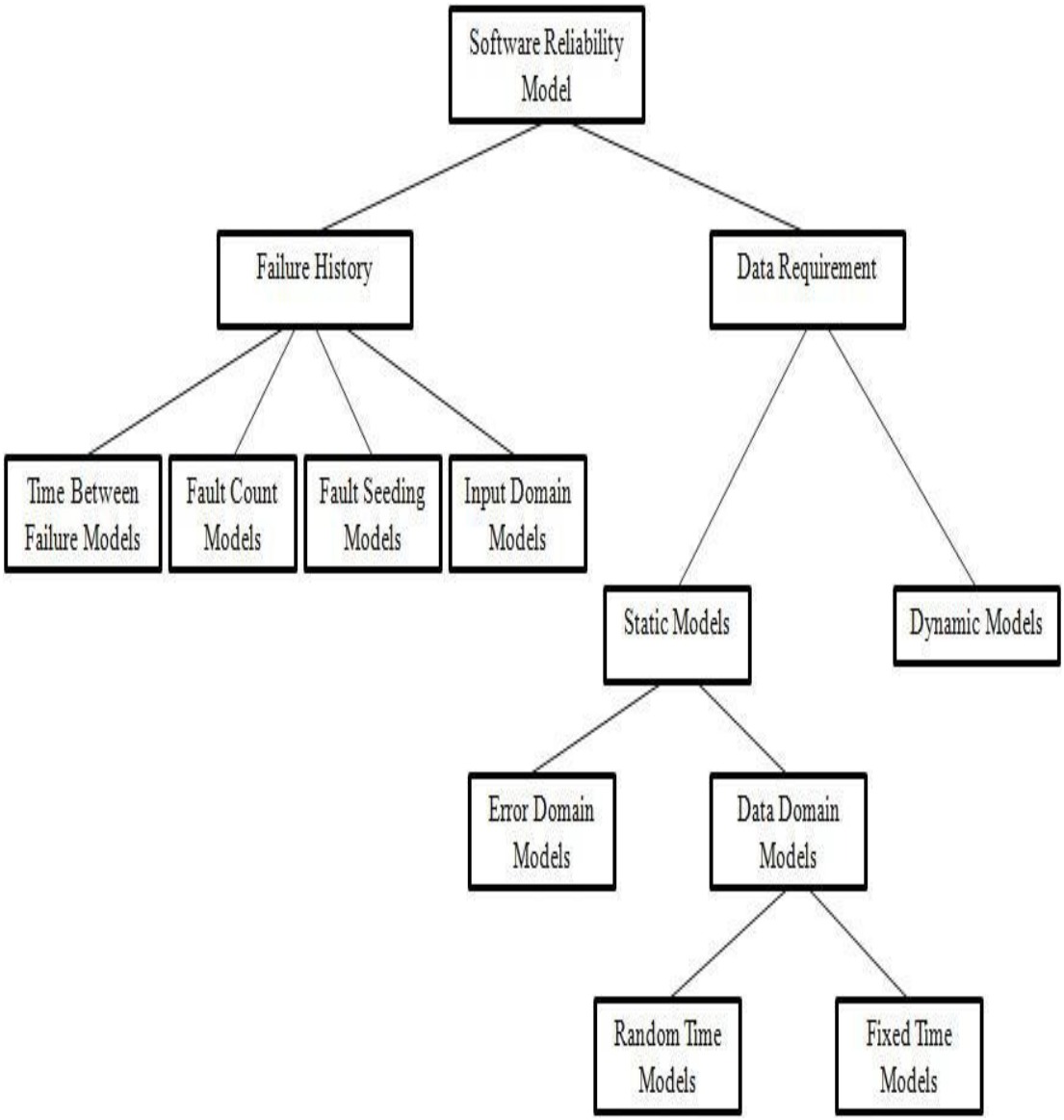

Figure 2.2: Hierarchy of Software Reliability Models

There are six dimensions which are consider for software reliability models are presented at initial stage, which is based on failure history and data requirements. Failure history categorize into four parts on its failure data: Time Between Failure, Fault Count, Fault Seeding and Input domain Models. When data analyzed, it is divided into two parts static model and dynamic model. Further static model is divided into two parts Error Domain Model and Data Domain Model. Data domain is divided into two parts random time Model and Fixed time Model[98].

2.2.1 Time Between Failure (TBF) Model

It is one of the earliest model for software reliability estimation. Consecutive failures in the system for longer period cause failures of the system. The data which are

considered for analysis is the time between i^{th} and $(i-1)^{th}$ failures. Fault data parameters estimated from observed values of the times between failures and estimates of software reliability, mean time to the next failure. Some of the pertinent models under this category are Jelisnki- Moranda (JM) model[150], Schick and Wolverton (SW) model, Goel and Okumoto (GO) Imperfect Debugging model and Littlewood Verrall (LV) Bayseian model.

2.2.2 Fault Count Model

In this model, the number of faults counted in assigned time interval rather than time between failures. When faults removed from the system, the number of observed failures will decrease. In this test, the time interval fixed in advance and the number of failure values treated as a random variable. Parameters of the failure rate estimated from the observed value of failure count or from failure times. Software reliability estimate means calculated from the next failure. Some models which are considered Musa Execution Time model, Goel-Okumoto Non-Homogeneous Poisson Process (NHPP) model, Goel Generalized Non-Homogeneous Poisson Process (NHPP) model, IBM Binomial and Poisson model, Shooman Exponential model, Generalized Poisson model and Musa-Okumoto Logarithmic Poisson Execution Time model.

2.2.3 Fault Seeding Model

In this model seeded a known number of faults in a program. It assumed to have an unknown number of indigenous faults in the program. Indigenous and seeded faults counted and identified during the testing. Using different methodology and models, the number of indigenous faults in the program and the reliability of the software estimated. In this category Mills Seeding Model implemented at the initial stage for seeding the faults, later Basin and Lipow are introduced related to some models on the base of seeding methodology.

2.2.4 Input Domain Model

The fundamental approach of the input domain is to generate the test case from an input domain. It is difficult to partition the input domain into equivalence classes. The reliability is measure from the number of failures or execution of the test cases. For input domain, Nelson, Ramamurthy and Basin model introduced.

2.3 Object Oriented Paradigms

The objective of object-oriented design is to build a class model holds the quality of the domain. Object oriented models are easy to explain and easy to modify as well as to deliver the customer. The major benefit of object-oriented design is to reuse, maintainability of software components. Components can be reused or modifying by using the separating of his facilities. The goal of industry is to provide reliable and quality software within stipulated time by implementing the object oriented design concepts. The mechanisms, which are used in object-oriented paradigm inheritance, cohesion, coupling, abstraction, encapsulation, polymorphism and information hiding. These are the important feature to applicable reuse and accomplish the maintainability of the software. These mechanisms are described as follows.

Inheritance: It is one of the essential features of the object oriented. Booch said that object-oriented concept is useless without inheritance[97]. Wegner said that a language, which is not supported inheritance, known as object based rather than object oriented. It is the relationship between two or more classes in which top class or existing class known as the super/root/base/parent class and newly defined class known as the child/subclass/derived class. A subclass develops its own methods or inherit feature from its super class. It represents "Is-A" relationship, so an instance of a sub class is also an instance of its super class. A subclass reacts for all the operation, which perform through super class. Inheritance has many forms of usage to inherit features when the inheritance executed.

Inheritance supports polymorphism. It moves in class hierarchy. To understand the class it has look at several places in hierarchy. As the super class contain a part of the sub-class, its consequences in a form of coupling between super class and sub class. Several problems has been defined in this reference such as rigidity problem by Seidewitz in 1996, fragile class problem by Pooley and Stevens in 1999 etc. Firesmith said that inheritance structure badly design are also expose the understandability, reusability and maintainability of the software.

Abstraction: Abstraction means to hide unnecessary information. The objective of abstraction used to simplify the specification of system by focusing only on details of leading parts while suppressing others. It helps in dealing the complexity of a system.

It increases readability and easy to understandability the system. It is the way of notion and view of the system. According to Booch "An abstraction represent the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide sharply" According to Budd, 2011, abstraction is divided into three different forms, specialization and multiple views. Division into parts is the application of principle of 'divide and conquer'. A huge system is divided into sub system and sub systems are further divided into sub systems, till the system is easy to reach and understand. In specialization, the system organized into a hierarchical structure in which layers at the lower levels need to be understanding layers at the high level. Views are the third form of abstraction. Each view represents different aspects of the system.

Encapsulation: It is a powerful tool to manage complex system. When we create object in an object-oriented language we can hide the complexity of the internal process of the object. What are the reasons to hide the internal complexity? It provides a simple and understandable way to use object without understand the internal complexity. It is use to manage changes. Today most of the vehicle engine works on gas, petrol or diesel. However the engine of the vehicles are used have alternative options to change the option. Each of the engines has different types of internal mechanism yet we are able to drive each of them, because internal complexity hidden from the end user. It means, mechanism motivates the car changes the system itself functions, the same way from the user views.

In object-oriented methodology, encapsulation means grouping of information with the methods that manipulate them. It is a language mechanism for restricting access to some of the class object's member (Benjamin, 2002). Encapsulation is a combination of grouping mechanism and protection mechanism (Scott, 2006). Encapsulations help in keeping the things belonging to an entity part together. Clients should access the services of the class through message passing. There is no need to know internal details of the class components. In 1972, Parnas introduced the information hiding. Encapsulation and information hiding both lead to a stable software design as changes localized and changes to a class do not affect the clients of the class. Software becomes flexible and can accommodate changes easily.

Polymorphism: It has many forms. It is the ability to associate more than one implementation with the same operation. It occurs in a form of operation with same name or image implemented in different ways in different classes. For the implementation of given operation, depends on the object class that contains the operation. The instance of the object sends the message does not have to affect about the receiver instance of the message. Same message interpreted in different ways depending upon the receiver instance of the message. An object oriented design is created with the help of polymorphism is easy to change and new methods are added to existing class without modifying them. In OOP, operations deployed in different ways depending upon the situation they are operating. It allows objects with different internal structures to have a common external interface. Polymorphism is particularly effective while implementing interface. The operation of a message with its implementation decided at compile-time or run-time. It classified into two categories: Adhoc Polymorphism and Universal Polymorphism.

Coupling: Coupling is used to measure the relative interdependency between various classes. It is used to establish a connection from one entity to another entity. Coupling is the degree to which the elements are connected. Classes depend upon another to provide functionality of the system. Two classes are tightly coupled, if they depend highly upon details of each other. A loosely coupled class is easy to understand and change because such classes do not require relating with many other classes. It is also easy to reuse such a class. Therefore, loosely coupling between classes improves the maintainability, reliability and reusability of the system.

In object-oriented approach, classes coupled in various types of relationships. These relationships described between classes by applying generalization, dependency and association. Generalization-specialization is a parent-child relationship in which the children inherit the structure and behavior from their parents. All classes in an inheritance hierarchy refer to properties of a single object. Dependency relationship creates a link between two classes in which one class uses the other as a parameter in the signature of an operation. An Association relationship creates a link between two classes as a unidirectional or bidirectional. In unidirectional, one class is referencing the other, whereas in the bidirectional association each class is referencing every other class in a bidirectional manner.

Cohesion: The representation of cohesion from reality to implement it to software designed with structure design methodology. It represents how tightly the internal elements of a module are bound to one another. The strongest form of cohesion is the functional cohesion, in which all the elements of a module related to one another to perform a single function in structure design, is the basic building block in object-oriented design. Representation of cohesion of a module extended with cohesion of a class. Cohesion in object-oriented terms denotes relatedness to the members of the class. A class with a degree of cohesion represents only one concept. However, a class with low cohesion performs many unrelated tasks. Such a class is difficult to maintain, understand and reuse.

Modularity: It is a way of managing complexity. It achieved by breaking a monolithic program into smaller and manageable components or modules. There is minimal interaction between the modules and maximal interaction within the modules. It is based on the divide and conquer concept. It reduces maintenance effort and increased reliability and reusability of the program. Therefore, the concept of modularity refers to a class as the basic unit.

2.4 RELIABILITY METRICS

Software metrics are used to monitor and check the progress of the project and also to identify and indicate the quality of the final product. Several metrics have been proposed by the various researchers for measuring the structural properties of an object oriented software product[156]. According to Rosner, 2006, object oriented metrics are used to mainly to understand the degree to which the concept of object oriented are realized in a system. Object oriented are applicable at class, package and system level. In this thesis, inheritance, cohesion and coupling are considering for measure the design defects.

2.4.1 Chidamber and Kemerer Object Oriented Metrics (1991)[39]

It is originally defined as the CK metrics also known as MOOSE (Metrics for Object Oriented Software Engineering) metrics. It consists of six metrics estimated for each class: WMC, DIT, NOC, CBO, RFC and LCOM1. It is amended by the authors RFC`, LCOM2, LCOM3 and LCOM4.

(a) ***Weighted Methods Per Class (WMC):*** This metrics measure the complexity of an individual class. The value of WMC has been found to lead more faults. The value of metrics is obtained by computing weighted sum of complexities of methods of a class. Two different weighting functions are considered: first uses the nominal weight of 1 for each method, the number of method only[86] and the uses for the cyclomatic complexity metrix (McCabe, 1976) to measure complexity of method (Li and Henry, 1993). It is a measure of size and equivalent to the number of methods in the class.

$$\text{WMC} = \sum_{i=0}^{n} Ci$$

Where n is the number of methods in a class and C_i is the complexity of class. In software development, it is advised to use small size of methods instead of large amount of methods. Smaller amount of methods will reduce program complexity and increase readability of program[39].

(b) ***Depth of Inheritance Tree (DIT):*** It is defined as the longest path from the class to the root of the inheritance tree. Inheritance in object oriented programming can be described as a hierarchical tree. A class in the lower level of the tree has more methods and variables to inherit from its upper class levels. The root of the inheritance tree inherits from not any class and is level at zero from the inheritance tree.

(c) ***Number of Children (NOC):*** It shows the number of immediate sub-class derived from a base class. It represents the width of the class hierarchy. High value of NOC indicates high reuse of the base class. If the base class is thoroughly tested, high reuse of the base class could reduce the chance of defect. However, high values may indicate an incompatible abstraction in the design as a class with a large number of children has to provide more generic service to support all the children. This represents to introduce more complexity into the parent class and also makes it fewer cohesive. A class with a huge NOC demand more testing, which consequence on many other classes in the system.

(d) ***Lack of Cohesion in Methods (LCOM):*** The purpose of this metrics is to measure the lack of cohesion in the methods of a class. It is used to count the number of non-similar methods pairs, in terms of attribute usage and similar method pairs.

Different attributes occurring in different methods of a class indicate that methods do not process related information. It is based on the principal that a variable occurring in many methods of a class causes that class to be less cohesive than others where the same variable is used in few methods. A class with low cohesion is also not fit for reuse.

(e) ***Coupling between Objects (CBO):*** It provides the number of other classes that are coupled to the other classes. A class is coupled to another class if it uses instances and member function of other class. Excessive no of class coupling restrain reuse of individual classes. Classes with a large number of classes are coupled very sensitive to change, which makes maintenance of the design very difficult. For such classes testing effort is also high.

(f) ***Response for a Class (RFC):*** It counts the number of occurrence of calls to other classes from a particular class. It gives the number of methods that can potentially be executed in response to a message received by an object of that class. The larger is the number of methods that can potentially be executed in response to a message received by an object of that class. The larger is the number of methods that could potentially respond to a message, the greater the complexity of the class.

2.4.2 Li and Henry's Metric (1993)[45]

Li and Henry amended the metrics[45], proposed by the Chidamber and Kemerer in 1991. They also proposed various other metrics and evaluate the relation between their metrics and their maintenance effort. They describe the object oriented metrics which are as follows:

(a) ***Number of Ancestor Classes (NAC) and Number of Descendent Classes (NDC):*** The extension of the inheritance related metrics given in the CK metrics suite: DIT and NOC. DIT measures the number of classes by the longest route to the root class of the hierarchy while NOC counts the number of immediate subclasses. NAC counts the number of relatives' classes. NAC and DIT will take same value in the case of single inheritance and different in case of multiple inheritances. The Number of Descendent Classes (NDC) metric counts the descendent classes in a hierarchy tree.

(b) ***Number of Local Methods (NLM) and Class Method Complexity (CMC):*** NLM counts the number of methods in a local class while CMC measures the combined complexity of all methods of a class.

(c) ***Coupling through Inheritance (CTI), Coupling through Abstraction (CTA) and Coupling through Message Passing (CTM):*** It elaborates in different forms of coupling between classes like coupling through inheritance, coupling through abstract data types and coupling through message passing. They indicate that coupling through inheritance can be measured by using NAC and NDC separately or both. He also suggested about two more metrics like CTA and CTM metrics.

(d) ***Size1 and Size2 Metrics:*** Both are used for measure of class size. Size1 of a class is used to measure the number of semicolons in the class while size2 of a class is used to measure the number of attributes and methods in the class.

2.4.3 Lorenz And Kidd Metrics[44]

It is also known as LK metrics[44]. LK proposed ten metrics. These metrics described give a fair cross section broad area of the design. As CK metrics, most of the LK metrics are direct metrics and include directly countable measures.

a. ***Number of Public Methods (PM):*** It simply counts the number of public methods in a class. As LK says that, this metric is useful to estimating the amount of work to develop a class or subsystem.

b. ***Number of Methods (NM):*** It counts the total number of public, private and public methods defined in the class. LK suggest this metric as a useful indication of the classes

c. ***Number of Public Variables per Class (NPV):*** It counts the number of public variables in class. LK considers the number of variables in a class to be one measure of its size. The fact that one class has more public variables than another might show that the class has more relationship with other objects. It behaves as a main class.

d. ***Number of Variables per Class (NV):*** It counts the total number of variables in a class. Lorenz and Kidd proposed said that the ratio of private and protected variable to the total number of variables indicate the effort required by that class for

providing information to other classes. Private and protected variables are therefore represent as data to service the methods in the class.

e. ***Number of Methods Inherited by a Subclass (NMI):*** It is used to measure the number of methods inherited by a subclass. It is not defined as to whether that inheritance is public or private. Whatever, is any class using methods from a subclass would not necessarily have access to all the inherited methods.

2.4.4 Abreu Metrics [40]

Abreu[40], proposed a set of six metrics related to design. The importance of this metrics is on the feature of inheritance, encapsulation and coupling. These six metrics are defined as follows:

a) ***Polymorphism Factor (PF):*** It is based on the number of overriding methods in a class as a ratio of the total possible number of overridden methods. Polymorphism comes from inheritance; and Abreu says that in few cases, overriding methods reduce complexity by increasing understandability and maintainability.

b) ***Coupling Factor (CF):*** It counts the number of inter-class communication. Abreu, considers coupling as increasing complexity and reducing both encapsulation and reuse to understandability and maintainability.

c) ***Method Hiding Factor (MHF):*** It is the ratio of hidden methods to total methods. It is proposed for measure of encapsulation.

d) ***Attribute Hiding Factor (AHF):*** It is the ratio of hidden attributes to total attributes. It is proposed as a measure of encapsulation.

e) ***Method Inheritance Factor (MIF):*** It counts the number of inherited methods as a ratio of total methods. There is a similarity with the NCR metrics of LK. Abreu, proposed MIF as a measure of inheritance and therefore as a means of expressing the level of reuse in a system. It could also claim to be as to assessment of testing requirements.

f) ***Attribute Inheritance Factor (AIF):*** It counts the number of inherited attributes as a ratio of total attributes. It is proposed for expressing the level of reuse

in a system. It is also claimed that excess use of reuse causes to reduce in understandability and testability.

2.4.5 Bansiya Metrics[42, 43]

Class level metrics are defined by the Baniya and Davis[42,43] which are defined as follows:

a) ***Data Access Metrics (DAM):*** It is the ratio of the number of private and protected attributes to the total number of attributes in the class.

b) ***Direct Class Coupling (DCC):*** It counts the different number of classes that a class is directly related. Metrics includes classes that are directly related by attributes declaration and message passing in methods.

c) ***Cohesion among Methods of Class (CAMC):*** It computes the relatedness among methods of a class based upon the parameters list of methods. It is computed using the summation of the intersection of parameters of a method with the maximum independent set of all parameter types in the class.

d) ***Measure of Aggregation (MOA):*** It counts the number of data declaration whose types are user defined classes.

e) ***Measure of Functional Abstraction (MFA):*** It is the ratio of the number of methods inherited by a class to the total number of methods accessible by member methods of class.

2.4.6 Tang Kao and Chen Metrics[41]

It is used to represent a set of new metrics as developed by Tang[41], derived from observation in studying CK metrics.

a) ***Inheritance Coupling (IC):*** It provides the number of parent class to which a given class is coupled. A class is coupled to its parent class if one of its inherited methods is functionally dependent on the new or redefined methods in the class. A class is coupled with its parent class if it satisfied with these conditions:

- If any inherited method uses a variable that is defined in a new method.

- If any inherited method calls a redefined method and uses the return value of the redefined method.

- If inherited method is called by a redefined method and uses the return value of the redefined method.

b) ***Coupling between Methods (CBM):*** It provides the total number of new methods to which all the inherited methods are coupled. An inherited method is coupled to a new method if it is functionally dependent on a new method in the class. Therefore, the new methods to which an inherited method is coupled can be measured. It measures the total number of function dependency relationship between the inherited methods and the new methods. As a matter of fact, this metric is the version of IC metrics. The objective behind this metric is that IC only measure the number of parent classes to which a given class is coupled without the CBM, additional function dependency complexity at the method level is not considered.

c) ***Number of Object/Memory Allocation (NOMA):*** It is used to measure the number of statements that allocate new objects or memories in a class. The objective behind this metric is that classes with large number of objects allocation statements tend to introduce additional complexity for memory management.

d) ***Average Method Complexity (AMC):*** It provides the average method size for each class. Pure virtual methods and inherited methods are not counted. The objective behind this metric is that a large method, which contains extra codes tend to introduce more faults than a small method.

2.5 Present Approaches

Software reliability is the primary stage of software development to mitigate defect and errors at design level. The efforts of researchers and practitioners are appreciable. There are various approaches aiming to improve the reliability of the software from requirement phase of the software until the delivery of the software. There are various approaches/ frameworks/metrics/tools for ensuring a reliable product, some pertinent and relevant approaches are briefly discussed are as follows:

2.5.1 Software Reliability Model with Time-Dependent Fault Detection and Fault Removal[145]

In 2016, Zhu and Pham develop a software reliability model to care fault-dependent detection, fault removal from maximum number of faults software. The general practice of the most researchers for software reliability growth models is that, fault is independent and can be removed completely after detection. But it is not fact due to various factors including software complexity, efficiency, management hierarchy etc. author used the genetic algorithm to estimate the model parameters [157]. To compare the goodness of fit for all the models, four major releases of software product at Tandem Computers. Eight NHPP models are reviewed and found G-O model provides the best solution in terms of goodness of fit[145].

2.5.2 Design of Software Fault Prediction Model Using BR Techniques[146]

In 2015, Mahajan, Gupta and R. K Bedi used Bayesian Regularization (BR) technique has been used to finding the software faults before the testing is applied. It helps to reduce the cost of software testing. The objective of BR technique is to minimize a combination of squared errors and weights, and then find the correct combination to produce an efficient network. The accuracy of BR algorithm based neural network has been compared with Levenberg Marqar (LM) algorithm and Back Propagation Algorithm (BPA) for finding the software defects[161]. Author found that BR techniques provide better result in compare to other two algorithms [146, 152].

2.5.3 Selecting Software Reliability growth models and Improving their Predictive Accuracy using Historical Projects Data[147]

In 2014, R. Rana et. al, proposed a software reliability growth model and improving their accuracy from previous data. Most of the organization has to focus on two important decisions are: how to allocate testing resource and when the software is ready to release. Software reliability growth models (SRGMs) provide observational report and predicting reliability of software systems. More than hundred models for software reliability have been proposed and measured, but today there is no clear guide on which models should be used for a given software development process or for a particular domain. Author used SRGMs for their ability to provide empirical basis for making decisions. Using defect intensity growth rate from earlier projects

increase the accuracy of the prediction[160]. Results show that the Logistic and Gompertz models provide the accurate result. Author observes that classifying a given project based on its expected shape of defect flow help to select the most appropriate result[147, 153].

2.5.4 Software Reliability Modeling using Fault Tree Analysis and Stochastic Petri Nets[132]

In 2013, Khandelwal and Rath proposed a software reliability modeling using fault tree analysis and stochastic Petri Nets. Today, software used by every people in every aspect. Most of the failures occurred due to software failure instead of hardware failure. These faults are either removed by debugging them when they encounter or through preventive measure. Most of these developed in heterogeneous manner rather than homogeneous which makes it difficult to define the performance of the system from input and output. Author performed in two different methods to model the software system. Reliability is predicted using fault tree analysis. The stochastic petri net is elaborated to show how the SPN overcomes the problem of fault tree. The stochastic petri net can be used to model the dynamic behavior of the system and reliability is predicted using the failure rate of different components[132].

2.5.5 Reliability Estimation Framework for Complexity Perspective[143]

In 2012, A. Yadav and khan proposed the reliability estimation framework to minimize the complexity of object oriented design. In this framework, the object oriented design constructs correlates with complexity and complexity with reliability. There are various approaches to make the system reliable. Object oriented design is one of the important approaches to estimate reliability in respect of complexity. The proposed framework by the author consists of five phases. Each phase is integrated with each other. This framework performs as a bridge between object oriented design constructs, complexity and reliability[56]. Framework measures and minimizes the complexity of software design at the early stage of software development life cycle which contributes a reliable product to produce[94]. Review and revision is the additional phase of the proposed framework. Final suggestions from the expert and review are incorporated in this framework[143].

2.5.6 Role of Software Reliability in Performance Improvement and Management[131]

In 2012, P. Sashikala describes the failure behavior of the software. It assesses the reliability of software by predicting faults or failures for a software. The author explores the relationship that exists between quality attributes and performance attributes. The derived knowledge helps in improving the performance of the software over a period of time and handles the software more effectively. The performance of software has to be managed to deliver best performance. Two methods are used to manage software performance (i) Reactive performance method focus on actions or remedies that will be taken once the software performance problem is encountered. (ii) Proactive performance management anticipates potential software performance problems and steps to detect removal of those problems early in process. Author proposed guidelines in proactive performance management can be adopted for good results in reliability and performance by combining best practices in both the fields[131].

2.5.7 Software Reliability Growth Model for Genetic Programming[125]

In 2011, Zainab Al-Rahamneh et. al. have been presented a variety of software reliability growth models with the use of genetic programming. It is the collection of methods for the automatic generation of computer programs that solved specified problems. Genetic algorithm and genetic programming are different only in form of represent the solution. In GP, the population of solution is not a fixed length character strings which encode possible solutions to the problem in a form of computer program represent in a tree structure.

From literature survey it is found that software reliability growth models are existed but they have to face many problems when they are used in projects. Author used genetic programming to create software reliability growth model which can predict to increased faults during the software testing process. Author compared the proposed SRGM based on genetic programming with the performance of other common growth models. Experiment shows that proposed GP model is better than the other (Yamada S-shaped, Generalized Poission, NHPP and Schneidwind) reliability models[125].

2.5.8 Estimation of the Reliability of Distributed Applications[127]

In 2010, M. Cristescu et. al. proposed reliability as a important characteristics for use in crisis distributed applications. Distributed systems are the basis of the next generation communication systems. Distributed systems consist of processes running in parallel on heterogeneous platforms, are prone to conditions of "stress communication errors, failures in communication nodes and bottlenecks". A reliable distributed application is defined as a system whose behavior is predictable, despite failures and reconfiguration. Distributed systems provide several benefits such as it serve the global business and social environment in which humans live and work. Distributed systems can improve the quality of services, in terms of reliability, availability and performance of the complex systems. The study represented by several distributed applications made under the object oriented programming techniques. The objective is to estimate the reliability of these applications using object oriented design metric validation techniques[127, 150].

2.5.9 Software Reliability Prediction and Estimation[126]

Software reliability models are used for the prediction and estimation of software reliability. Author proposed an approach for software reliability model selection based on experiences from history software projects. It is accepted that the current software project is similar with a history of software project. Author develop a prototype consist of three function modules data processing, selection and prediction of software reliability facts. It has some characteristics which is as follows (i) It is available for windows operation system and user friendly. (ii) It provides more than ten software reliability models including software reliability growth models. (iii) It provides the model selection which can help users[126].

2.5.10 Reliability Improvement Predictive Approach to software testing with Bayesian Method[128]

In 2010, B. Cheng-Gang et al used Bayesian method for reliability improvement predictive approach to software testing. The objective of software testing is to improve the software reliability. From literature survey it is found that none of the existing testing methods pay a lot attention to improve software testing strategy based on software reliability improvement. The reliability has been predicted on the basis of

software testing strategy which improves the reliability online. The proposed reliability improvement predicted approach to software testing with Bayesian method can optimize test allotment through online. The author has taken case study and find that software testing method filters more defects to increase the reliability of the system and the case study indicate that the proposed approach can get better results in the sense of improving reliability than random testing[128].

2.5.11 Object Oriented Design Taxonomy[129]

In 2008, Jonathan et al discuss in his thesis "An Object Oriented Design Taxonomy" works on design patterns and principles. These design patterns and principles are captured during the software development process that may be beneficial to software developers. Design can be used before the system is built and while the system is being built. It also required when the system maintenance is incorporated on an existing system. The design provides an insight into the current state of the art in the object oriented design area. Researcher cover a large area of topics starting with an overview of software design, look into the various articles and decisions that occur inside the process and an overview of the various consequences and methods involved in learning to design software. Author presented two main contributions in his research. Primary contribution is object oriented design taxonomy in the area of commerce and the secondary contribution is a performance of the knowledge gap between the novice and expert developers using the commerce design taxonomy[129].

2.5.12 Framework for Selecting Software Reliability Metrics[130]

In 2006, Haifeng Li et al proposed a framework for selecting software reliability metrics. It is based on Analytical Hierarchy Process (AHP) and expert judgment. It is a multi attribute decision making theory, which provides valued evidence to decision maker by measuring expert judgment. It is a series of technological attempts which apply the experience and brainstorming of information. The proposed framework is to solve how software reliability metrics can be selected for each of the phases in software development cycle. Researcher selected 30 metrics and identified five criteria and selected five experts of software reliability field for the study. The metrics whose weights are high among other metrics in each phase are top ranked metrics like requirement phase "Fault density", "Test Coverage" and "Error Indices" are the top ranked metrics. On the basis of selection method in special metrics are analyzed to

explain the selection results. On the basis of result analysis consistency of comparison metrics and the sensitivity of the method are well accepted[130].

2.6 Relevant Findings

After a detailed study of various approaches of the object oriented software reliability, the following implications are drawn.

- Thorough study on software reliability models has been done two three decades.
- Present software reliability metrics has been reviewed thoroughly for last twenty years.
- Existing approaches of reliability models do not consider defect into account. Most of the researcher consider defect at coding level or after deployment and maintenance of the software. Therefore, reliability improvement in the design phase is important to mitigate the defects.
- Reliability estimation can be done through object oriented design constructs.
- On the basis of literature survey it is not found that design defect is a factor of software reliability yet as well as reliability is not quantify in terms of design defect of object oriented design.
- There is a gap between object oriented design constructs and design defects. The gap needs to be focused.

2.7 Conclusion

The increase of design defect makes reliability of object oriented design more difficult. In addition, most software defects have lack of knowledge of reliability views and use immature reliability methods for keeping safe from design defects for the whole software development life cycle (SDLC). This increase the demand to make software more reliable and defects are identified at early stage of software development life cycle. The general problem of with the reviewed models is that none of them allowed for non-existing failures data that is software usage history with a known duration of time in operational use with no detected failures.

In reliability growth models, the reliability of one version of a program is predicted from the reliabilities of previous version. While many of the models have been discussed in the following section bit they are completely incompatible because researchers want to estimate reliability based upon object oriented constructs like inheritance, cohesion, coupling and polymorphism. Software reliability growth models are incompatible because in growth models the improvement of reliability is measured with each successive modifications of the software. Growth model anticipate that trials from previous version are relevant to the reliability of the final version. If no failures are identified, the reliability predictions are not meaningful because they are based on values of the model parameters, which should be estimated[159].

Various reliability growth models can produce very different results for the same information. Most of the reliability growth models estimate the reliability when no failures are identified in testing because they cannot integrate information about the number of tests successfully executed. The identified approaches have its own limitations. Although, most of the approaches are quantitative but almost most of them are based on failures approaches. No research is identified for defect mitigation of object oriented design. Therefore, there is requirement to develop a reliability improvement framework that mitigates defects of object oriented design software at design level.

CHAPTER 3: RELIABILITY IMPROVEMENT FRAMEWORK

CHAPTER 3: RELIABILITY IMPROVEMENT RAMEWORK

"The perfect kind of architecture decision is the one which never has to be made"

— Robert C. Martin-

3.1 Background

Today, industries and organization extend their business all over the world by using internet. On the other hand, industries and organizations also collect information about various failures occurred due to software defects. The development of quality software, still leftovers is a serious matter for software organization. It requires proper guidelines, experience, practices and undocumented expert knowledge. Due to highly competition environment in software industry, customer pressure causes companies move faster and faster speed to deliver the products[46]. Software defects are the root causes of software failures[82]. Many studies found that, software maintenance traditionally defined as; any modification made on software code during its development process or after its delivery, consumes more than 90% cost of the whole software project[31]. While design defects are sometimes unavoidable, either they prevented or removed by the development team from the code level as early as possible. Software does not have fixed size, destroy, age, wear-out, color and weight. Software cannot be seen or touch. Software can also have small defect can accomplish in a form of tragedy[47].

There are many aspects of software quality; quality concern must necessarily with its defects. The job of software developer is to deliver quality products to the customer in specified time and cost. Software products must also meet the customer needs with reliable and consistent job. To execute the reliable software, developers should remove almost all the defects. Defects are so important, while people are doing lot of mistakes. Even experienced developers typically do a mistake for every ten-to-twenty lines of code. Although most of the defects corrected, after compile their programs. They often still lot of defects left in the final product. When the same programs used at global level in various ways that the designer cannot anticipate, obviously some defects can have unpredictable consequences. As widely used software systems enhanced to meet new needs, source of problems can be exposed and a petty defect can be truly become very dangerous. Since, there is no way to know the level of mistakes and its consequences on the software.

Software reliability is a key characteristic for software quality and it is an essential factor to measure software failures[95, 145]. Detecting and fixing of these defects is difficult to understand, time-consuming and manual process[48]. Sometimes the number of defects typically exceeds the resources available to address them. In many cases, matured projects forced to shift with both known and unknown defects due to lack of testing resources to deal with every project. Fixing of the problems may not sure make the software reliable. In 1991, changing of a three lines of code in a program contains millions lines of code, telephone of the whole California city along the eastern seaboard stopped communication[47]. Mozilla programmers claimed, "Everyday, almost 300 bugs and defects appeared which can only handle by the Mozilla programmers". Once the software is properly working, may also break[35]. Around, 55 million people mainly in the North Eastern United States, one of the biggest power blackouts in history. The causes of this blackout were nothing to do with a software bug[25]. In 2016, the disaster of a 25-year old satellite is fail because of software bug that lasted for a 13 microseconds[47]. The above incidents happened due to unreliable software. Therefore, software plays a vital role to save the life of people.

From last four decades, more than 200 models developed by the various researchers to quantify software reliability still, remains largely unsolved. Software reliability published as people try to understand the characteristics of how and why software fails. Software failures surrounding in between three major factors i.e. time, failures and process environment. Figure 3.1 shows the relation between error, fault and failures. In view of software reliability, error is a computer programmer action that comes in a form of fault. A fault is a software defect that causes failure. Failure life cycle is a pattern, which goes through during its life-time. It begins with defect identification and finishes when a defect removed. The term defect refers to something that is wrong with a program[49]. It could be a misspelling, a punctuation mistake, or incorrect statement line in a program. Defects can be in programs, in designs or even in the requirements, specifications or other documents. Defects may be in the form of redundant data or extra statements, incorrect statements or extra statements in program. A defect, in fact is anything that reduces the programs ability to complete and effectively meet-out the user requirement. According to Thomas Muller *"A human being can make an error, which produces a defect in the code, in*

software or a system, or in a document. If error in code is executed, the system will fail to do what it should do (or do something it shouldn't), causing a failure."

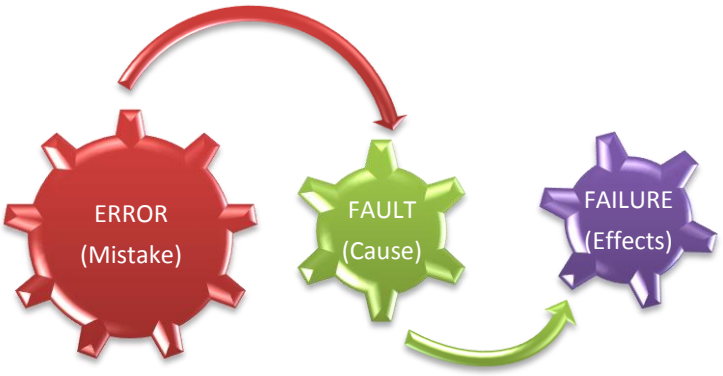

Figure 3.1: Failure Life Cycle

There are several approaches to makes the system more reliable. Object oriented is one of the important approaches to improve reliability in terms of OOD defects. It is one of the modern approach and popular concept in today's software development environment. Object oriented design is itself as an important tool for solving most of the software development problems[50]. It provides the functionality to binds the data and methods to prevent by unauthorized access. It also supports design constructs like inheritance, cohesion, coupling and encapsulation[51].

A class data members and methods encapsulated and communicate only through the interface provided by encapsulation; whether it is public, private or protected. Maintainability and reusability of classes are the main features of object-oriented approach. Due to these object oriented design constructs; it is easy to design a class hierarchy, easy to make a design structure and easy to understand. These features developed by using the concept of OOD constructs by minimizing or maximizing the values according to requirement of users[52]. Quality software can be developed which is good in quality, easy to use, less cost, less effort and reliable. Further, with these issues the researcher proposed a framework for improving reliability of object oriented design.

3.2 Reliability Improvement Framework

Software reliability improvement is an activity that focused on current reliability models and improved by applying statistical implications techniques to identify cause of failure, defect or error during system testing or execution of the program. Reliability improvement framework of an OOD believes that measurement is a tool for measuring the effectiveness of any estimation activity[53]. Most of the analytical methods of reliability standards based on the Markovian models and exponential failure time distribution[54]. The measurement models also called software reliability models. Most of the software reliability models and metrics discussed. Reliability improvement framework integrates the measurement phase along with other phase of reliability improvement.

3.2.1 Premises

A framework is a hypothetical description of a complex process. It provides actual base for future research. The framework for reliability improvement of OOD has the following assumptions:

- Reliability of an OOD is directly or indirectly affect by various factors, in which design defects taken as a major factor.
- Researcher can choose any of the reliability factors to improve the reliability of object oriented design.
- Design defects of object-oriented design is affect by object oriented design constructs such as cohesion, coupling and inheritance.
- Reliability of an object oriented design is affected by design defects.
- The framework improves the reliability by controlling the design defects of object-oriented design.

3.2.2 Proposed Framework

The proposed framework for reliability improvement of OOD presented in figure 3.2.2. The proposed framework consists of four phases.

- Identification Phase
- Mapping Phase
- Measurement Phase
- Improvement Phase

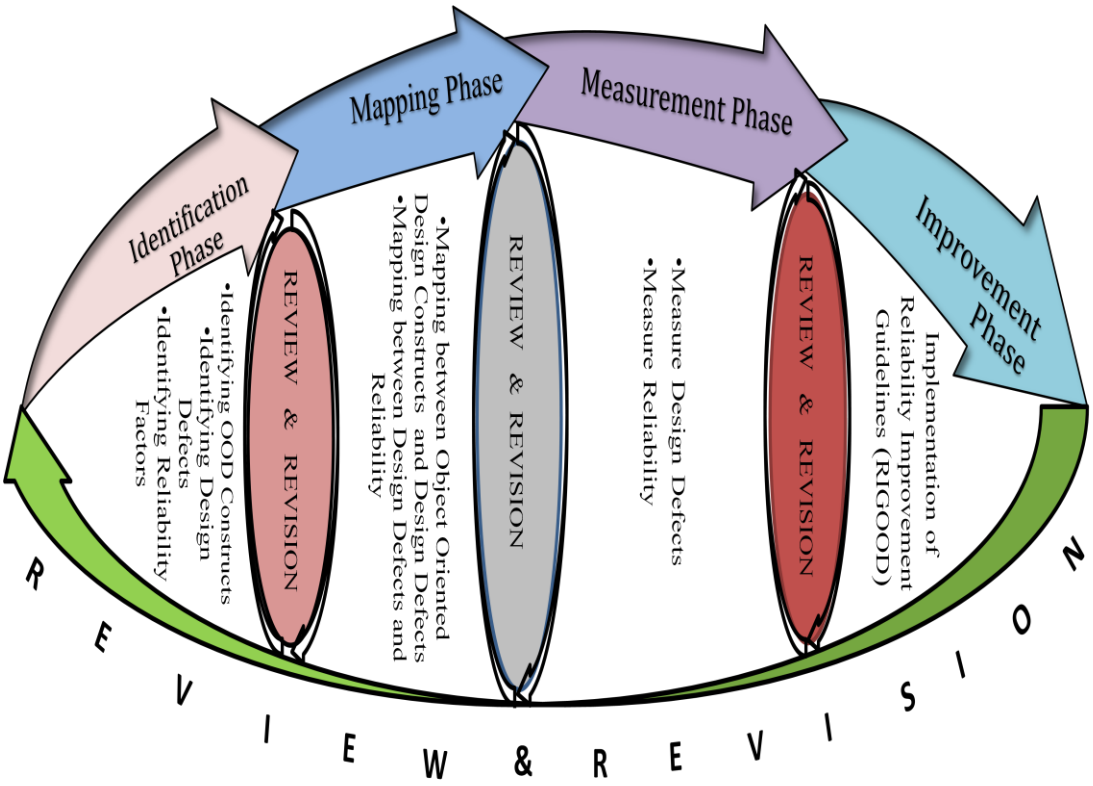

Figure 3.2.2: Framework for Reliability Improvement of OOD

In the identification phase, the relevant object oriented design constructs, reliability factors and design defects used as a key factor for identifying reliability. Second phase is the mapping phase; where mapping between object oriented design constructs with design defects; and mapping between design defects and reliability have been established. In measurement phase, object oriented design metrics; Depth of Inheritance Tree (DIT) for Inheritance, Cohesion among Methods in a Class (CAMC) for Cohesion and Coupling between Objects (CBO) for Coupling are used. Rigidity Estimation Model (RiEM) and Reliability Estimation Model (ReEM) proposed for design defect and reliability computation. Reliability is measure to mitigate the design defect, with the use of reliability estimation model. Improvement phase is the last phase for analysis of object oriented design metrics. Reliability

improvement guidelines, validation of suggestive measures and finalization of framework of object-oriented design are proposed. Finally, on the basis of review, the whole system is revised.

(i) Identification Phase

The objective of reliability improvement framework is to estimate reliability of an object-oriented design by controlling object oriented design constructs in related to the design defects. It is classified into three phases:

- Identifying Object Oriented Design Constructs
- Identifying Reliability Factors
- Identifying Design Defect

Identifying Object Oriented Design Constructs: The objective of object oriented design is to overcome the problem of procedure oriented programming. Procedure oriented programming provides a conventional approach to solve the problem in a sequence manner. The fundamental job of procedural programming is the function of the programming and then focused on the data. When multi-function is used, data are used as global level so that they may be assessed by multiple users and functions. Multiple users and global data are more error prone where occurrences of defects are more in function. Procedural approach does not solve the problem of real world[55, 140]. These problems solved through object-oriented concept. Object oriented design deals with the data as crucial element in the program development and does not allow it to flow freely around the system. There are various object oriented design constructs for OOD such as inheritance, cohesion, coupling, polymorphism, abstraction and encapsulation. Researcher identified three important design constructs used as follows:

- Inheritance
- Cohesion
- Coupling

The modern approach of object-oriented design protects data from unexpected changes from external functions through the encapsulation. Data can inherit in class of hierarchy from root to child. Cohesion, coupling, data binding, polymorphism, abstraction, dynamic binding, message passing and inheritance are some of the approaches used in object oriented[56]. To establish a relation between object oriented design constructs and design defects; inheritance, cohesion and coupling used as a major construct.

Identifying Reliability Factors: Software reliability is the essential factor to estimate the quality of software. The reliability of these models based on the reliability of the collected data. Software reliability is defined in IEEE as "The probability of a software system or component to perform its intended function under the specified operational conditions over the specified period-of-time"[5]. The attributes of reliability are examined from critical review on McCall's, ISO 9126-2000, Dromey's and Boehm's quality factors[57, 59]. Researcher has identified the major reliability factors are as follows: Maintainability, Error, Fault, Failure, Defect, Complexity, Maturity, Flexibility, Consistency, Correctness, Simplicity, Reusability, Recoverability, Testability and Accuracy. These identified factors affect the reliability either directly or indirectly[56].

Identifying Design Defects: Design defects are one of the important factor that decide the time of shipping of software. Design defect also affected the reliability of data as well as system. From the above identified reliability factors, it is found from critical review and literature survey, that design defect makes the system unreliable[47]. Design defects have negative impact on software reliability. If design defect increases, reliability of the system decreases. Consequently, design defects taken as a major factor for improving software reliability of object-oriented design[55].

Designers perform a good object oriented by following the OO design principles. If designers know the reason and symptoms of bad design, then it is useful for them to prevent the bad design. There are some fundamental reasons for bad design like changing technology, expertise, lack of design skills and design practice. Now it is the time of OOD, that various properties of OOD support the changes without modifying the early or present modules. But, designer should always be careful about some characteristics of OOD, which make the design difficult. Martin proposes various

primary symptoms whether designs are rotting[116, 117]. They are rigidity, fragility, immobility, needless complexity, opacity and viscosity.

- **Rigidity:** The concept of rigidity is the design change in simple way; the entire design will be change. The design is hard to change because every alter forces to other changes to the same part or other part of the system.
- **Fragility:** The idea of fragility is that changes cause new bugs i.e. the design is easy to break. Alter cause the system to break in places that have no conceptual relationship to the part that was changed.
- **Immobility:** It means unsuccessful to reuse software from different or same design. It is hard to untangle the system into components that can be reused in other systems.
- **Needless Complexity:** When it contains code, is currently not useful. It usually happens when developers try to predict future requirements and changes put into the code parts that do not have to be there at the moment.
- **Opacity:** It is a tendency of software code to be difficult to understand. The constant refactoring is needed in this case to prevent this from happening.
- **Viscosity:** It is difficult to things is harder than doing things wrong. It is hard to the right thing because sometimes is just easier to do.

(ii) Mapping Phase

To develop relation between two or more variables in such a way that organized changes in the value of one variable came with changes in the other values. These changes are calculated by using statistical representing how closely two variables are dependent and independent. The value of relationship between different variables lies between +1, 0 and -1. These values represent +1 (positive correlation), o represent (no relation) and -1 represent (negative correlation). This phase of framework to estimate reliability of object oriented design defects in two stages:

- Mapping between OO Design Constructs and Design Defect
- Mapping between Design Defect and Reliability

Mapping between OO Design Constructs and Design Defects: Today object oriented design based software is a demand for all the industry, aviation, education, medical, traffic, banking and railway. For the development of mapping between object oriented design constructs and design defects. Inheritance, cohesion and coupling are the most important object oriented design constructs. In order to establish a relation between design constructs and design defects through critical review; the effects of object oriented design constructs are reflected on design defects[53, 58, 59]. Object oriented design constructs have an affects on the design defect. When the design defect increases, which increase the misunderstanding and thus introduce more faults and failures with the design that decrease the reliability of the object oriented designs[100].

Object oriented design constructs such as inheritance and coupling are high make the system harder to understand and increase the defect of the design58. Inheritance and coupling have a positive impact on design defect while cohesion has a negative impact on object oriented design. This resolves that inheritance and coupling increases in the object oriented design, increases the design defect in the design. Similarly, cohesion increases in design, decrease the design defect in the design. Hence, object oriented design constructs inheritance, cohesion and coupling of the object oriented design affects design defect either positively or negatively.

Mapping between Design Defects and Reliability: Reliability and design defects are closely related with each other. Design defect has been taken as a major factor of software reliability in identification phase. According to definition "Reliability is the probability of failure free operation for a specified time in a specified environment". Software reliability theory is based on the concept of failure and design defects. Software design is started at the early stage of software development. If defects are identified and removed from the software, then reliability of the software may improve. It is difficult to achieve a certain level of reliability, if a system executes with a single defect in software design. Defect is reciprocally related to the software reliability. Software reliability growth models are used to detect and explain the cause of failures and faults of software system. The cause of software faults are the occurrence of the unpredictable events[94]. If faults are found and fixed, it is presumed to be increase the reliability of the software. Here, reliability will be considered in reference to the design defects of object oriented design constructs.

(iii) Measurement Phase

When a process or characteristics does not perform within its specification, it is treated as defect[60]. During the process of measurement, it will be misleading if you do only data collection work. Numbers of defect cannot be used to control things. The numbers must properly describe the process being assured, and they must be sufficiently well defined and verified to provide a reliable basis for action[61]. To measure the design defect and mitigate the defects in object oriented design that improves the reliability of the software. It is very difficult and expensive to correct the design after coding of the software. The measurement phase contains the following step:

- Measure Rigidity using Rigidity Estimation Model (RiEM)
- Measure Reliability using Reliability Estimation Model (ReEM)

Measure Rigidity using Rigidity Estimation Model (RiEM): To establish mapping between object oriented design constructs and rigidity design defect, the influence of design constructs on rigidity design defect examined by measuring the value of rigidity estimation model. It observed that each of the design constructs affects rigidity defect and rigidity defect affect the reliability of the design. Literature survey shows that object oriented design construct either increase or decrease the value of rigidity design defect[75]. To measure the value of rigidity design defect, researcher proposed Rigidity Estimation Model (RiEM). Object oriented design constructs such as inheritance and coupling increase the value of rigidity defect while cohesion decrease the value of rigidity defect[75]. With the help of identified metrics of inheritance (DIT), coupling (CBO) and cohesion (CAMC), the proposed rigidity estimation model are evaluated. Rigidity design defect is completely depends on these object oriented design constructs. A multiple correlation is established to develop the rigidity estimation model (RiEM) for the design.

Measure Defects using Reliability Estimation Model (ReEM): Software reliability covers methods, models and metrics to estimate and predict software reliability[62]. In most of the cases defects derived from the reliability estimate, but often the failure intensity is used as the parameter in the reliability model. Software reliability computation is difficult to quantify directly by knowing its related factors. Software

reliability is influence by several factors in which design defect takes as a key factor to estimate reliability of software. None of the reliability estimation model is presented in the literature in respect of design defects. Reliability estimation model (ReEM) estimates the approximation value of reliability for an Object Oriented design. Design defects negatively affect reliability. As design defect increases reliability of the design decreases and vice versa. Presently reliability estimation model will compute reliability against a case study of an object oriented design. Proposed reliability estimation model is used to quantify the reliability.

(iv) Improvement Phase

It is the last phase of the framework. In this phase, the framework has been improved by mitigating the design defects. What are the suggestive measures proposed in order to mitigate the design defects? For this, the improvement phase proposed guidelines for object-oriented design are as follows:

Reliability Improvement Guidelines: Reliability improvement guidelines for object oriented design (^{RI}GOOD) in reference to the defects have been developed. Based on reliability improvement guidelines improvement phase implements the suggestive measures and changes in object oriented design constructs. Reliability improvement guidelines are the preventive measures that are taken to protect the design against failure occurrence and design defects. It is used to apply for mitigation of design defect and improvement reliability in the object oriented software. The guidelines will help to calculate the defect value and adjust the object oriented design constructs to improve reliability of the software and mitigate the design defects of the software design.

3.3 Review and Revision

Review and revision are the additional phase of the reliability improvement framework of object oriented design defects. In this phase, all the phases of framework are examined and altered. Final description and improvements are incorporated in this phase. This phase is free to enter any phase of reliability improvement framework of design defects. Review and revision alteration can be done at any phase of software reliability estimation and the changes occurred at any stage of framework may implement during the examination and alteration.

3.4 Framework Significance

The framework has the following significance:

- Framework is useful to minimize the design defects of software design at the early stage of software development.
- It may also help to the effect of object oriented design constructs over the object oriented design defects.
- It may also help to determine the effect of object oriented design defects over the estimation of software reliability.
- It may facilitate to develop alternative object oriented designs.
- It may also serve to find out which design of two versions of object oriented software is more reliable than the older one.
- It may help to find out which object oriented design is more improved among the various designs of different software.

3.5 Conclusion

A reliability improvement framework for object oriented design defects has been developed. The proposed framework develops the mapping between the object oriented design constructs with design defects also develop mapping with design defects and reliability. No such framework has been available in the literature that improves software reliability of object oriented design by taking defects into consideration. The framework performs as a bridge between object oriented design constructs, design defects and reliability.

The objective of framework measures and mitigates the design defects of software design at early stage of software development life cycle contributing to reliable improvement. Reliability and design defects estimation models have been proposed with the help of proposed framework. Rigidity estimation model has been developed which accepts OOD constructs into consideration and proposed reliability estimation models assumes design defects in circumstances for estimating reliability of object oriented design. Framework enables to rays various questions i.e. can design defects be consider as a key factor to reliability? How object oriented design

constructs and design defects are related with each other? What are the effects of object oriented design constructs which can increase or decrease the design defects of object oriented design? What is the relation between object oriented design defects and reliability of software design?

CHAPTER 4: IMPLEMENTATION OF THE FRAMEWORK

CHAPTER 4: IMPLEMENTATION OF THE FRAMEWORK

Every big computing disaster has come from taking too many ideas and putting them in one place.

- Gordon Bell-

4.1 Background

Implementation means, the act of reaching some aim or executing some instruction in some order to carry out the effect. The purpose of framework is a hypothetical description of a complex entity or process by which facts or guidelines are carried out to a workable model[63]. The proposed framework integrates the model with each other that improves both the development of the software. Each phase of the proposed reliability improvement framework for object oriented design contains guidelines for the improvement of the framework. Theses phases are named as identification phase, mapping phase, measurement phase and improvement phase. To define the usage and effectiveness of the framework, implementation of the framework is significant. Implementing the framework means developing an approach for defect mitigation by using the reliability improvement guidelines of the proposed framework. The developed approach will identify and measures the object oriented design defects so that defects can be mitigate at the design phase in software development life cycle (SDLC).

The implementation of the proposed framework is choosing different sets of object oriented design constructs and design defects. Object oriented design defects can be measure in various ways and consequently defect mitigation guidelines can be produce. The approach of this research is concerned for the implementation of the framework. In the identification phase of the framework indentifying object oriented design constructs (Inheritance, Cohesion and Coupling), identifying reliability factor and design defects. Positive or negative impact of mapping on object oriented design constructs and design defect are analyzed in mapping phase.

Mapping phase established two relations (i) Mapping between object-oriented designs constructs and design defects (ii) Mapping between design defects and reliability. Some of the design constructs such as high coupling and inheritance increase the impact of design defect while high cohesion decrease the impact of design defect on object oriented design. For defect, rigidity estimation model and

reliability estimation model are developed to quantitatively minimize in measurement phase and the improvement phase.

4.2 The Framework

The framework proposed in chapter 3 for reliability improvement of object-oriented design contains four phases. These phases are identification phase, mapping phase, measurement phase and improvement phase. Each of the phases is revised and reviewed after implementation. Identification phase is responsible for identifying object oriented design defects, is a factor of reliability or not. Identification of design defect is important factor of reliability, cause of failure due to defect occur in object oriented design. Selection of object oriented design constructs dealing the need of reliability in software. Mapping phase establishes the mapping between object oriented design constructs and design defect and mapping between design defects and reliability.

Measurement phase includes measure object oriented design metrics, measure defects and reliability by using rigidity estimation model (RiEM) and reliability estimation model (RiEM). Improvement phase is the last phase which incorporated to contain the changes, mitigate the design defect to improve the reliability of object oriented design. Review from analysis and expert views are incorporated and guidelines contains the improvements in improvement phase. Each phase is deeply revised and reviewed for the possible improvements. Feedbacks obtained from the reviewers are analyzed and accordingly suggestions are incorporated, if necessary for further improvement of the phase.

4.3 Framework Implementation

The proposed framework for reliability improvement of object-oriented design is implemented in four phase as represented in following ways:

4.3.1 Implementing Identification Phase

The identification of the proposed reliability improvement framework is associated with the following identification activities:

(i) ***Identifying Object Oriented Design Constructs:*** The researcher has identified the following relevant object oriented design constructs i.e. inheritance, cohesion and coupling. These object oriented design constructs are used for estimating design defects of object-oriented design by using rigidity estimation model (RiEM). Each of the object oriented design constructs are described in the following section:

- **Inheritance Construct**

One of the important characteristics of object oriented design is inheritance. It provides the reuse of the existing design. Inheritance refers to the concept of a new class being declared as an extension of previously defined class[61]. Inheritance is the ability of one class acquired the properties of another class. Inheritance works as a reusable mechanism of designing or building one class from another class. It acts as a major need and ability behind object oriented design to create new class from the existing one. Once a methods is defined in super class, that behavior is automatically inherited by all the subclasses. A class and its children share common set of properties.

Inheritance act as a major need and ability behind object oriented relationship between two classes. It is a key feature of the object oriented paradigm. This mechanism supports the class hierarchy design and captures IS-A relationship between a super class and its sub classes. It represents a hierarchy between different classes of objects[61]. Class design deals with functional requirements of the system; it is the highest priority in object-oriented design. The use of inheritance is demanded to reduce the amount of software maintenance necessary and ease the burden of testing[64-67]. By reuse of software through inheritance is claimed to produce more maintainable, understandable and reliable software[64]. If the number of classes is increase by inheritance, there are greater possibilities of having defects in the class[68]. Higher number of methods, attributes and classes increases high number of design defects.

- **Coupling Construct**

Coupling is the degree in which each program module believes on each one of the other modules. In structured design and programming the importance of coupling is used as main attributes related to the goodness of decomposition has been well

understood. Experts of software engineering assure that design with low coupling and high cohesion lead to products that are both more reliable and more maintenance[69-71, 143]. It provides an interaction among classes and is interdependent on other classes[72]. Coupling facilitates to bring data from one class to another class. If two classes interchange large amount of data; then they are called highly dependent with each other. High coupling make the design difficult to understand and maintain. Objects from distinct classes should have as small coupling as possible not only to make them more understandable; so that they may easily extracted from a particular application and reuse in new situation[73].

High coupling between two classes makes typical to understand one of them in separation. While in low coupling leads to self-contained and thus easy to understand, maintainable objects. Coupled classes depend on external classes to implement their functionality. Refactoring is one of the significant approaches to reduce coupling in a class[74]. Coupling is directly related to message passing[75, 166]. Classes are coupled, when methods declared in one class use methods or attributes of the other class[76, 79]. High object oriented design makes the system become more defects, hence defect increases when coupling increases[77].

- **Cohesion Construct**

Cohesion is the degree to which methods of a class are related to one another and work together to allow well bounded behavior. It is used to measure the functional intensity of a module[79]. To achieve high reliability software, most of the researchers and developers agreed that reliable software design changes decomposition of the problem into small modules. High cohesion indicates good class subdivision. Lack of cohesion or low cohesion increases complexity, thereby increasing no of defects during the development process[78, 79]. Cohesion works on a single method. It avoids the interaction with other classes. A class in which cohesion occurs has a minimal interaction with other classes[82]. To measure cohesion there is no satisfactory measure result[103,104]. Due to very less interdependence among classes, it is possible to reduce defects of the design. The aforementioned concludes that cohesion decreases defects.

(ii) ***Identifying Reliability Factors:*** Software reliability is the important factor to estimate the software quality as to estimate the software cost. Software reliability is defined as the probability of failure free-operation of a software system for a specified

time in a specified environment. Reliability is important, because software has become essential part of everyday life. It has been argued by various researchers and developer to define reliability in terms of its low level attributes[3]. Attributes can be directly measures to quantify software reliability. From literature survey, a list of reliability factors identified and presented in figure 4.3.1 (a). Some of them have positive effect and some of them have negative effect on reliability. Every factor has its own characteristic that has its own positive and negative impact[102].

Software defects are one of the major factors that can decide the time of software delivery[58]. Software defect is also involved in the reliability of data as well as system. In 1977, McCall's proposed one of the most consented quality model in which reliability has been taken as one of the significant factor among various quality factors. Further McCall considered reliability sub factors as consistency, accuracy and fault tolerance. It is based on three major representations such as product revision, product transaction and product operations[107]. In 1978, Boehm's propose quality model, which is similar to McCall quality model[106]. It presents a hierarchical quality model structure of characteristics each of which contributes to quality. The lowest level structure of the characteristics hierarchy in the model is the primitive characteristics metrics hierarchy. The high level characteristics represent basic high level requirements of actual use. The intermediate level characteristics 7 quality factors which are as follows: portability, reliability, efficiency, usability, testability, understandability and flexibility[106]. Software quality models proposed, confusion happened and new standard model was required. Therefore, ISO/IEC began to develop the required agreement and promote common standard for the whole world[148]. The benefit of ISO 9126 model is that it separates the internal and external characteristics of a software product[106]. The disadvantage is that there is not any method that these features can be measured. The model defined six special characteristics including Functionality, Reliability, Usability, Efficiency, Maintainability and Portability which are further divided into 21 sub characteristics[108].

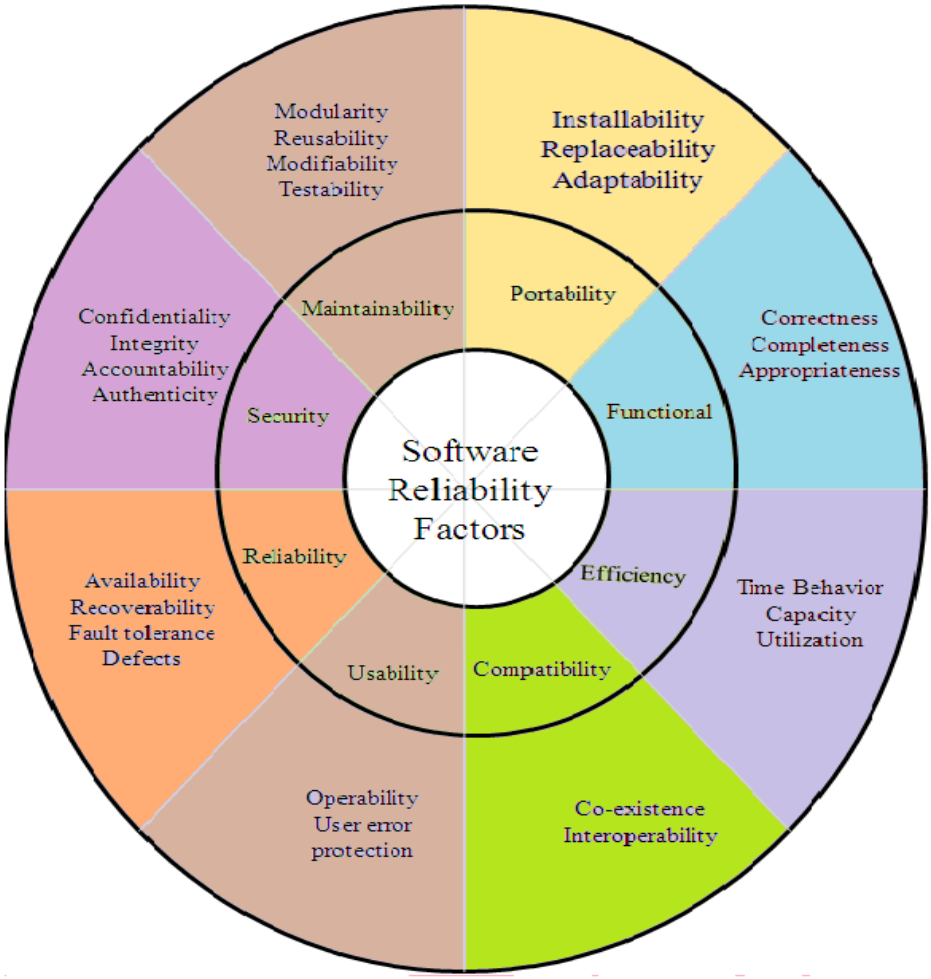

Figure 4.3.1 (a): Software Reliability Factors

In 1995, Dromey proposed product based quality model that recognize that quality evaluation differs for each product. Dromey's model attempt to increase the understanding of the relationship between the attributes of the software quality[57]. In 1996, SATC's quality model works for improving the software quality. It helps for software managers in establishing metrics programs to meet their basic needs with minimum cost. It follows the structure of ISO-9126-1 software quality models. In 2002, Bansiya's QMOOD extends Dromey's model. This includes four levels: (i) Identifying design quality characteristics. (ii) Identifying object oriented design properties (iii) Identifying object oriented design metrics (iv) Identifying OO design components. In 2006, Aspect oriented software quality model is proposed by Kumar et. al.. It is an extension of ISO-9126-1 software quality model. It included four new sub characteristics i.e. modularity, code-reusability, complexity and reusability[110]. UML model REASQ was developed by Castillo I et. al. to clarify the AOSD[111]. SQuaRE's model in proposed in 2011, to solve some problems of 9126 standard[112].

Figure 4.3.1 (a) software reliability factors describe all reliability factors into 8 factors and sub divided into 32 factors.

(iii) Identifying Design Defects: From the above discussion of various software quality models it is found that reliability is one of the important factors for software quality. Reliability is divided into sub characteristics in various factors. These factors affect reliability on some level. Researcher has selected all the sub factors of software reliability factors of quality models which are shown in figure 4.3.1 (a) in outer ring. From various research and literature survey it is found that defect is the sub-factor of software reliability, but it is not mentioned in any research paper what are the significance of object oriented design constructs and design defects on software reliability. If number of defects increased, reliability of the software decreased.

Changing of design defects of the software leads to more chance of failure of the software. This enforces the unreliability of the software. There are many examples and incidents from survey that due to design defects in the software by which reliability of the software is decreased[20]. On the basis of discussion and literature survey it is found that defects is taken as the important factor of reliability that negatively affects on the reliability of the software.

Object oriented principles advise the designers what to support and what to avoid. Now it is the time of period for OOD, because properties of OOD (Inheritance, Cohesion, Coupling, Polymorphism and Encapsulation) support the modification without changing the previous or existing modules. Martin proposes various primary symptoms tell whether designs are rotting[116, 117]. They are rigidity, fragility, immobility, needless complexity, opacity and viscosity. Rigidity is considered for design defect for object oriented software in this research. In 2014, Fillingham et. al. demonstrate how semantic rigidity can lead to significant long term problems that can contribute to model failure[118]. Object oriented design is essentially different from software developed using traditional methods. The purposes of design principles are to attach bad use of inheritance and poor dependencies of design structure along with other among other kind of design errors[119].

4.3.2 Implementing Mapping Phase

In today scenario, object oriented software is on peak demand in every field of the industry, aviation, education, hospital, banking and various other organizations. Most of the organization adopt object oriented concept in developing software or they have to upgrade from structured programming to OO programming[113]. Inheritance, cohesion and coupling are the most important object oriented design constructs which are mapped with reliability factors shown in figure 4.3.2 (a)

(i) ***Mapping between OO Design Constructs and Design Defects:*** In today scenario, object oriented is a booming language for developers and IT industry in corporate. It provides facility of inheritance, cohesion, coupling, encapsulation, abstraction and polymorphism for improving the ability of the software to be reused, tested and extended and maintained[114]. If inheritance and coupling decreases the consequence of the reliability on software in development life cycle is represented in Table 4.3.2 (a) represent the influence of design constructs on design defects. Up arrow (↑) represents the high influence and down arrow (↓) represents the low influence. If value of inheritance and coupling of class is high, design defects of a class design will be high. However, cohesion will be increased design defects of a class diagram is decreased. Value of design defect is neither 0 nor negative. Design defects can be controlled by inheritance, cohesion and coupling. Figure 4.3.2 (a) represents the mapping between object oriented design constructs and design defect.

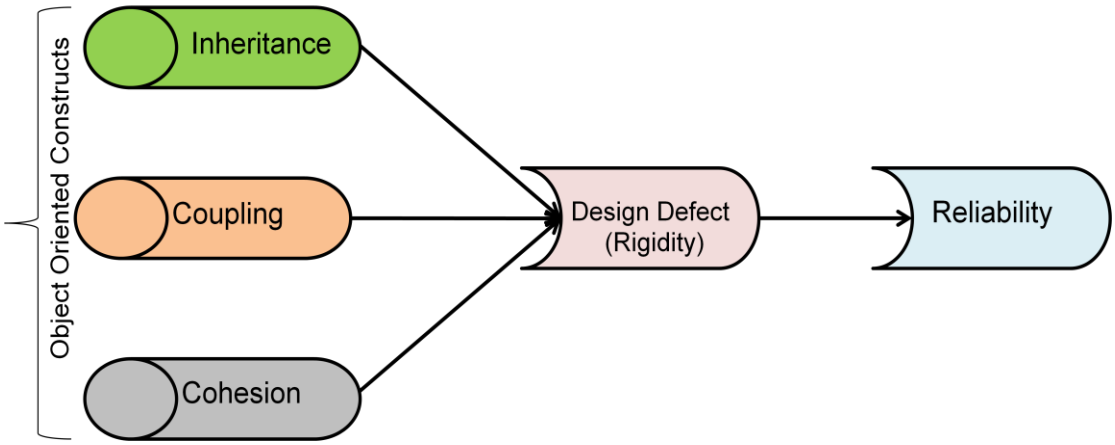

Figure 4.3.2 (a) Mapping between OO Design Construct, Design Defect and Reliability

Table 4.3.2 (a): Impact of OO Design Constructs on Design Defects

Design Constructs → / Design Defects ↓	INHERITANCE	COHESION	COUPLING
Rigidity	↑	↓	↑
Fragility	↑	↓	↑
Immobility	↑	↓	↑
Needless Complexity	↑	↓	↑

(ii) ***Mapping between Design Defect and Reliability:*** From literature survey and critical review it is found that early stage of design defects are removed, is a key role to achieve the reliability of the software[56]. Software development life cycle and reliability engineering help the designers to change, examine and maintain the reliability requirement in the early phase of the software development life cycle[94]. High design defect in the design is likely more occurrence of error that decreases the reliability of the software. Finally it is concluded that when design defect increase, reliability of the software decrease and when design defects decrease reliability of the software increase. It means reliability and design defect are highly correlated with each other is defined in figure 4.3.2 (b).

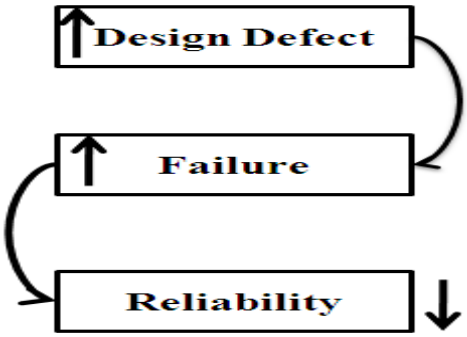

Figure 4.3.2(b): Impact of OOD Defect on Reliability

4.3.3 Implementing Measurement Phase

In this phase, for each of the identified object oriented design constructs (Inheritance, Cohesion and Coupling) is identified by the researchers. The metrics measure the design defects and reliability of the software design. Design defect estimation model and reliability estimation model have been calculated in the measurement phase.

(i) ***Measuring Object Oriented Design Constructs:*** The purpose of software metric is required for measuring the quality and design defect which helps to estimate the cost and project effort[120]. Metrics are used to measure various quality factors like fault-proneness, maintainability, reusability, reliability and understandability. There are generally two types of analysis applied to current software metrics[164]. In first category, theoretical analysis at conventional level software metrics as they are applied to traditional, non-object oriented software design. Second, category of analysis is more specific to object oriented design and development. Three object oriented design constructs are taken into consideration for the computation of the proposed object oriented design metrics which are as follows:

- **Measurement of Inheritance**

From review and research it is found that most of the other empirical studies of inheritance metrics have focused on maintenance effort and fault proneness. An object oriented programming approach is based on various key concepts like inheritance, cohesion, coupling, abstraction, encapsulation, polymorphism etc. Inheritance is one of the important key constructs of object oriented design. Various object oriented metrics are proposed and reviewed by the researchers available in the literature[85]. Some of the inheritance metrics are Method Inheritance Factor (MIF), Attribute Inheritance Factor (AIF) proposed by the Abreu[40]. Weighted Method per Class (WMC), Number of Children (NOC), Measure of Functional Abstraction (MFA) and Number of Descendent Classes (NDC) are proposed later for measure complexity of each method. Among the various metrics define to assess the design based inheritance, the most widely accepted and experimented is Depth of Inheritance Tree (DIT)[45, 64, 78, 86]. Object oriented design having inheritance allow all the methods and variables defined in the base class are accessible in the sub class. This property of inheritance increased the opportunity for defects in design and appears design as less reliable.

Object oriented design concept is one of the popular concepts in software development environment. These object oriented design is based on class, methods and attributes. The interaction of the methods and attributes are collected on the class that define a complete operation for each class and classify it by using inheritance. Object oriented design increases reusability of design and code of software which increase the cost, effort and time of software development.

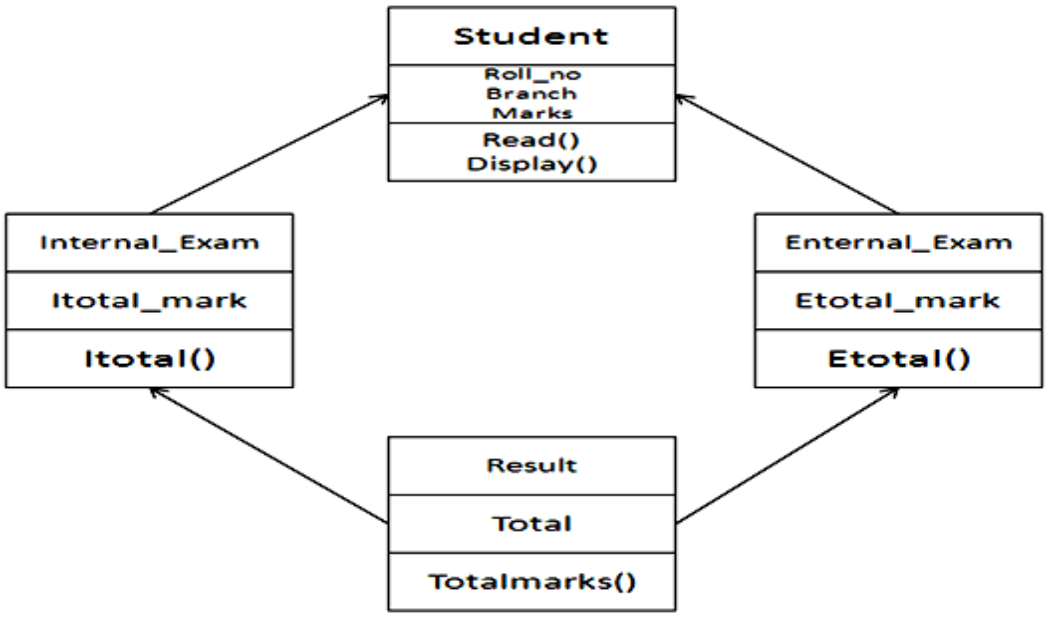

Figure 4.3.3 (a) Depth of Inheritance Metrics (DIT)

A defect is an item or work done in a prior step that cause trouble in a later step is termed as 'inherited' defect. It is an error or fault that travels along with the item and becomes a future problem in the same process or another process. Moha said that "design defects are bad solutions" to recurring design problems in object oriented systems[118]. Design defects are problems ranging from high level design to low level design. Thus it is concluded that inheritance increases design defects and decreases the reliability of the software design.

- ***Inheritance Metric for Design Defect of an Object Oriented Design***

Inheritance metric for object oriented design defines the average of summation of inheritance metric for defect mitigation perspective. Inheritance metric defect mitigation provides overall defect of a design hierarchy through inherited class and is estimated by taking average of the estimated values of the class.

$$DIT = AVG \{ \sum_{i=1}^{n} (DIT\ (i))\} \qquad (1)$$

Where n is the number of classes in an object oriented design. Avg. is an average of summation of inheritance metric design defect of class. It provides the statistical or expected single value of inheritance metric for design defect of an object oriented design which represents all the estimated values of inheritance class. In above figure 4.3.3(a) Depth of Inheritance Metrics (DIT) for Result class is 2 as it has two ancestor's classes internal_exam, external_exam and student. The value of DIT for internal exam and external exam is 1 as it has one ancestor class. The value of DIT for student class is 0, because there is not any ancestor class and it behave as a root class.

- **Measurement of Coupling**

There are different metrics of coupling proposed by the various researchers. Coupling and defect are closely related with each other. Two classes are coupled when methods declared in one class use methods or instances variable defined by the other class. As sharing word arranges is a tremendous faults to terminate reliability[121]. Due to sharing nature, information may leak either directly or indirectly from user domain to any unauthorized party. Therefore, coupling should be minimized by decreasing the value of design defect of object oriented design. In figure 4.3.3 (b), three classes are used Book, Publication and Sales. Class book used the instances of class publication as a Pub and from sales class as a Market. Therefore, the value of coupling for class Book is 2 while for the publication and sale is 0. Because these classes are not shared any method attribute from other class.

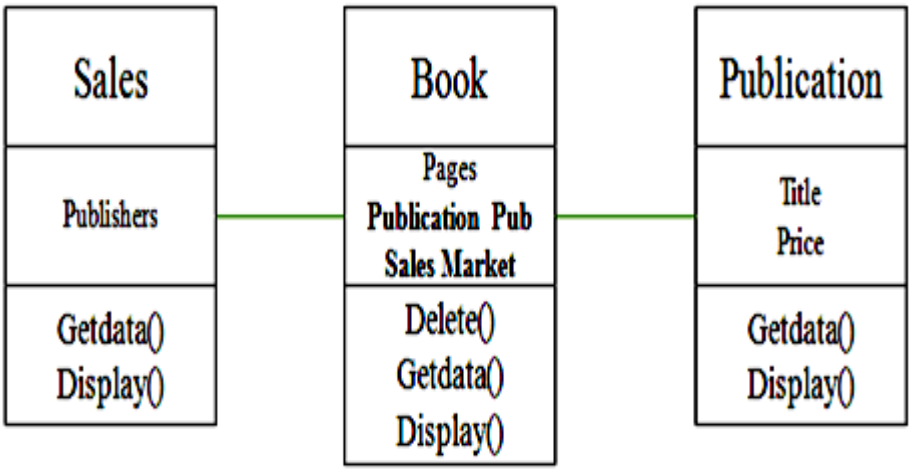

Figure 4.3.3 (b) Coupling Between Objects (CBO)

➢ **Coupling Metric for Design Defect of an Object Oriented Design**

Coupling metrics is defined as a summation of magnitude relationship among classes in row wise direction. Coupling between objects (CBO) is calculated as below, where CBO class is coupling metric design defect relationship per class.

$$\mathbf{CBO = AVG\ \{\sum_{i=1}^{n}(CBO(i)\}} \qquad \mathbf{(2)}$$

Where n is the number of classes in the object oriented design. Avg. is an average of summation of coupling metric of design defect of a class. It provides the statistical or expected single of coupling metric for design defect of an object oriented design which represents all the estimated values of inheritance class.

- **Measurement of Cohesion**

Cohesion is defined at class level. It is the measure of relatedness or consistency in functionality of a component. It is a highly suitable design characteristic because it measures change of responsibilities, independence of components and control of complexity. Cohesion can change the quality of a design, by assisting identity and redesign components that have distributed functionality or inconsistencies and that are complex. A class defined as the combination of data members and methods. Both data members and methods related with each other. The metric evaluates the relatedness of methods in the interface of a class using the parameter lists defined for the methods

Table 4.3.3 (a) Cohesion among Methods in Class (CAMC)

Operation	Attr_1	Attr_2	Attr…N
Method_1	C(1,1)	C(1,2)	C(1, n)
Method_2	C(1,2)	C(2,2)	C(2, n)
Method_M	C(m,1)	C(m,2)	C(m, n)

- **Cohesion Metrics for Design Defect of an Object Oriented Design**

Bansiya et. al. proposed a design class Cohesion Among Methods in Class (CAMC). It uses a PO matrix that has a row for each method and a column for each data type that appears at least once as the type of a parameter in at least one method in the class. The metric determines the overlap in the object type of the methods parameter list. The amount of overlap in object types used by the methods of a class can be used to predict the cohesion of the class. If all the methods of a class have access to similar parameter types, then it can be reasonable assumed that the methods process closely related information and thus must be cohesive in terms of the information processed[105]. In table 4.3.3(a), CAMC defined as the ratio of the total number of 1's in the matrix to the total size of the matrix types of method of a class. The cell value C (i, j) in row i and column j in the matrix is 1, when the i^{th} method has a parameter of the j^{th} data type and is 0 otherwise. A class is cohesive if all methods of the class use the same set of parameter types. N represents the number of method and m represents the number of operations.

It is defined as a summation of magnitude relationship among classes. It is defined as the average between low cohesion design defect and high cohesion design defect per class [42]. CAMC is defined as:

$$\mathbf{CAMC} = \frac{\mathbf{1}}{\mathbf{n.m}} \sum_{\mathbf{j=1}}^{\mathbf{n}} \mathbf{X(j)} \qquad (\mathbf{3})$$

$$\mathbf{X\ (j)} = \sum_{\mathbf{i=1}}^{\mathbf{m}} \mathbf{C\ (i, j)}$$

(ii) ***Measuring Rigidity using RiEM:*** The proposed OO design metrics being used for estimating design defects of object oriented design. To determine the relationship between various variables, regression analysis has been performed. Regression analysis is a set of statistical procedure for estimating the relationship between various variables. Thus, it shows the relationship between dependent and independent variable[59]. To determine the relationship between two or more independent variable are known as multiple correlations and the equation describing the relationship is known as multiple regression equation. Multiple linear regressions establish a relationship between multiple independent and dependent variables[122]. A

multiple regression equation has been generated from the regression equation which is as follows:

$$Y = \alpha_0 + \beta_1 X_1 + \beta_2 X_{2+} \ldots\ldots \beta n Xn \quad (4)$$

Where, Y is the dependent variable α and β are the regression coefficients and X_n are independent variables. Object oriented design constructs affects design defects. Design defects of software systems can be controlled by adjusting object oriented design constructs. Object oriented design metrics can be taken as independent variables while design defects is taken as dependent variables. Putting the value of object oriented design metrics of DIT, CBO and CAMC in equation (5) to measure the Rigidity defect equation which is as follows:

$$\text{Rigidity (R)} = \alpha + \beta\,(\text{DIT}) + \lambda\,(\text{CBO}) + \delta\,(\text{CAMC}) \quad (5)$$

Where R is the rigidity of class design which is dependent variables. DIT in inheritance metric of class, CBO is coupling between objects of class, CAMC is cohesion among methods in class are taken as independent variables. Design is dependent on independent variables that are OO design metrics and α (Alpha), β (Beta), λ (Lambda) and δ (Delta) is treated as regression coefficient.

Coefficient of above equation is calculated by the use of SPSS software. Standard values of the design defects are computed by the use of existing approach. Computed values of design defects through the use of existing approaches are 0.545, 0.55, 0.225 and 0.345. Design defects regression coefficients, for object oriented design metrics i.e. Depth of Inheritance Metrics (DIT) for Inheritance Measuring, Coupling Between Objects (CBO) for Coupling Measuring and Cohesion Among Methods in Class (CAMC) for Cohesion Measuring are computed as $\alpha = 0.0579$, $\beta = 0.1244$, $\lambda = 0.4126$, $\delta = -0.2892$ respectively. Putting values of α, β, λ and δ in equation (5) to generate the equation for rigidity defect measurement. Equation (6) computes the rigidity and is termed as rigidity of the design as rigidity estimation model (RiEM).

$$\text{Rigidity (R)} = 0.0579 + 0.1244(\text{DIT}) + 0.4126(\text{CBO}) + 0.2892(\text{CAMC}) \quad (6)$$

(iii) ***Measuring Reliability using ReEM:*** Design defect is one of the most important key factors of reliability. Mapping between design defect and reliability is

shown in mapping phase. High design defects decreased the reliability of the design. To make the system more reliable it is necessary to have less design defects in the system. Reliability of the system is dependent on design defects of the software to some level. The developed equation (6) is being used for estimating reliability of object oriented design because reliability is dependent on design defects and design defects depend on object oriented design metrics. A linear regression for measuring reliability can be established between design defect and reliability. A multiple linear regression has been established to get coefficients. The multiple linear regressions establish a relationship between dependent variables and multiple independent variables as discussed in design rigidity estimation model. Thus the multiple regression equation can be taken are as follows:

$$Z = \lambda 0 + \Gamma 1X1 + \Gamma 2X2 + \ldots\ldots\ldots \Gamma nXn \qquad (7)$$

Where Z is the dependent variable, λ (Lambda) and Γ (Gamma) are the regression coefficients, and X is the independent variables. Reliability is taken as dependent variable on rigidity defect variable. In terms of reliability, equation (7) is generated which are as follows:

$$Y = a + b(X) \qquad (8)$$

Where $\hat{Y}$ is the dependent variable a and b are the regression coefficients and X are the independent variables. In terms of reliability, equation (8), can also be written as:

$$Re = \alpha + \beta(Ri) \qquad (9)$$

Where, Re is the reliability which is dependent variable, Ri is Rigidity defect which work as dependent variables and α, β are treated as reliability regression coefficients. Reliability depends on rigidity defect. Researcher tried to estimate reliability of object oriented design by taking measuring rigidity defect. Rigidity defect act as independent variable and calculated in equation (6). A linear regression equation has been generated. Coefficients of the equation (9) are generated by the use of SPSS software. Standard values of the reliability are calculated by the use of existing approach[123]. Computed values of design defects through the use existing

approach are α = 1.021 and β = - 0.3651. Putting values of α and β in equation (9) to generate equation (10) for reliability measurement. Equation (9) computes the reliability is termed as reliability of the object oriented design as reliability estimation model (ReEM).

$$\mathbf{Re = 1.021 - 0.3651(Ri)} \qquad \mathbf{(10)}$$

4.3.4 Implementing Improvement Phase

It is the last phase of the reliability improvement framework. In this phase, the analysis of metric values on the basis of object oriented design metric, rigidity model, reliability model and suggestive measures are developed and mitigate the defects to improve the reliability of object oriented design. The improvement phase compels all the improvements and changes obtained while review the framework. It includes three phases which are as follows:

- ***Analysis of Depth of Inheritance Tree (DIT):*** With the help of analysis and the available result of DIT metric, it will be used to develop guidelines for object oriented design to improve reliability. From the measurement, it is found that the value of DIT is near to 0 for minimum limit. It may serve as an assistance to change rigidity defect at the objective level of the reliability. The value shows that DIT to its minimum limit to 0 will refers for better result and provide more reliable design. Thus from the above details it can be resolved that:

'The high value of DIT of an OO Design, increases the rigidity defect'

- ***Analysis of Coupling between Objects (CBO):*** With the help of analysis and the available result of CBO metric, it will be used to develop guidelines of object oriented design to improve reliability. From the measurement it is found that the value of CBO is near to 0 for minimum limit. It may serve as an assistance to change rigidity defect at the objective level of the reliability. The value shows that CBO to its minimum limit to 0 will refers for better result and provide more reliable design. Thus from the above details it can be resolved that:

'The high value of CBO of an OO Design, increases the rigidity defect'

- ***Analysis of Cohesion among Methods in a Class (CAMC):*** With the help of analysis and the available result of CAMC metric, it will be used to develop guidelines to improve reliability of object oriented design. From the measurement it is found that the value of CAMC is near to 0 for minimum limit and 1 for the maximum. It may serve as an assistance to change rigidity defect at the objective level of the reliability. The value shows that CAMC lies between 0 and 1 will refers for better result and provide more reliable design. Thus from the above details it can be resolved that:

'The high value of CAMC of an OO Design, decrease the rigidity defect'

- ***Analysis of Rigidity Estimation Model (RiEM):*** Rigidity estimation model is mapped with object oriented design constructs. During computation of rigidity it is found that the value lies between 0 and 1 is 0.828. on analysis of RiEM model value of the design will always be greater than 0 and less than 0.7629. If rigidity of an object oriented design is greater than 0.7425 than reliability of the system decreases, on the other hand it is found that on procedure of rigidity defect values, if rigidity defect values less than 0.7629, reliability of the design increases. This may serve as a help to change rigidity and achieve target level of reliability. The range signifies that a value of design defect closure to its minimum limit i.e. 0, will signifies a better and more reliable design. Thus from the above details it can be resolved that:

'The high value of rigidity of an OO Design, increases the rigidity defect of design'

- ***Analysis of Reliability Estimation Model (ReEM):*** Reliability estimation model is correlated with reliability estimation model. While analyzing the reliability estimation model, it is found that value of reliability of the design will always be less than 1. Reliability of the object oriented design may increase when values of rigidity defect movement to minimum value i.e. 0. This may serve as a help to change rigidity defect and achieve target level of reliability. The value signifies that a value of reliability near to its maximum limit indicates a better and more reliable design. Thus from the above details it can be resolved that:

'The high value of reliability of an OO design, decreases the rigidity defect of design'

- ***Development of Reliability Improvement Guidelines:*** On the basis of analysis of object oriented design metrics, suggestive measures are produced. These suggestive measures are Reliability Improvement Guidelines for Object Oriented Design (^{RI}GOOD). The proposed reliability improvement guidelines have been extracted from the analysis of the metrics and models developed.

RIG001: Keep the value of DIT for the object oriented design closure to its minimum limit value to 0.

RIG002: Keep the value of CBO for the object oriented design closure to its minimum limit value to 0.

RIG003: Keep the value of CAMC for the object oriented design closure to its maximum limit value to 1.

RIG004: Keep the value of rigidity estimation model for the object oriented design lies between 0 and 1.

RIG005: Keep the value of reliability estimation model for the object oriented design lies between 0 and 1.

In the light of above guidelines, the following characteristics are made by the designer in order to improve the reliability of object oriented design for developing more reliable software.

- Keep the inheritance of object oriented design as low as possible.
- If class inherited, make sure that attributes and methods are not used unnecessary.
- If there is inheritance in classes, make assurance that data members and methods are not used unnecessarily by the relative classes.
- Acquiring of data members and methods by some other methods must be avoided until necessary.
- Try to avoid coupling between classes as much as possible.

- The attributes defined in class not passed as parameter to other method of a simple class.
- Any simple class should not have a return type of a complex class until required.
- Too much method makes a system complex and increases the probability of defects.

4.4 Demonstration of Measurement

For making implementation of measurement phase more clear to the readers, a case study (Loan System) has been carried out as follows:

4.4.1 Measuring Object Oriented Design Constructs

In identification phase, object oriented design metrics are identified and measure the value of metrics. Object oriented loan system design is taken from[139]. Figure 4.4.1is an object oriented loan system. Table 4.4.1computes the value of DIT, CBO and CAMC. During the measurement of object oriented design metrics, analysis of models have also been done. It is found from the data analysis that DIT and CBO increases the rigidity defect and decrease the reliability of the design while CAMC decreases the rigidity defect and increases the reliability of the object oriented design.

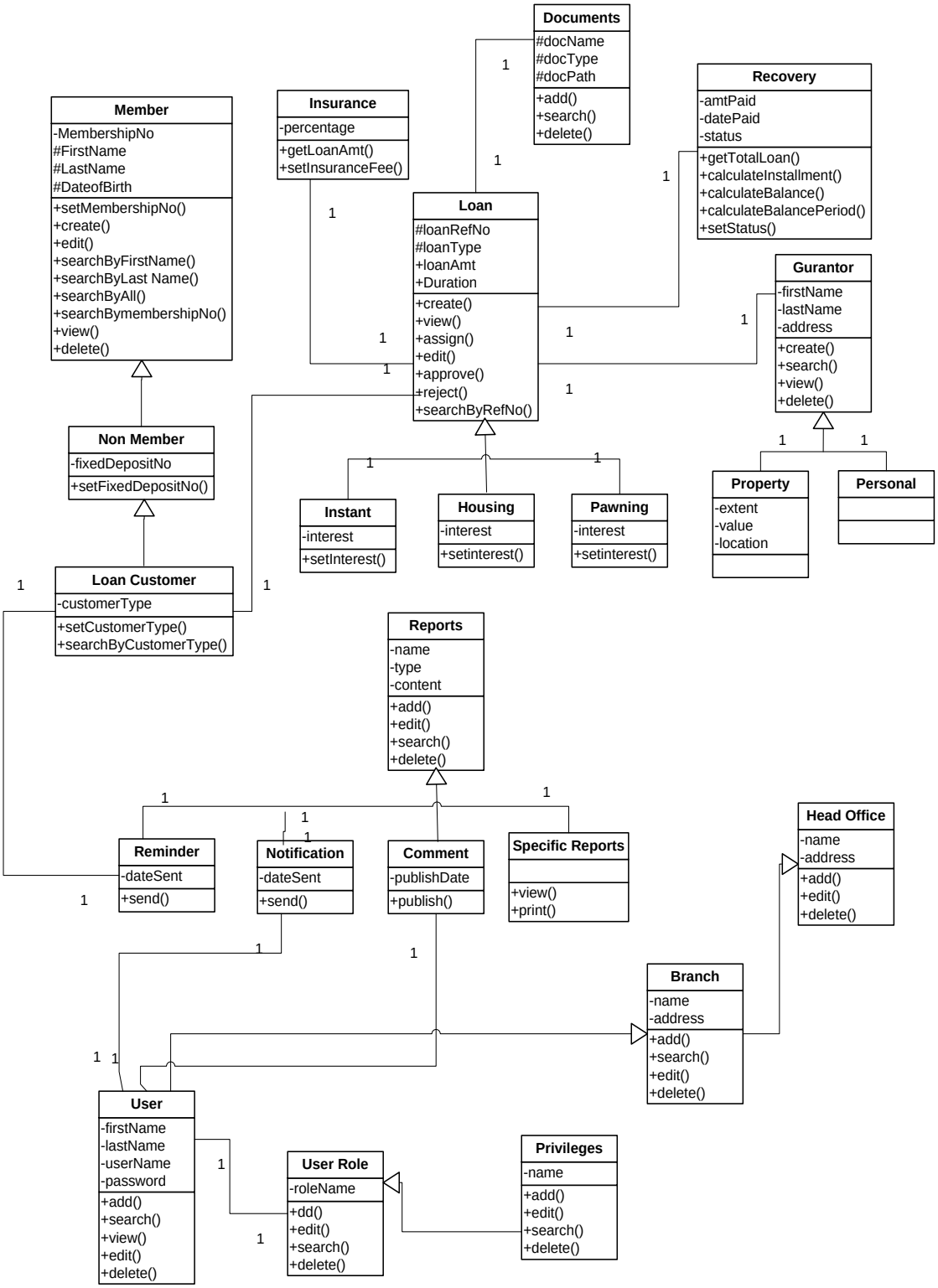

Figure 4.4.1: Object Oriented Design of Loan System

Table 4.4.1: Computed value of DIT, CBO and CAMC Metrics for OO Design

S. No	Class Name	DIT	CBO	CAMC
1	Member	0	1	0.24
2	Non Member	2	0	0.23
3	Loan Customer	2	1	0.12
4	Reports	1	2	0.32
5	Reminder	4	3	0.32
6	Notification	1	2	0.23
7	Comment	3	1	0.18
8	Sys Specific Reports	4	2	0.15
9	User	2	0	0.21
10	User Role	2	2	0.16
11	Privileges	1	2	0.18
12	Branch	0	1	0.32
13	Head Office	0	1	0.36
14	Property	1	0	0.08
15	Personal	2	1	0.85
16	Guarantor	3	1	0.36
17	Recovery	2	1	0.23
18	Loan	2	2	0.21
19	Documents	3	2	0.35
20	Interface	3	1	0.26
21	Instant	2	1	0.25
22	Housing	1	**0**	0.14
23	Pawning	2	2	0.23
Total		**43**	**29**	**5.98**
Average Metric Value		**DIT = 1.86**	**CBO = 1.26**	**CAMC = 0.26**

4.4.2 Measurement of Rigidity

The proposed rigidity estimation model is used for measuring the design defect of taken object oriented design of loan system Equation (6) is used for measuring the rigidity of object oriented design. Estimated values of object oriented design construct metrics (DIT, CBO and CAMC) used in equation (6). The computed value of rigidity is 0.7286 for the loan system and is shown in table 4.4.3

4.4.3 Measurement of Reliability

The proposed reliability estimation model is used for estimating the reliability of an object oriented design. Presently, reliability is calculated for an object oriented design loan system. Equation (10) is used to measure the reliability of the design. Estimated value of rigidity design defect is used to measure the reliability of the object oriented design. Value of rigidity is used in equation (10) to measure the reliability by using reliability estimation model (ReEM). The measured value of reliability is 0.7550 for the loan system design and is shown in table 4.4.3

Table 4.4.3: Computed value of Rigidity and Reliability for Loan System

Object Oriented Design	DIT	CBO	CAMC	Rigidity (Defect)	Reliability
Loan System	1.86	1.26	0.26	0.7286	0.7550

4.5 Conclusion

Software reliability is an important area of software engineering that has been not considered by the developers from to three decades. It becomes an important part for software organization and research. It is the capability of software to perform its specified level of performance. As the development growth of software industry, the software group to perform its specified level of performance. As it goes faster, the software group goes on developing technology that makes software easier, faster and less costly to build high quality software. Software failure occurs due to errors in the design phase or in the process through which design translated into the machine code. From last two decades, due to increase in dependency on software, the size and complexity of system has grown at very high level. Due to change of design defect in software systems, reliability of the system decreases. A highly complex system makes

the system less reliable. The objective of researcher is to keep the reliability of software by controlling the design defect of software.

To consider the demand and develop an approach which can be applied at the design phase of object oriented software, the researcher has proposed a framework for software development. The implementation of any framework indicates its usage in real life. The framework for software reliability improvement for defect mitigation perspective is implemented. The researcher has identified various attributes of software reliability. With the help of literature survey researcher does not found anywhere that design defect is a factor of reliability but it is found that design defect reduce the quality and reliability of software. A single defect makes the system unreliable. Due to available of design defect, various hazards have occurred previously. Due to occurrence of these hazards people lost their life, researcher has taken design defect one of the factors to improve reliability of object oriented design. Design defect reflects the negative impact on software reliability. Researcher identified three object oriented design constructs inheritance, cohesion and coupling. The researcher mapped object oriented design metrics with design defect and design defect with software reliability and proposed rigidity estimation model and reliability estimation model.

Design defect is treated with three identified object oriented design constructs in order to measure rigidity of object oriented design. These object oriented design metrics are as follows DIT (Depth of Inheritance Tree), CBO (Coupling between Objects) and CAMC (Cohesion among Methods in Class) are used for each of the object oriented design constructs inheritance, coupling and cohesion. The reliability of object oriented design is treated with design defect of object oriented design in order to estimate reliability of the design. Measured values of design defect and reliability are analyzed with each other. It is found that a calculated value of design defect is less than the calculated values of reliability. It is implemented on an object oriented design namely loan process system. On the base of measurement, researcher has quantified the value of rigidity and reliability. Analysis of metrics and models produced the reliability improvement guidelines for object oriented design. These guidelines will help to improve reliability of the design of object oriented software by mitigating the design defect with the help of object oriented design metrics.

CHAPTER 5: VALIDATION OF FRAMEWORK

5.1 Background

5.2 Validation

- 5.2.1 Design of Experiment
- 5.2.2 Pre Tryout
  - (i) Measurement of Object Oriented Design Constructs
  - (ii) Measurement of Rigidity Using RiEM
  - (iii) Measurement of Reliability Using ReEM
  - (iv) Implementation of Reliability Improvement Guidelines
  - (v) Recollection and Comparison of Metric and Model Values
- 5.2.3 Review and Revision
- 5.2.4 Tryout
  - (i) Collection of Metric and Model Values
  - (ii) Improving Reliability of Object Oriented Design
  - (iii) Recollection of Metric and Model Values

5.3 Statistical Analysis

- 5.3.1 Hypothesis Testing
- 5.3.2 Statistical Interpretation

5.4 Conclusion

CHAPTER 5: VALIDATION OF THE FRAMEWORK

"If there is no struggle, there is no progress"

*- **Frederick Douglass-***

5.1 Background

Validation is the process of finding or testing the truthfulness of something. The acceptance of any model depends upon its testing and validation. Validation in design phase of software engineering is important for the continuing advancement of both design theory and the professional practice of engineering. Validation process is required by the researcher in design phase to guide the development and evaluation of new methods. Developers and testers need validation process to determine which methods to employ, as well as when and how to employ them. It is the validation which makes one to believe in the result of any methodology. Validation is especially important in large and complex project organization i.e. aviation industry, automobiles, health, robotics etc. Early stage design decision is characterized by uncertainty and ambiguity in large organizations. Later design decisions are made through processes involving teams with a variety of expertise, decision making, skills and information. In some situations, it may be challenging to establish the benefits of new methodology[80].

Validation is very important in the field of research. The recognition of any new proposed model/approach/ methodology/framework is based upon its complete validation of the project. Keeping in mind of the importance of validation in research, the framework proposed for improving reliability of objects oriented design is validated through theoretical as well as empirically. The objective of validation makes any one to believe in the result of any methodology[87]. The reliability improvement framework is proposed in chapter 3 for estimating reliability of object oriented is validated through empirical validation. To empirically validate the model, first it is implemented in chapter 4. The consequence of the proposed model from chapter 4 is validated to verify whether the model is developed to achieve the determine objective. The metrics used during implementation along with the guidelines developed are used in the validation of the models.

Statistics is a way for the collection, analysis and interpretation of data. These tools are used to process and analyze the data in an experiment. It improves the quality of the data with the design of the experiments and collection of sampling. It demands null hypothesis and alternate hypothesis for the analysis of the result. Statistical interpretation of the result of an experiment is one of the way to accept or reject the consequences of any research. On the basis of some set standards provided by the statisticians, the null hypothesis accepted or rejected. Rejection of the null hypothesis extends to the acceptance of the research outcome or vice versa.

Software engineering literature has reflected a relation for methods to validate measures of attribute of software products. Common practices for the validation of software engineering measures are not acceptable on scientific grounds. There are basically two types of validation i.e. theoretical and empirical validation[80]. Theoretical validation is carried out to show that a measure is really measuring the attribute. Empirical validation is carried out to demonstrate that a measure is useful in the sense that it is related to other variables in expected ways.

5.2 Validation

It is the most fundamental form of validation. It serves as the requirement to establish the usefulness of a measure of empirical validation[88]. It proves that a measure is useful in the sense that it is related to other variables required in the theory[89]. Empirical validation is achieved in terms of two investigations i.e. pre-tryout and tryout. Pre-Tryout involves a small set of data. If the analyses of the results of pre-tryout are satisfactory, tryout will be carried out on a larger set of data. Satisfactory results of tryout will conclude the result of the models.

5.2.1 Design of Experiment

Experiments are used for testing the objective and explain the nature of originality. It is the way of conducting a controlled test or investigation. An experiment is an arranged process contained with the objective of verifying, finding or establishing the validity of hypothesis[81]. Experiment and data collection are the instruments by which theories are validated. The objective of an experiment is to collect adequate data in order to obtain a statistical accurate result[82]. For the purpose of validating the proposed framework for mitigating object oriented design defects

and improving reliability of object oriented design experiments are performed. In pre tryout, an object oriented design is carried out as input.

Data have been collected for object oriented design constructs and design models then according to the ^{RI}GOOD, the given design is refactored. The values of metrics and models are recollected by implementing the ^{RI}GOOD on object oriented design. The comparison of metric values and models shows that the redesign value model is more reliable after refactoring than the old models. After analyzing, the result of pre-tryout, since no significant changes are noticed a tryout is carried out with the largest set of data. Ten object oriented design are taken for analysis. The same procedure is applied for ten designs. On the basis of the outcome, statistical interpretations are computed.

5.2.2 Pre Tryout

It has been carried out on an object orient design for each object oriented design constructs including inheritance, cohesion and coupling. The object oriented design OSCAR has been taken from[83]. The class hierarchy of the OSCAR design is shown in figure 5.2.2 (a) and the detailed design is attached at Annexure -2. Initially the identified metrics which are used for design are DIT, CBO and CAMC. Rigidity defect and reliability is calculated by using the RiEM and ReEM model. The design is refactored using the proposed ^{RI}GOOD. Metrics and models are recollected and compared with the previous value. The comparisons establish the improvement on the reliability of the OSCAR design.

(i) ***Measurement of Object Oriented Design Constructs:*** Object oriented design metrics is implemented in chapter 4, and compiled quantitative value for the OSCAR design. 15 classes are used in OSCAR designs which are as follows (Sequencer, Action, Transformable Object, Scene, Cue, Renderer, Light, Camera, Actor, Surface Property, Geometric Model, Phigs, Movie Byu, Phigs Model and Movie Byu Model). Computed value of metrics DIT, CBO and CAMC are shown in table 5.2.2 (a) and object oriented design constructs like inheritance, cohesion and coupling are used to compute for design metrics. The value of depth of inheritance tree (DIT), coupling between objects (CBO) and cohesion among methods in a class (CAMC) is computed and shown in table 5.2.2 (a). All the values are computed through SPSS software.

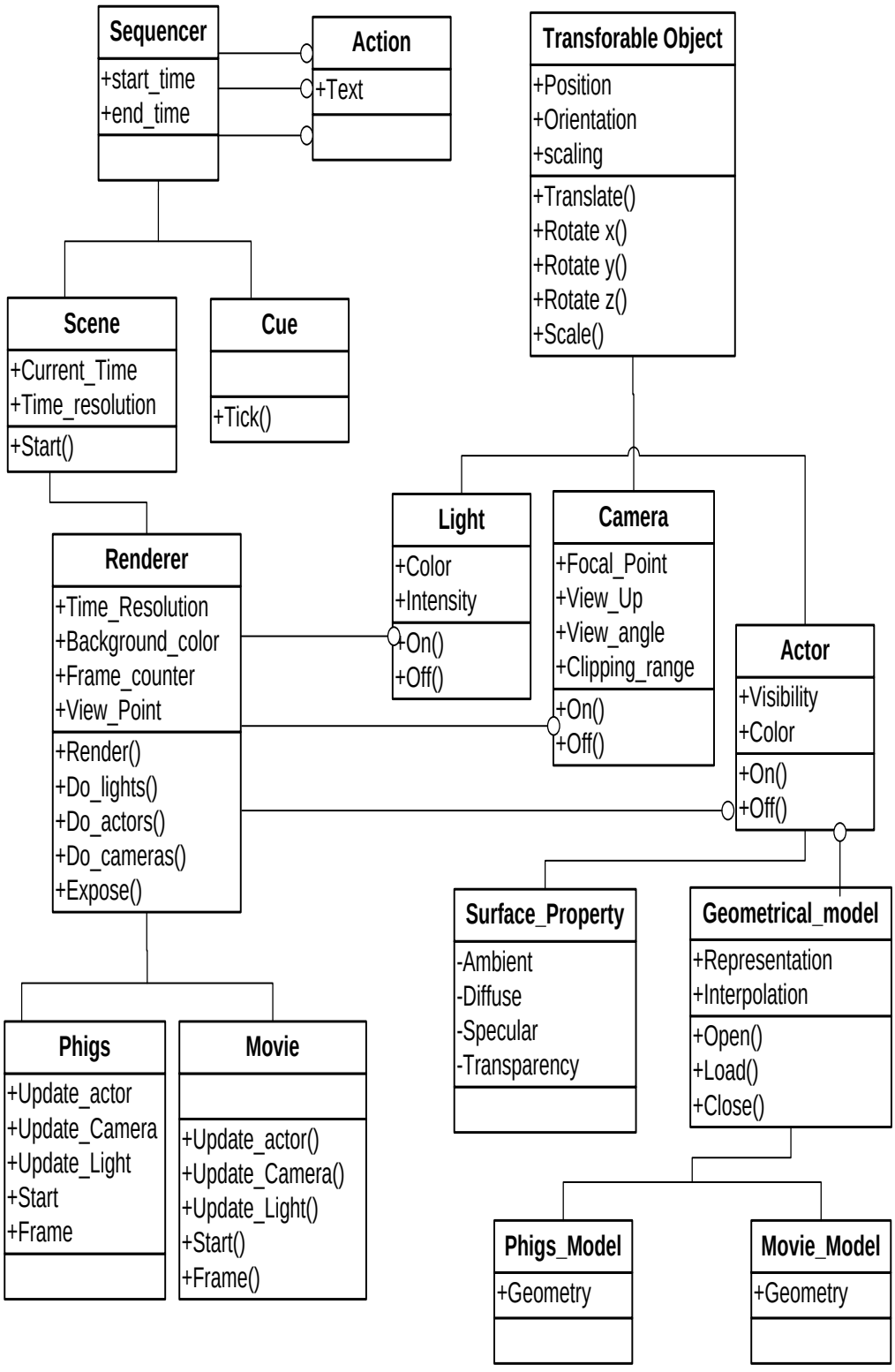

Figure 5.2.2 (a): Object Oriented Design of OSCAR System

Table 5.2.2 (a): Computation of DIT, CBO and CAMC for OSCAR Design

S. No	Classes	DIT	CBO	CAMC
1	Sequencer	0	1	0.11
2	Action	2	2	0.68
3	Transformable Object	1	3	0.54
4	Scene	1	3	0.34
5	Cue	3	2	0.56
6	Renderer	0	2	0.69
7	Light	2	1	0.78
8	Camera	1	2	0.88
9	Actor	3	1	0.65
10	Surface Property	1	2	0.34
11	Geometric Model	1	0	0.82
12	Phigs	4	1	0.34
13	Movie	2	2	0.12
14	Phigs_Model	1	1	0.62
15	Movie_Model	0	2	0.78
Total		22	25	8.25
Average Metric Value		1.44	1.66	0.55

(ii) ***Measurement of Rigidity using RiEM:*** Rigidity Estimation Model (RiEM) proposed in chapter 3 and implemented in chapter 4, computes the rigidity defect of the OSCAR object oriented design system. In chapter 4, equation (6), establish the relation between rigidity defect and object oriented design metrics like DIT, CBO and CAMC. The computed value of rigidity defect for OSACR design is 0.7629 and shown in table 5.2.2 (b)

Table 5.2.2 (b) Computed value of Rigidity and Reliability of OSCAR design

Metric Design	DIT	CBO	CAMC	Rigidity (Defect)	Reliability
OSCAR Design	**1.467**	**1.667**	**0.55**	**0.7629**	**0.7425**

(iii) ***Measurement of Reliability using ReEM:*** Reliability estimated model (ReEM) is proposed in chapter 3 and implemented in chapter 4 estimates the reliability of an OSCAR object oriented design. In chapter 4, equation (10) establishes the relation between rigidity defect and reliability of the object oriented design. Reliability is dependent variable while rigidity is independent variable in equation (10). The computed value of reliability is 0.7425 for the OSCAR design of object oriented design is shown in table 5.2.2 (b).

(iv) ***Implementation of Reliability Improvement Guidelines:*** To implement guidelines for improvement of reliability of object oriented design, value of rigidity is minimized. Object oriented design defects are depend on object oriented design constructs i.e. inheritance, cohesion and coupling. The computed value of object oriented design metrics (DIT, CBO, CAMC) are shown in table 5.2.2 (a). Rigidity Estimation model and Reliability estimation Model are applying on OSCAR object oriented design model. In the above table 5.2.2(b) the values of rigidity is greater than the values of reliability. This indicates that the design is rigid and less reliable. It also indicates that the design is highly rigid and less reliable. So, the designer required some guidelines to control the value of rigidity to mitigate the defect and improve the reliability. Therefore, researcher tries to mitigate the length of inheritance and value of coupling, and increase the value of cohesion by using refactoring regulations. There are various refactoring regulations at design level. The refactoring regulation works on the basis of using Move_Method, Move_Field, Pull_Up, Pull_Down and

Move_Class. The Move_Method and Move_Field rules state that move a method X and attributes Y from class A to class B that uses the method and attributes. The method X and attribute Y is completely removed from class A after refactoring. Field_Up and Field_Down rules state that a field is move from sub class to super class or super class to sub class. A Move_Class is applied when new class/function/module are generated by using the existing facilities. A class A has some methods and attributes. A new class is generated by using some of the method and attribute from class A to class B [154].

(v) ***Recollection and Comparison of Metric and Model Values:*** By using the above mentioned rules of refactoring, the methods of class Phigs and Movie has same methods in OSCAR design. Repletion of methods are defined in the class. With the help of refactoring rule methods of these classes are move to renderer class. These methods ar completely removed from Phigs and Movie class as shown in figure 5.2.2 (b). Figure 5.2.2 (b) is obtained from redesign of OSCAR design by using the refactoring rule. Computed value for object oriented design constructs (inheritance, cohesion and coupling) metrics DIT, CBO and CAMC respectively are used is shown in table 5.2.2 (c) of OSCAR redesign. The comparison of OSCAR design and redesign system is shown in table 5.2.2 (d).

Results received from the pre tryout validation on an object oriented design are statistically examined. Figure 5.2.2(c) comparison of OSCAR design and redesign of metric values, figure 5.2.2 (d) Comparison of OSCAR design and redesign of rigidity defect and reliability values, figure 5.2.2 (e) comparison of OSCAR design and redesign of Rigidity defect values and figure 5.2.2 (f) comparison of OSCAR design and redesign of reliability values are represented. By comparing the value of OSCAR design and redesign as shown in table 5.2.2 (d) it is found that the newer values of DIT and CBO are less than the older one. For CAMC values of OSCAR redesign is more than the values of OSCAR older design. Comparing the newer and older values of rigidity defect and reliability, it is found that the new values of rigidity defect are less than the old values of OSCAR design. Similarly, the new value of reliability is more than the old value. Figure 5.2.2 (d) shows comparative analysis of the older and newer values of rigidity and reliability values. The values reflect a significant improvement of reliability of OSCAR redesign over the OSCAR design.

This shows the strength of the proposed re4liability improvement guidelines for object oriented design [RI]GOOD.

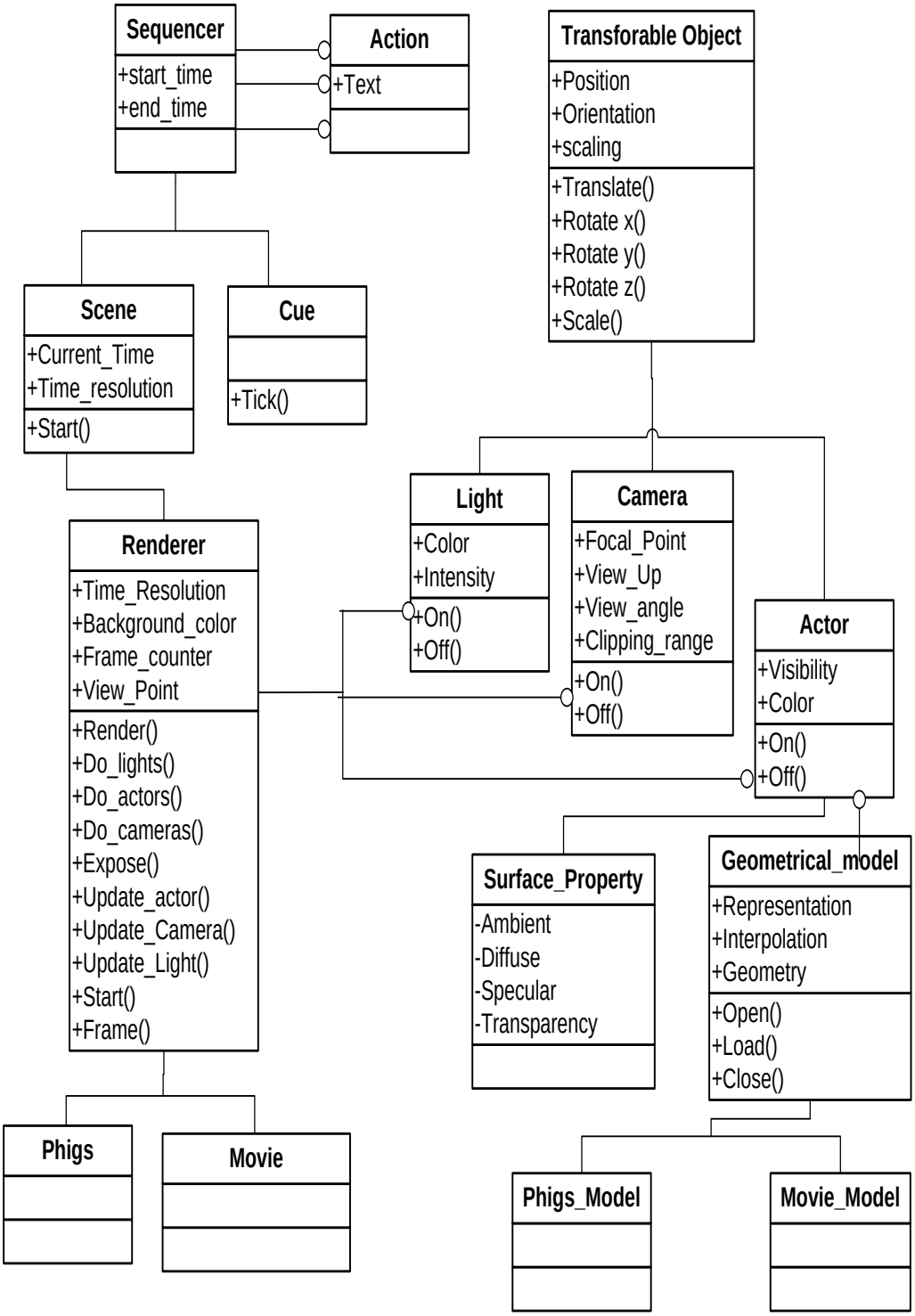

Figure 5.2.2 (b) Object Oriented Redesign of OSCAR system

Table 5.2.2 (c): Computation of DIT, CBO and CAMC for OSCAR Redesign system

S.No	Classes	DIT	CBO	CAMC
1	Sequencer	0	1	0.38
2	Action	2	2	0.86
3	Transformable Object	1	3	0.88
4	Scene	1	3	0.94
5	Cue	1	2	0.79
6	Renderer	0	2	0.89
7	Light	2	1	0.78
8	Camera	1	2	0.88
9	Actor	3	1	0.78
10	Surface Property	1	2	0.58
11	Geometric Model	1	0	0.82
12	Phigs	2	1	0.76
13	Movie	1	2	0.32
14	Phigs_Model	1	1	0.86
15	Movie_Model	0	2	0.78
Total		**17**	**19**	**11.3**
Average Metric Value		**DIT= 1.14**	**CBO= 1.26**	**CAMC= 0.75**

Table 5.2.2 (d): Comparison of Design and Redesign of OSCAR System

Metric & Model Values	DIT	CBO	CAMC	Rigidity (Defect)	Reliability
OSCAR Design	1.44	1.66	0.55	0.7629	0.7425
OSCAR Redesign	1.14	1.26	0.75	0.5027	0.8375

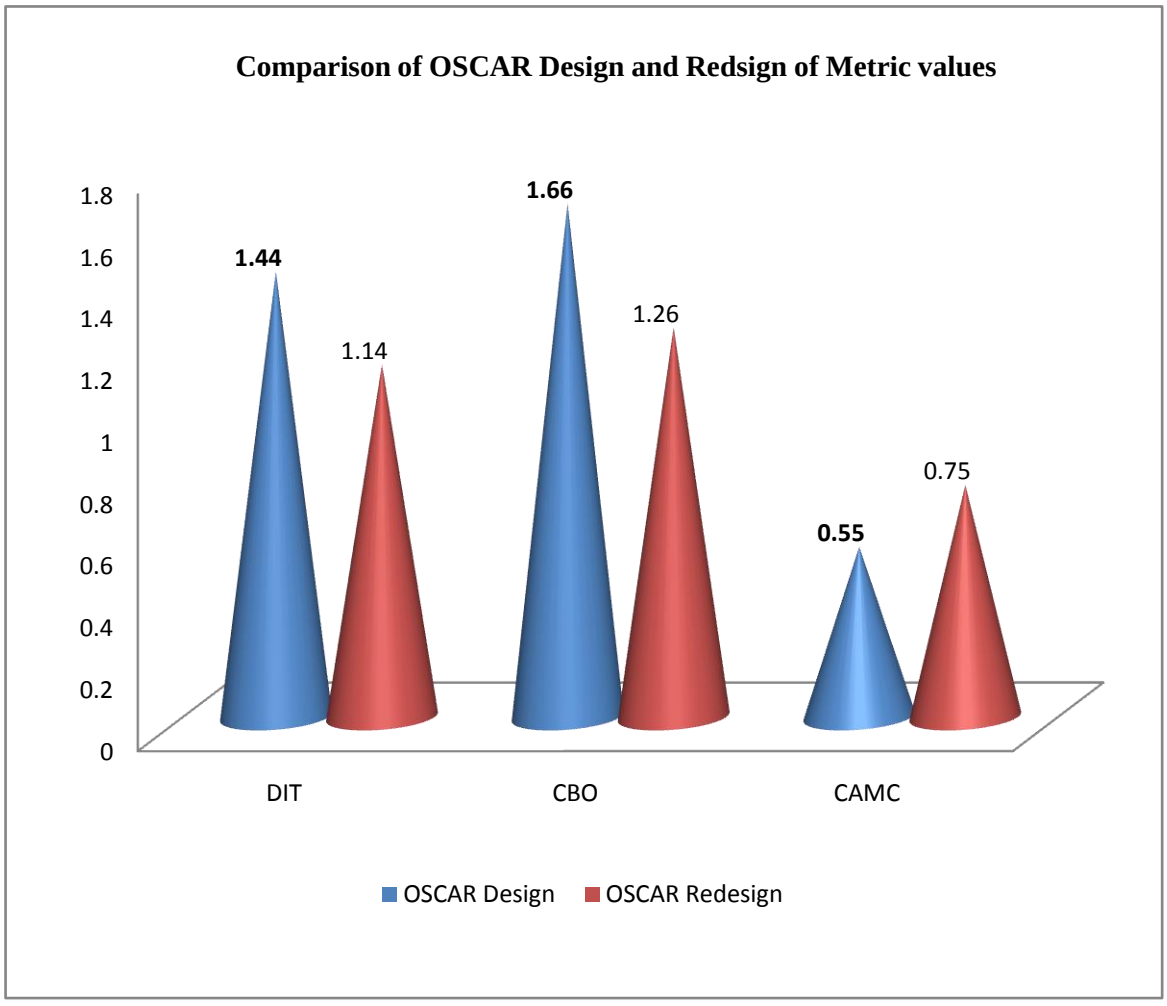

Figure 5.2.2 (c): Comparison of OSCAR design and redesign of metric values

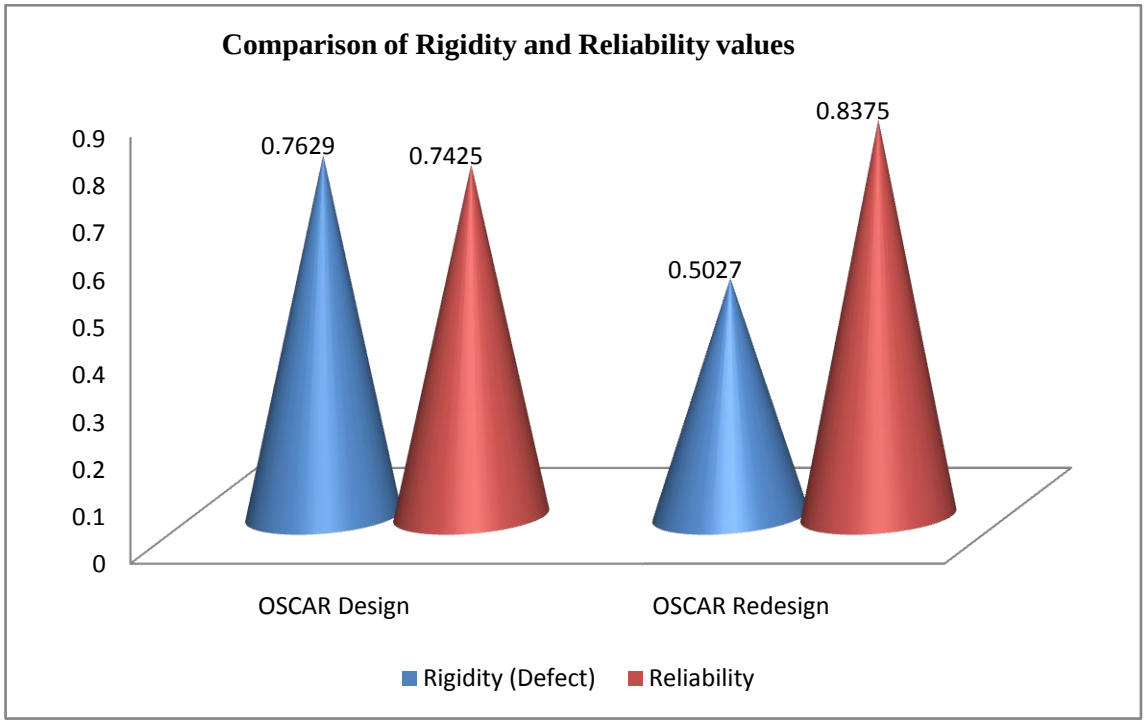

Figure 5.2.2 (d): Comparison of Rigidity and Reliability values

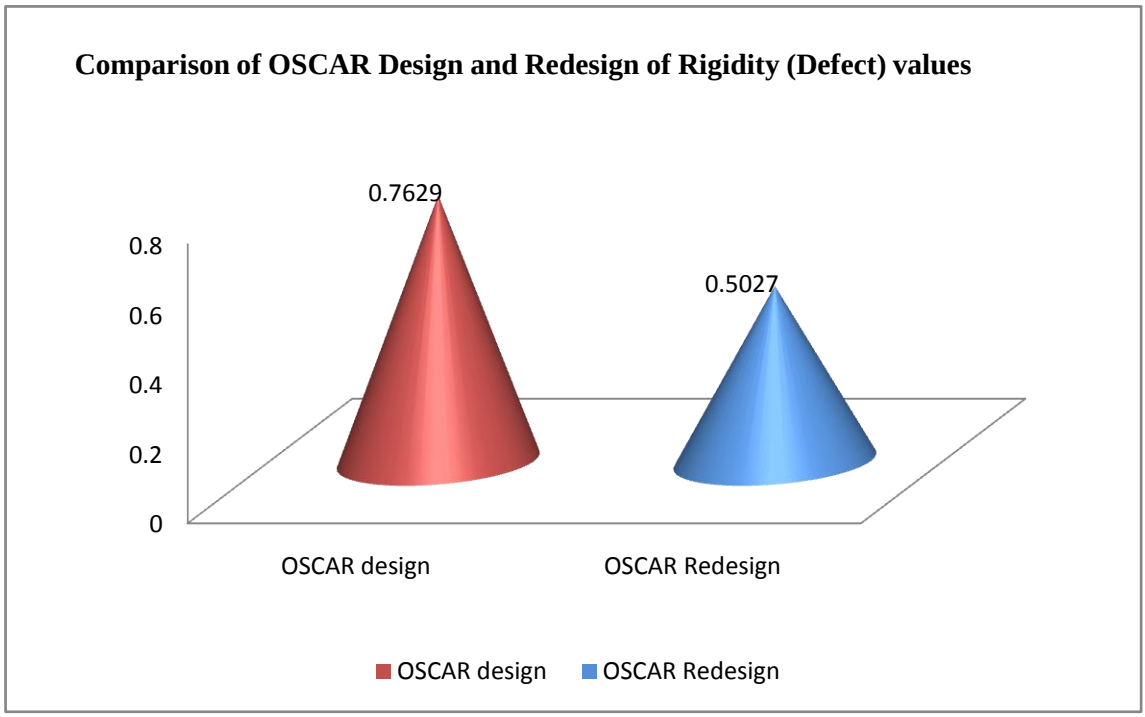

Figure 5.2.2 (e): Comparison of OSCAR design and Redesign for Rigidity Defect values

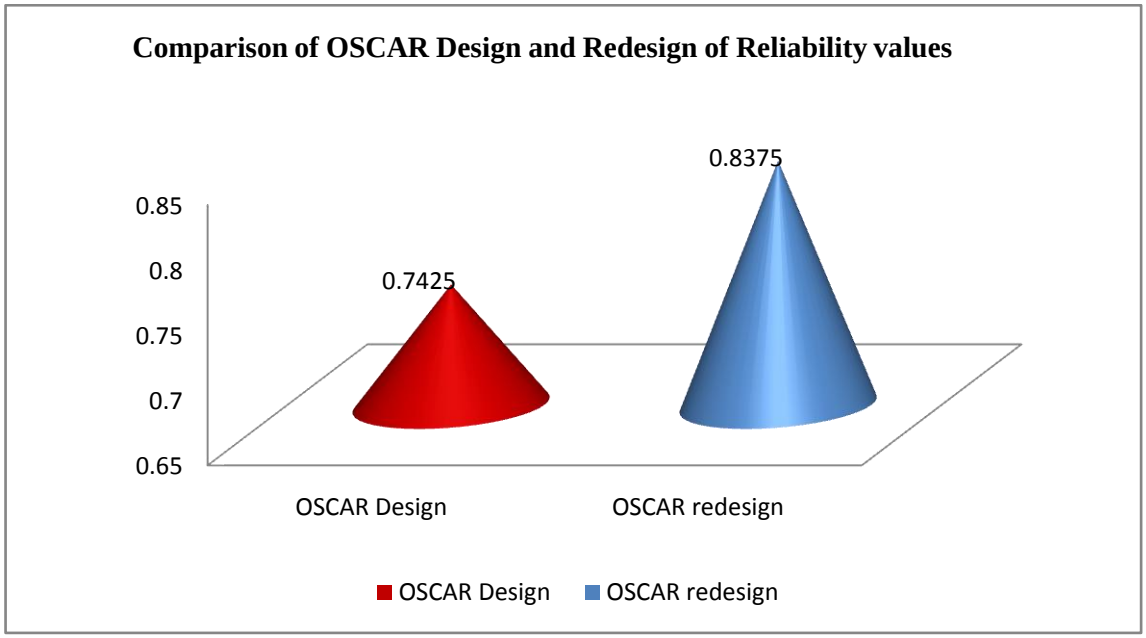

Figure 5.2.2(f): Comparison of OSCAR Design and Redesign for Reliability values

5.2.3 Review and Revision

Results obtained from pre tryout are analysed. A critical review of the outcome strengthens the acceptability of the proposed framework. Revision is not required if any suggestions are not applicable by the reviewers. Therefore, the researcher accommodates the same framework for further try out with a large set of data.

5.2.4 Tryout

Validation is an ongoing process and hence there are degrees of validity more evidence increase the validity of an approach. For collecting more and more evidences for validating of the proposed work, a tryout is carried out by adopting the pre tryout. For collecting more and more manifest for the proposed framework, a tryout is carried out by adopting the pre tryout. The tryout contains ten object oriented designs. These designs are developed by the B.Tech, M.C.A and M.Sc students in partial fulfilment of their final semester project. All the projects were developed in a software organization at UPTECH, Consultancy, Pvt. Ltd, Hazratganj, Lucknow. The size of the object oriented design ranges is 15 to 35 classes.

(i) ***Collection of Metrics and Model Values:*** Object oriented designs were used to analyse the values of proposed object oriented design metrics and models. For each

object oriented design metric and model values were collected. A tabular presentation of the metric values for all the ten designs has been given in table 5.2.4 (a).

Table 5.2.4 (a): Collection of Metric Values for Ten Designs

Metrics → Designs ↓	DIT	CBO	CAMC	Defect (Rigidity)	Reliability
Design 1	0.4561	0.9678	0.5678	0.3497	0.8933
Design 2	0.3472	0.5748	0.6573	0.1482	0.9669
Design 3	0.5671	0.8256	0.4241	0.3464	0.8945
Design 4	0.6721	0.9281	0.2426	0.4543	0.8551
Design 5	0.1218	0.7861	0.2781	0.3170	0.9053
Design 6	0.1416	0.9192	0.3456	0.3548	0.8915
Design 7	1.2357	0.8921	0.3452	0.4799	0.8458
Design 8	0.8256	0.8857	0.5641	0.3629	0.8885
Design 9	1.3278	0.8264	0.6721	0.3697	0.8860
Design 10	0.7856	0.7862	0.5468	0.3219	0.9035

(ii) ***Improving Reliability of Object Oriented Design:*** After collection of all the object oriented design metric values, rigidity and reliability values, each of the ten designs is thoroughly analysed keeping in mind for the object oriented design defect. By using the reliability improvement guidelines for object oriented design (^{RI}GOOD) proposed in the study, rigidity within each design has been mitigate and hence values of reliability is improved as much as potential. All the process are possible with the help of refactoring as the aim to develop alternate designs having the same functionality but less rigidity or much reliable.

(iii) ***Recollection of Metrics and Models Values:*** To assess whether the proposed ^{RI}GOOD guidelines is able to mitigate the rigidity and improve the reliability in the

design. Taken for the experiment or not, the value of three metrics and models were again collected for all the ten refactored design. The values are shown in table 5.2.4 (b)

Table 5.2.4 (b): Recollection of Metric Values for Ten Designs

Metrics → Designs ↓	DIT	CBO	CAMC	Defect (Rigidity)	Reliability
Design 1	0.4123	0.8236	0.5678	0.2848	0.9170
Design 2	0.3172	0.4721	0.6982	0.0902	0.9881
Design 3	0.4071	0.7832	0.5432	0.2746	0.9207
Design 4	0.5221	0.8267	0.2136	0.4022	0.8742
Design 5	0.1107	0.6827	0.3341	0.2567	0.9273
Design 6	0.1226	0.8276	0.4456	0.2858	0.9167
Design 7	1.0357	0.7856	0.4452	0.3821	0.8815
Design 8	0.6256	0.7821	0.6254	0.2776	0.9197
Design 9	1.0278	0.7934	0.7221	0.3043	0.9099
Design 10	0.6856	0.5723	0.6456	0.1926	0.9507

5.3 Statistical Analysis

Statistics develop techniques to overcome problem in data collection and data analysis. It is a mathematical tool used for gathering, organizing, examining and interpreting the values and information. The objective of establishing statistical significance or validation of the proposed reliability improvement framework, statistical analysis is carried out on the object oriented designs incurred from the UPTECH, consultancy, Pvt. Ltd, Hazratganj, Lucknow.

Statisticians have developed several test of hypothesis for the purpose of testing of hypothesis which can be classified as parametric test and non-parametric test. T-test is based on T-distribution and is considered an appropriate test for judging

the significance of difference between the means of two samples in case of small samples. As the sample size is small, T-test is applied for finding out the level of significance and rejection of the null hypothesis. After the rejection or acceptance of a null hypothesis is based upon either (0.05) or (0.01) level of significance for one tailed or two tailed test. Alpha level 0.01 of significance for a one tailed test is taken for rejection of the null hypothesis. The complete process of the following statistical analysis is summarized as: the first step starts with the formulation of null hypothesis and alternate hypothesis. The old metrics value and new metric values are placed for statistical analysis to find the conclusion that whether there is a significant relation between the pre treatment and the post treatment of the data. The obtained t value will determine whether to reject the null hypothesis and accept the alternate hypothesis[124].

5.3.1 Hypothesis Testing

It is used to conduct a quantitative research to attempting answer a research question or set of hypothesis. A null hypothesis reflects that there is no significance relationship between two or more parameters whereas alternate hypothesis supports the relationship[124]. Rejection of a null hypothesis provides a stronger base to accept the relationship or to accept the alternate hypothesis. As this study relates that design defect with object oriented design constructs inheritance, cohesion and coupling following null and alternate hypothesis were made for the purpose of validation of the proposed reliability improvement framework.

- **Null Hypothesis (H_{01}):** Rigidity defect of OO design cannot be mitigating by minimizing inheritance of OO design.
- **Alternate Hypothesis (H_{11}):** Rigidity defect of OO design can be mitigating by minimizing inheritance of OO design.
- **Null Hypothesis (H_{02}):** Rigidity defect of OO design cannot be mitigating by minimizing coupling of OO design.
- **Alternate Hypothesis (H_{12}):** Rigidity defect of OO design can be mitigating by minimizing coupling of OO design.
- **Null Hypothesis (H_{03}):** Rigidity defect of OO design cannot be mitigating by maximizing cohesion of OO design.
- **Alternate Hypothesis (H_{03}):** Rigidity defect of OO design can be mitigating by maximizing cohesion of OO design.

- **Null Hypothesis (H_{04}):** Rigidity defect of OO design cannot be mitigating by minimizing rigidity of OO design.
- **Alternate Hypothesis (H_{04}):** Rigidity defect of OO design can be mitigating by minimizing rigidity of OO design.
- **Null Hypothesis (H_{05}):** Reliability of OO design cannot be maximized by mitigating rigidity defect of OO design.
- **Alternate Hypothesis (H_{05}):** Reliability of OO design can be maximized by mitigating defect of OO design.

5.3.2 Statistical Interpretation

By observing metric value in table 5.2.2 (d) can be derived very easily that the ^{RI}GOOD treatment for all the design has worked well. The new metric values for the metrics DIT and CBO in table 5.2.2 (c) is less than the table 5.2.2 (a). Similarly, the new metrics value for CAMC in table 5.2.2 (c) is more than the value of table shown in table 5.2.2 (a). The new values of rigidity estimation model for design defect in table 5.3.2 (e) are comparatively less than those in table 5.2.2 (d). In the same way, the new value of reliability estimation model for reliability is shown in table 5.2.2 (e) is increased from table shown in table 5.2.2 (d). The table 5.2.2 (a) shown that the value rigidity is minimized and reliability is improved for all the ten designs of object oriented design. The initial value shows that ^{RI}GOOD is able to mitigate the rigidity of object oriented design and improve reliability of object oriented design is become proved.

A graphical representation of comparative study between new and old values of DIT, CBO, CAMC, rigidity estimation model and reliability estimation model is shown in table 5.2.2 (a), 5.2.2 (b), 5.2.2 (c) and 5.2.2 (d) respectively. But it is not enough to be able to prove its satisfactory. To make the model and ^{RI}GOOD derived from its satisfactory or for the validation of the approach. It must be verified whether the difference in the metrics value and model values is due to the use of ^{RI}GOOD or it is just a sampling error. For this purpose, the level of significance of the proposed approach must be computed. While studying inference data analysis, it was found that the t-test for the situation given below is appropriate for the purpose:

When the same group of individual taken a pre test, then the group is exposed treatment. The group is again tested after treatment to determine whether the

determine treatment has been statistically significant as determined by mean gain scores. The t-test was carried out for describing level of significance of the approach.

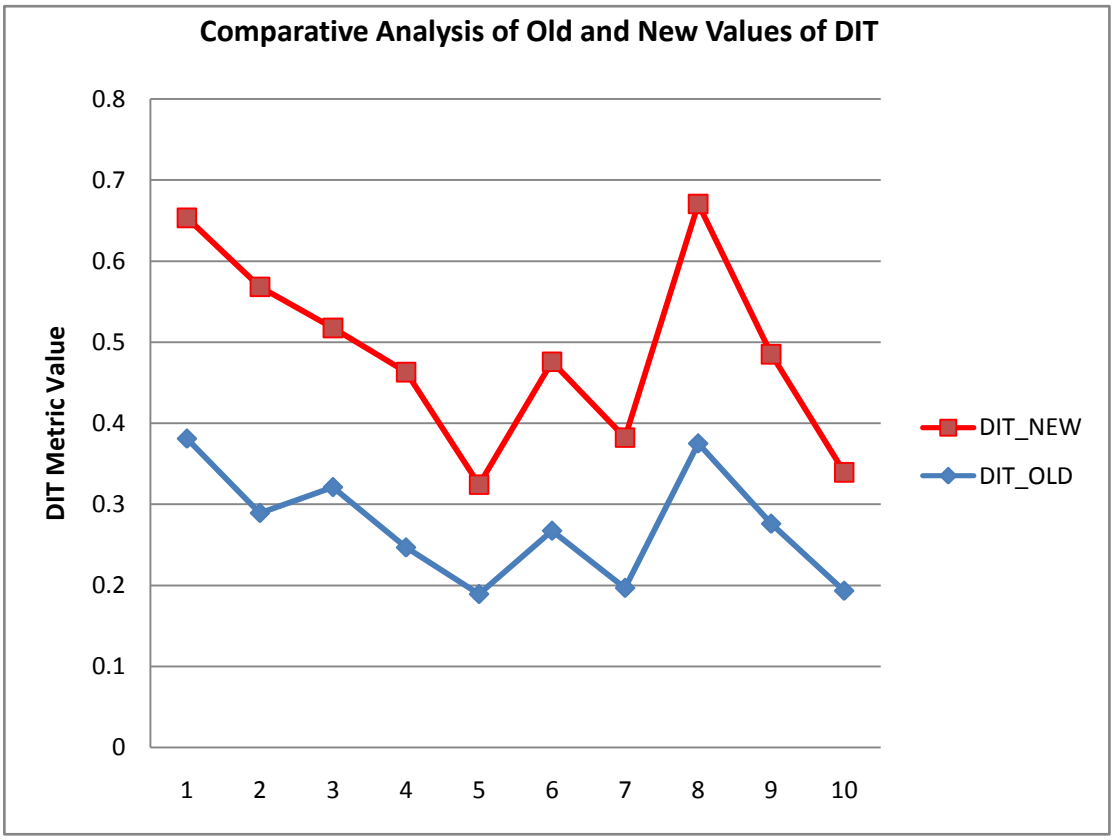

Figure 5.3.2 (a): Comparative Analysis of Old and New Value of DIT Metric

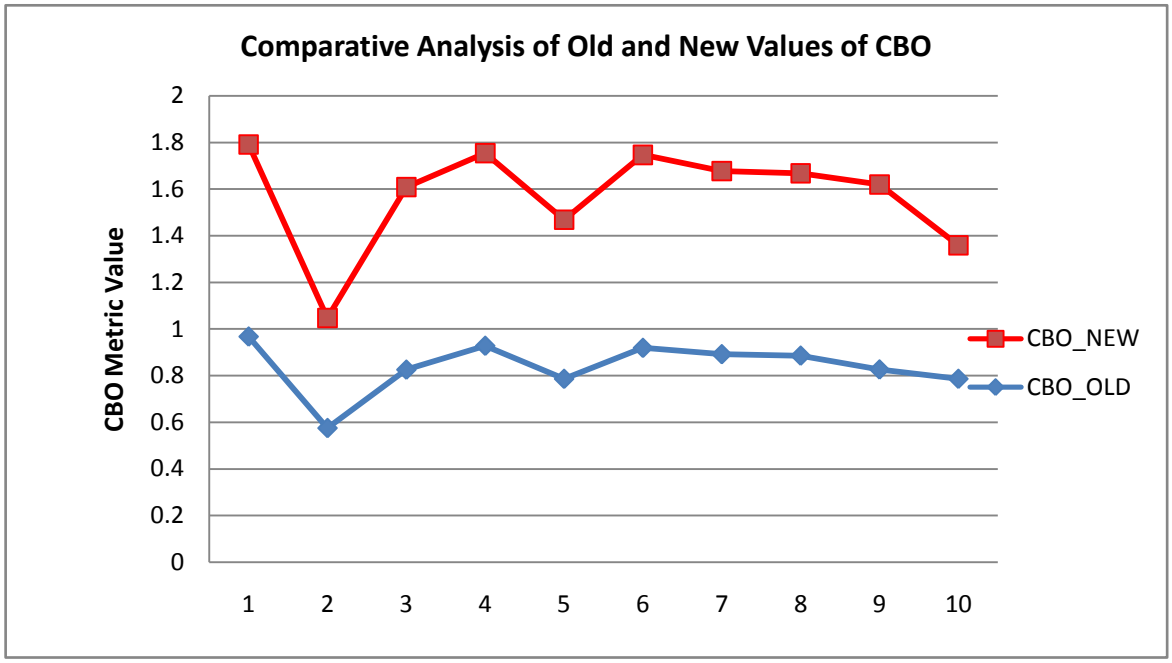

Figure 5.3.2 (b): Comparative Analysis of Old and New Value of CBO Metric

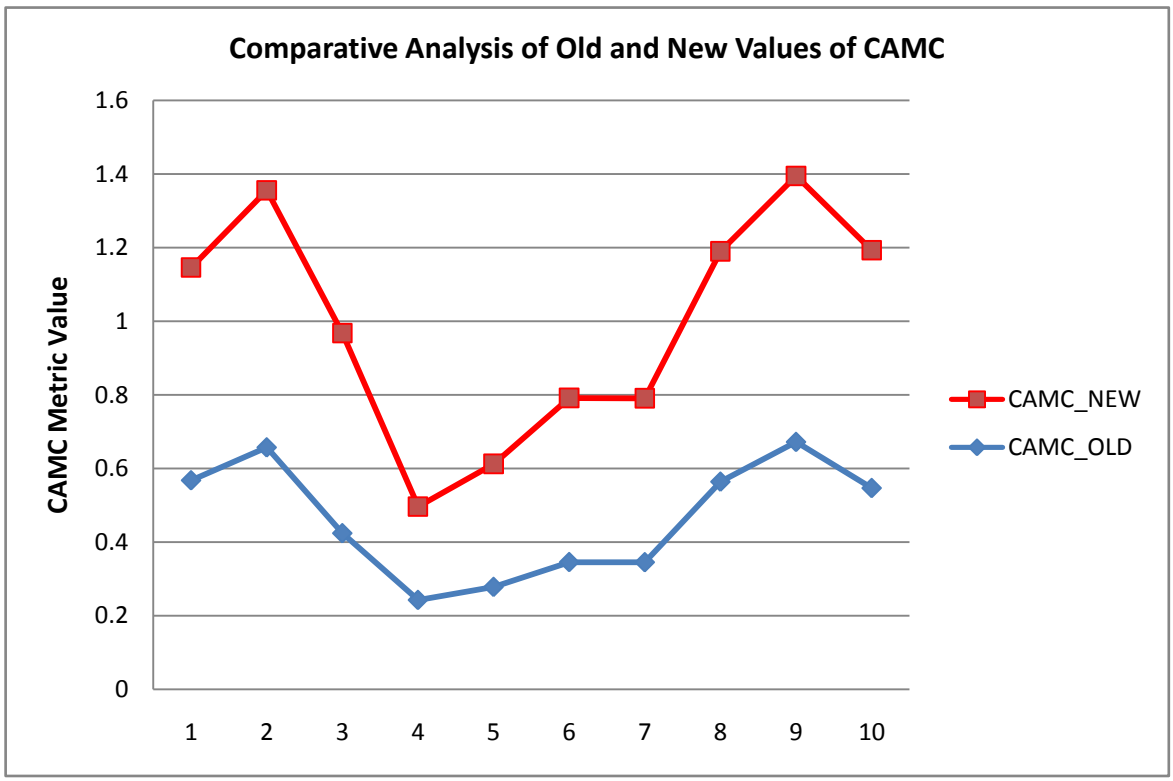

Figure 5.3.2 (c): Comparative Analysis of Old and New Value of CAMC Metric

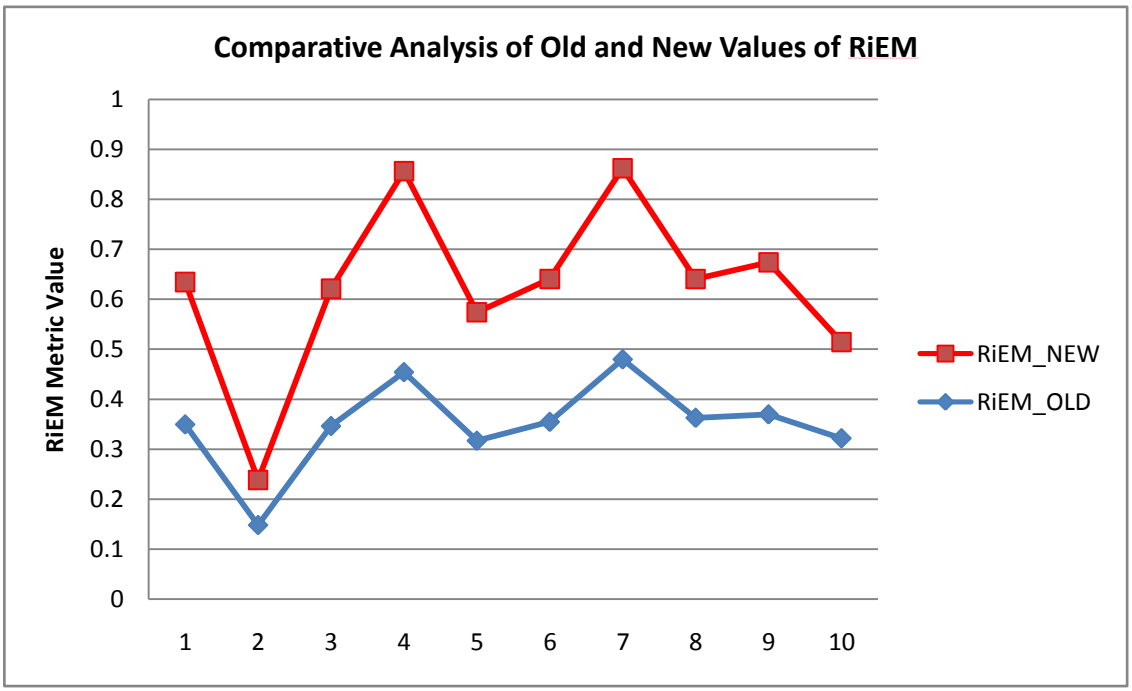

Figure 5.3.2 (d): Comparative analysis of Old and New Value of Rigidity Metric (RiEM)

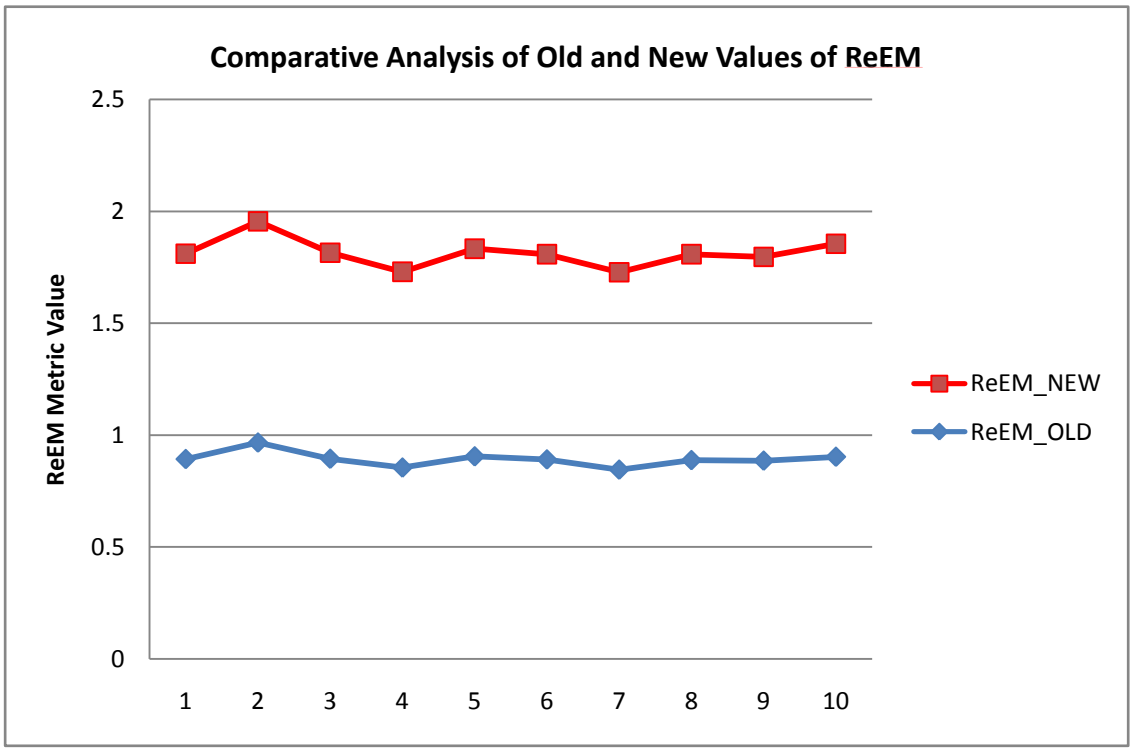

Figure 5.3.2 (e): Comparative analysis of Old and New Value of Reliability Metric (ReEM)

(i) Level of Significance of DIT

To find out the significance of the difference between the means of old DIT values and new DIT values, the means of both old and new values are calculated as shown in table 5.3.2 (a) Pearson Coefficient of correlation comes out to be 0.845900. The coefficients show that the old value of DIT before treatment and new values of DIT after [RI]GOOD treatment are highly correlated. The degree of freedom for both DIT value is 9. For application of the t-test in the scenario, homogeneity of variance F must be tested. The value of homogeneity can be obtained by dividing the larger value with the lower variance. The large variance is 0.004962 for old DIT and the smaller one is 0.002916 for new DIT metrics. The larger to smaller ratio yields F value as 1.701353. Therefore, F value is less than 4.03 (The F critical value for 2 variances of degree of freedom 9), it can be concluded that the variance are homogeneous. This test provides the ground for applicability of t-test. The t-values come out to be -4.930187. As the value exceeds the t critical value of 2.262 for a two tailed test at the 0.05 level for 9 degree of freedom, the null hypothesis H_{01} is strongly rejected and the alternate hypothesis H_{11} is accepted. Therefore, it is validated that **rigidity of object oriented design can be mitigate by minimizing inheritance in object oriented design.**

Table 5.3.2 (a): Computation of Statistical Significance of DIT

Statistical Values / Metric Values	Mean	Std. deviation	Variance	No of Samples (N)	Pearson Coefficient of Correlation	Degree of Freedom	Test for Homogeneity of Variances (F)	T-Value
DIT (Old)	.273560	.0704388	0.004962	10	0.845900	9	1.701353 (<4.03) Therefore variance are homogeneous	-4.930187
DIT (New)	.214350	.0540026	0.002916	10				

(ii) Level of Significance of CBO

To find out the significance of the difference between the means of old CBO values and new CBO values, the means of both old and new values are calculated as shown in table 5.3.2 (b) Pearson Coefficient of correlation comes out to be 0.9112. The coefficient shows that the old value of CBO before treatment and new values of CBO after [RI]GOOD treatment are highly correlated. The degree of freedom for both CBO value is 9. For application of the t-test in the scenario, homogeneity of variance F must be tested. The value of homogeneity can be obtained by dividing the larger value with the lower variance. The large variance is 0.014852 for old CBO and the smaller one is 0.0012410 for new CBO metrics. The larger to smaller ratio yields F value as 1.196823. Therefore, F value is less than 4.03 (The F critical value for 2 variances of degree of freedom 9), it can be concluded that the variance are homogeneous. This test provides the ground for applicability of t-test. The t-value comes out to be -6.565808. As the value exceeds the t critical value of 2.262 for a two tailed test at the 0.05 level for 9 degree of freedom, the null hypothesis H_{02} is strongly rejected and the alternate hypothesis H_{12} is accepted. Therefore, it is validated that **rigidity of object oriented design can be mitigate by minimizing coupling in object oriented design.**

Table 5.3.2 (b): Computation of Statistical Significance of CBO

Statistical Values / Metrics Value	Mean	Std. deviation	Variance	No of Samples (N)	Pearson Coefficient of Correlation	Degree of Freedom	Test for Homogeneity of Variances (F)	T-Value
CBO (Old)	0.839200	0.111399	0.012410	10	0.9112	9	1.196823 (<4.03) Therefore variance are homogeneous	6.565808
CBO (New)	0.734930	0.121869	0.014852	10				

(iii) Level of Significance of CAMC

To find out the significance of the difference between the means of old CAMC values and new CAMC values, the means of both old and new values are calculated as shown in table 5.3.2 (C). Pearson Coefficient of correlation comes out to be 0.969700. The coefficient shows that the old value of CAMC before treatment and new values of CAMC after ^{RI}GOOD treatment are highly correlated. The degree of freedom for both DIT value is 9. For application of the t-test in the scenario, homogeneity of variance F must be tested. The value of homogeneity can be obtained by dividing the larger value with the lower variance. The large variance is 0.026857 for old CAMC and the smaller one is 0.024624 for new CAMC metrics. The larger to smaller ratio yields F value as 0.916881. Therefore, F value is less than 4.03 (The F critical value for 2 variances of degree of freedom 9), it can be concluded that the variance are homogeneous. This test provides the ground for applicability of t-test. The t-values come out to be 5.311179. As the value exceeds the t critical value of 2.262 for a two tailed test at the 0.05 level for 9 degree of freedom, the null hypothesis H_{03} is strongly rejected and the alternate hypothesis H_{13} is accepted. Therefore, it is validated that **rigidity of object oriented design can be mitigate by maximizing cohesion of object oriented design.**

Table 5.3.2 (C): Computation of Statistical Significance of CAMC

Statistical Values / Metrics Value	Mean	Std. deviation	Variance	No of Samples (N)	Pearson Coefficient of Correlation	Degree of Freedom	Test for Homogeneity of Variances (F)	T-Value
CAMC (Old)	0.464370	0.156922	0.024624	10	0.969700	9	0.916881 (<4.03) Therefore variance are homogeneous	5.311179
CAMC (New)	0.524080	0.163880	0.026857	10				

(iv) Level of Significance of RiEM

To find out the significance of the difference between the means of old RiEM values and new RiEM values, the means of both old and new values are calculated as shown in table 5.3.2 (d). Pearson Coefficient of correlation comes out to be 0.9657. The coefficient shows that the old value of RiEM before treatment and new values of RiEM after ^{RI}GOOD treatment are highly correlated. The degree of freedom for both RiEM value is 9. For application of the t-test in the scenario, homogeneity of variance F must be tested. The value of homogeneity can be obtained by dividing the larger value with the lower variance. The large variance is 0.007883 for old RiEM and the smaller one is 0.007778 for new RiEM metrics. The larger to smaller ratio yields F value as 1.013409. Therefore, F value is less than 4.03 (The F critical value for 2 variances of degree of freedom 9), it can be concluded that the variance are homogeneous. This test provides the ground for applicability of t-test. The t-values come out to be -10.277473. As the value exceeds the t critical value of 2.262 for a two tailed test at the 0.05 level for 9 degree of freedom, the null hypothesis H_{04} is strongly rejected and the alternate hypothesis H_{14} is accepted. Therefore, it is validated that **rigidity of object oriented design can be mitigating by minimizing rigidity of object oriented design.**

Table 5.3.2 (d): Computation of Statistical Significance of RiEM

Statistical Values / Metric Value	Mean	Std. deviation	Variance	No of Samples (N)	Pearson Coefficient of Correlation	Degree of Freedom	Test for Homogeneity of Variances (F)	T-Value
RiEM (Old)	0.350480	0.088784	0.007883	10	0.9657	9	1.013409 (<4.03) Therefore variance are homogeneous	-10.277473
RiEM (New)	0.275090	0.088194	0.007778	10				

(v) Level of Significance of ReEM

To find out the significance of the difference between the means of old ReEM values and new ReEM values, the means of both old and new values are calculated as shown in table 5.3.2 (e). Pearson Coefficient of correlation comes out to be 0.9658. The coefficient shows that the old value of ReEM before treatment and new values of ReEM after ^{RI}GOOD treatment are highly correlated. The degree of freedom for both DIT value is 9. For application of the t-test in the scenario, homogeneity of variance F must be tested. The value of homogeneity can be obtained by dividing the larger value with the lower variance. The large variance is 0.001051 for old ReEM and the smaller one is 0.001037 for new ReEM metrics. The larger to smaller ratio yield F value as 0.135147. Therefore, F value is less than 4.03 (the F critical value for 2 variances of degree of freedom 9), it can be concluded that the variance are homogeneous. This test provides the ground for applicability of t-test. The t-values come out to be 10.296300. As the value exceeds the t critical value of 2.262 for a two tailed test at the 0.05 level for 9 degree of freedom, the null hypothesis H_{05} is strongly rejected and the alternate hypothesis H_{15} is accepted. Therefore, it is validated that **reliability of object oriented design can be improved by mitigating rigidity of object oriented design.**

Table 5.3.2 (e): Computation of Statistical Significance of ReEM

Statistical Values / Metric Value	Mean	Std. deviation	Variance	No of Samples (N)	Pearson Coefficient of Correlation	Degree of Freedom	Test for Homogeneity of Variances (F)	T-Value
ReEM (Old)	0.893040	0.032422	0.001051	10	0.9658	9	0.135147 (< 4.03) Therefore variance are homogeneous	10.296300
ReEM (New)	0.920580	0.032201	0.001037	10				

5.4 Conclusion

Any model or approach is acceptable by the society or industry depends upon validation of that methodology or model. It is the validation which proves the usefulness of the methodology in society or industry. All the calculations of metrics and models are done with the use of SPSS software. For testing the usefulness of the framework for defect mitigation of an object oriented design and applying ^{RI}GOOD, a systematic validation is carried out. Initially, for the purpose of theoretical validation, expert review is conducted. Validation involves pre tryout and tryout. Pre tryout involves a small set of data whereas tryout involves a large set of data. The pre tryout is carried out on an OSCAR object oriented design. The result of the metrics values before applying to the ^{RI}GOOD treatment and after ^{RI}GOOD treatment indicates that the reliability improvement guidelines effectively mitigate the rigidity of object oriented design and improve the reliability of object oriented design. A productive pre tryout extends the next stage i.e. tryout.

The tryout was carried out on ten object oriented designs. These designs were initially analyzed and the metric values were computed. The designs are then treated by ^{RI}GOOD and the metric values are computed. The metric and model values for pre treatment and post treatment were undergone statistical analysis to establish the fact that ^{RI}GOOD treatment has successfully mitigate the rigidity defect of object oriented design and improved the reliability of object oriented design.

The t-test was carried out and it was found that the t-values obtained by calculation performed on old and new metric values were excessive the t-critical values. The t-test was also carried out for testing the level of significance between rigidity estimation model and reliability estimation model after applying ^{RI}GOOD and it was found that the t-values obtained by computing performed on models new values were exceeding the t-critical values. Hence, the null hypothesis formulated at the beginning of statistical analysis rejected and alternate hypothesis were accepted. Therefore, our claim that ^{RI}GOOD are able to mitigate the rigidity defect by improving the reliability of object oriented design proved to be correct.

CHAPTER 6: CONCLUSION AND FUTURE WORK

CHAPTER 6: CONCLUSION AND FUTURE WORK

A person who never made a mistake never tried anything new.

-Albert Einstein-

6.1 Background

Software reliability is the essential factor to estimate the software quality as well as to estimate software cost. Software reliability is defined as the probability of failure free operation for a specified period in a specified environment used by the user. Software reliability is generally accepted as a key factor of software quality since it quantifies software failure that makes the system dead. Failure turns the system to perform a required function according to its specification. The cause of failures is the existence of faults and defects in the program. These faults or defects are the errors, which are carried out by the computer programmer. A change in a single bit of a program may lead to wrong output. Software failures occurred due to software design errors. These design errors are the important part because of erroneous specification, immature program coding, insufficient testing and erroneous usage. Therefore, it is required to analyse the reliability of the metrics under different factors that can affect the reliability factor of software.

The demand of science and technology is the high quality software and hardware for making improvements and come-out[1]. The appearance of any industry like automobile, crude oil, banking, railway, aviation, restaurant, home-appliances all these are highly dependent on these software for basic requirements[89]. Due to increase dependency on software systems; size, complexity and defects of software intensive system have grown exponentially. Due to high increase no of defects, the reliability of the software decreases. Literature survey found that most the researcher has done defect mitigation at the code level or after implementation/delivery of the software. From last two decades there is not any research found who have consider factors to improve reliability by using design defect and object oriented design constructs. In chapter 2, literature review shows that various incidence reported due to existence of design level defect in software, complete software is crash. Most of the incidences happened because of unreliable delivery of software. Thirty reliability factors have been identified during the research work, among which design defect has been taken as a major factor to work in software reliability. Still, there is not any methodology to

mitigate defect at early stage of software development. It is challenging task, for software developer to improve reliability.

It is evident from the literature survey for estimating and improving reliability of object oriented constructs on the quality of the software[90-93]. Hence, it is feasible to carry out the same study on reliability because reliability is an important attribute of quality. By analyzing the feasible solution of the problem, the proposed research has been carried-out to provide the needs. A framework for reliability estimation of an object-oriented design has been introduced. The framework provides the methodology for mitigating the design defect with the support of object oriented design constructs like inheritance, cohesion and coupling to improve the reliability of an object-oriented design.

The reliability improvement framework is implemented for identifying the relevant reliability factors, object oriented design constructs and design defects as a major factor for software reliability. Identified object oriented design constructs are verified against the identified reliability factor. The result of the mapping decides whether a particular design constructs has a positive or negative impact on design defect. Based on the decision, three metrics namely DIT, CBO and CAMC identified. Rigidity Estimation Model (RiEM) and Reliability Estimation Model (ReEM) are proposed to estimate defect and reliability. Analysis of the object oriented constructs and design defect gives way to implement reliability improvement guidelines for OOD (^{RI}GOOD). Any proposed methodology, model and framework are accepted only when it is validated. Validation makes any one to believe in the result of any methodology[81]. Validation of the framework is performed by using t-test and Pearson coefficient correlation is carried-out to determine its level of significance. Statistical data shows that the framework is acceptable.

6.2 Major Findings

From the exhaustive literature survey on software reliability of object oriented design construct and design defect the researcher has made the following contributions:

- ***Development of Reliability Improvement Framework:*** The reliability estimation framework for an object-oriented design proposed in chapter 3. The framework mitigates the defects by using the design constructs at the design phase of object-oriented software. Reliability estimation framework for OOD contains four

phases (Identification Phase, Mapping Phase, Measurement Phase and Improvement Phase). Object oriented design constructs, software reliability factors and design defects have been placed at the identification phase. Second phase is the mapping phase, mapping between object-oriented design constructs and design defects, mapping between design defect and reliability and their impact on each other, have been established. Measurement of defect and reliability have been done by using the rigidity estimation model and reliability estimation model, in the measurement phase. In the last phase, reliability improvement guidelines are implemented for object-oriented design. Finally, the whole approach is revised on the basis of suggestions received from the reviewer and experts.

- ***Development of Reliability Estimation Model:***

Reliability estimation model maps the reliability with defect of an object oriented design. The proposed reliability estimation model is used to compute reliability of an object oriented design. The model discussed thoroughly in chapter 4. It was found that defect affects reliability of the design, i.e. presence of defect decreases the reliability of the object oriented design. The implications made by analyzing the reliability estimation model (ReEM).

- ***Development of Rigidity Estimation Model:***

Rigidity estimation model estimates the defect of an object oriented design. It correlates the defect with object oriented design constructs including inheritance, cohesion and coupling. The identified object oriented design metrics are used to compute defect of an object oriented design. The contribution is thoroughly discussed in chapter 4. It has been found that object oriented design metrics compute design defects. It has also been found that whether an object oriented design construct has positive or negative impact. High value of DIT and CBO increase the rigidity defect and high value of CAMC decrease the rigidity defect which increases the reliability of the object oriented design as discussed in chapter 5. These implications are made by analysing the rigidity estimation model (RiEM).

- ***Development of Reliability Improvement Guidelines for OO Design (^{RI}GOOD):***

Reliability improvement guidelines for object oriented design have been proposed from the analysis of the metrics (DIT, CBO and CAMC) and models (RiEM),

(ReEM). ^{RI}G1 and ^{RI}G2 states that a more reliable design can be developed by keeping the value of DIT and CBO for the object oriented design closure to its minimum limit to 0. ^{RI}G3 and ^{RI}G4 states that a more reliable object oriented design can be developed by keeping the value of CAMC close to its maximum limit i.e. approaching to 1. In the light of the proposed reliability improvement guidelines ^{RI}Gs, changes are made to the designe so that a more reliable object oriented design can be produced by minimizing rigidity defect of the OOD.

6.3 Significance of the Findings

The role of software in society always increases and leads a series of tragedy and conditions caused by its failures. Unreliability of the software shows that the presence of defect in the software may be simple a matter of life and death at the time when software fails to perform. Various incidents are discussed in chapter 2. There is not any methodology to measure the accuracy of failures and design defects in the software of the design phase remains a difficult problem[98]. Still, there is not any accurate definition "What are the factors that directly affect the software reliability". It is difficult to find the precise direction to estimate software reliability measurement becomes important for software industry, developers and users[151]. Therefore, a study carried out on improving reliability of OOD by mitigating design defect, with its direct contribution to the field of knowledge, may prove to be significant directly or indirectly in terms of following:

- It may help to improve both the design and architecture information of the software system, which in turn may ease the process of development and maintenance as well.
- It may help to identify and mitigate the design defect in the software design at the early stage of software development life cycle to produce reliable software.
- It may help to quantify the reliability of software and furnish the cost estimation of a software project, which facilitate the estimation and planning of new activities.
- It may help to determine the involvement of object oriented technology with the mitigation of design defects.
- It may also help the designer to develop various alternative designs of the same object oriented software.

- It may also help to determine which one of the two versions of object oriented software design is more reliable.
- It may help to find out which object-oriented design has more defects among designs of different software.

Moreover, it is also observed that the contribution of this successful proposed study may provide to be specifically significant in the manner:

- The proposed model based approach may promote and enable the developers to come out with good reliability models with better acceptability.
- The proposed model may provide the basis for the development of new modified or refined approaches.
- The proposed model may find a place among the measurement tools for reliability evaluation of object-oriented software.
- The proposed methodology may encourage other researchers to undertake the development of other new model based approaches.
- The proposed approach may be used to set the benchmark value for reliability estimation by the organization.

6.4 Answers to Research Questions

The various research questions may placed at the beginning of the research work are answered separately by the research findings in the following section:

Research Question: What are the factors that directly influence the reliability of an object oriented software design?

Research Findings: Design Defect and Object Oriented Design Constructs directly influence the reliability of an object oriented software design.

Research Question: Is design defect a reliability factor?

Research Findings: The researcher has identified more than 30 reliability factors among which researcher found that design defect is one of the important factors affecting reliability of object oriented design. Thus, the researcher has taken defect as a major factor of software reliability.

Research Question: Is there any standard framework available to develop reliability improvement of object-oriented design.

Research Findings: There is not any standard framework available to develop reliability improvement model of an object-oriented design.

Research Question: Is there any OO design metric or model available, which can be used in the early phase of software development life cycle to measure its reliability?

Research Findings: No estimation model is available to compute reliability with the help of rigidity.

Research Question: Can we develop an approach that may be used in the design phase to estimate the reliability of an object oriented software?

Research Findings: With the help of literature survey, the researcher has found that it is feasible to develop an approach that may be used in the design phase to estimate the reliability of object oriented software.

Research Question: Can effort of estimating the design defect of OOD be reduced by using good measurement tools?

Research Findings: Yes, effort of estimating the defect of OOD is reduced, through good measurement tools.

Research Question: Is the proposed approach to quantify object oriented design defect help to compare the reliability of various alternative designs?

Research Findings: The proposed reliability estimation model estimate reliability with the help of proposed rigidity estimation model. On the basis of the result, reliability of various alternative designs can be compared.

Research Question: Can a proposed approach be useful without assuring theoretical and empirical validity?

Research Findings: The proposed reliability estimation model along with reliability improvement guidelines have been theoretically and statistically validated.

Research Question: How general are the lessons learned in this study? Can they be applied in situations involving other metrics and models, or to organizations, which have different operational contexts?

Research Findings: The study has proposed a reliability estimation framework for an object-oriented design, using which various metrics and models can be developed. The metrics and models analysis yields reliability improvement guidelines, which can be applied to the design of all object oriented software regardless of organizations or operational contexts.

6.5 Future Direction

Research is an on-going process. Reaching one milestone promote the way to the next. As a future research plan, there may the following tasks to be performed:

- The methodology proposed for the computation may be automated so that the models computation can be done automatically and hence not to become tedious for large software.
- Researcher is planning to conduct more experiments on industrial data to draw more concrete conclusions about the implications of the metrics and models proposed.
- The reliability estimation framework for an object oriented design defect can be modified so that it can help to mitigate the defect and improve the reliability of any software design.
- Different execution of the reliability estimation framework for an object oriented design defects are possible. Researchers may implement the same framework by choosing another set of reliability factors and object oriented design constructs to improve reliability.

6.6 Conclusion

Estimation of reliability in the design phase has become repeatedly recommended. The researcher has deal with the major issues in the areas of software reliability. A reliability improvement framework for an object oriented design has been proposed. The objective of the framework is to minimize design defects and improve the reliability of the object oriented design by mitigating the object oriented design constructs and design defects. For the purpose, the framework incorporates design defect of an object oriented design as well as its defect measurement. The proposed framework is also implemented by the researcher. Implementation of the framework give way to implement reliability estimation model (ReEM), rigidity estimation model (RiEM) and reliability improvement guidelines for object oriented design (^{RI}GOOD).

The proposed framework has reflecting the impact of each object oriented design constructs (i.e. Inheritance, Cohesion and Coupling) on Rigidity Estimation Model (RiEM) of an OOD. Reliability estimation model (ReEM) reflects the impact of defect on the reliability of an object oriented design. The proposed framework has executed in three-fold: identification of object oriented design defect, mitigation of design defect and then reliability improvement guidelines has been developed that mitigate the design defect and improve the reliability of the object oriented design. With the help of identified object oriented design metrics, the design defect is mitigate

and improved the reliability of the object oriented design after implementing the guidelines is verified and reviewed. In addition, the framework can also be used to measure OOD according to the reliability or design defect of an object-oriented design. Successful validation of the framework has also manifested its satisfaction.

REFERENCES

1. M. R. Lyu, “Handbook of software reliability engineering”, IEEE Computer Society Press, Los Alamitos, California, Washington Brussels Tokyo, April 1996, pp. 1-26, ISBN: 0070394008.
2. S. R. Dalal, M. R. Lyu, and C. L. Mallows, “Software Reliability”, Available at:http://onlinelibrary.wiley.com/doi/10.1002/9781118445112.stat05035/abstract.
3. M. L. Shooman, “Yes, software reliability can measure and predicted”, Proceedings of the 1987, Fall Joint Computer Conference on Exploring Technology, IEEE Computer Society, 1987, ISBN: 0-8186-0811-0.
4. G. J. Myers, “The art of software testing”, Wiley New York, 1979.
5. IEEE Standard Glossary of Software Engineering Terminology, ANSI/IEEE Standard 729, 1991.
6. J. D. Musa, “Software reliability: measurement, prediction, application”, Professional Edition, McGraw Hill Publishers, New York 1988.
7. J. D. Musa, “Software reliability engineering: more reliable software faster and cheaper”, McGraw Hill, 2004.
8. J. L. Lions, “Ariane-5 Flight-501, Failures-Report by the enquiry Board”, 2010, Available at: http://www.di.unito.it/damiani/ariane5rep.html.
9. M. Fowler, “Refactoring: Improving the Design of Existing Code”, Addision-Wesley, New York, USA, ISBN-13: 9780201485677, pp. 431.
10. T. Mens, and T. Tourwe, “A Survey of Software Refactoring”, IEEE Transactions, Software Engineering, Vol. 30, pp. 126-139.
11. Y Shu, H. Liu, Z. Wu, and X. Yang, “Modeling of software faults detection and correction process based on the correction log”, Information Technology, Vol. 8, pp. 735-742.
12. A. J. Riel, “Object oriented design heuristics”, Addision-Wesley Longman Publishing Co. Inc. Boston, MA, USA, pp. 379, ISBN-13: 9780201633856.
13. W.J. Brown, “Anti-Patterns: refactoring software”, Architecture and Projects in Crisis Wiley, USA, 1998, pp. 309, ISBN-13: 9780471197133.
14. R. Subramanyam, M.S Krishnan, “Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects”, IEEE Transactions on Software Engineering, 2003, Vol. 29, No. 4, pp. 297-310.
15. M. Fowler, “Refactoring: Improving the design of existing code”, Addision-Wesley, New York, USA, 1999.

16. A. Bertolino, “Software testing research: achievements, challenges, dreams”, Future of Software Engineering, 2007. FOSE '07, 23-25 May 2007, pp. 85-103.

17. N. Bettenburg, W. Shang, W. Ibrahim, B. Adams, Y. Zou, and A. E. Hassan, “An empirical study on inconsistent changes to code clones at the release level” Sci. Comp. Program, 2012, Vol. 77, pp. 760-776.

18. V. R. Basili, “The empirical investigation of perspective-based reading”, University of Maryland, UK, 1996, pp.76.

19. H. Sanatnama, and F. Brahimi, “Graph drawing algorithms: Using in software tools”, Journal of Applied Science, Vol. 10, pp. 1994-2001.

20. M. M. Mantyla, and C. Lassenius, “What types of defects are really discovered in code reviews?” IEEE Transaction Software Engineering, Vol. 35, pp. 430-448, 2009.

21. N. Y. Moha, L. Gueheneue, L. Duchien, and A. F. L. Meur, “Décor: A method for the specification and detection of code and design smells”, IEEE Transaction Software Engineering, Vol. 30, pp. 126-139, 2010.

22. D. Budgen, “Software Design”, Second Edition, Addison-Wesley. 2003.

23. D. E. Perry, and A. L. Wolf, “Foundation for the Study of Software Architecture”, SIGSOFT Software Eng. Notes, Vol. 17, No. 4, October 1992, pp. 40-52.

24. R. Subramanyam, M.S. Krishnan, “Empirical Analysis of CK Metrics for Object-oriented design Complexity: Implications for Software Defects”, IEEE Transactions on Software Engineering, Vol. 29, No. 4, 2003, pp. 297-310.

25. M. R Lyu, “Software Reliability Engineering: A Road Map” Future of Software Engineering, pp. 153-170, 2007.

26. V. Basili, “Qualitative software complexity models: A Summary”, In Tutorial on models and methods for software management and Engineering, IEEE, Computer society press, Los Alamitos, CA, 1980.

27. P. Jalote, B. Murphy, M. Garzia, and B. Errez, “Measuring Reliability of Software Products” IEEE, pp. 1-8, 2004.

28. D. R. Levinson, “Hospital incident reporting systems do not capture most patient harm”, January 2012.

29. T. Illes-Seifert, and B. Paech, “Exploring the relationship of a file’s history and its fault-proneness: An empirical method and its application to open source programs”, Practice and Research Techniques Testing: Academic & Industrial Conference, Information and Software Technology, Vol. 52, 2010, pp. 539 – 558.

30. J. Grundy, "Aspect-oriented requirements engineering for component based software systems", Proceeding of 4th International symposium on Requirement Engineering Limerick, Ireland, 7-11 June 1999, RE 99, IEEE Computer Society, Washington, DC, 1999.

31. W. J. Brown, R. C. Malveau, W. H. Brown, H. W. McCormick III, and T. J. Mowbray, "Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis", 1st ed. John Wiley and Sons, March 1998.

32. N. Fenton and S. L. Pfleeger, "Software Metrics: A Rigorous and Practical Approach", 2nd ed. London, UK: International Thomson Computer Press, 1997.

33. Ladan Tahvildari, and Kostas Kontogiannis, "A Metric-Based Approach to Enhance Design Quality through Meta-Pattern Transformations", Proceedings of the Seventh IEEE European Conference on Software Maintenance and Reengineering (CSMR'03), 2003, pp. 183-192.

34. R.A. Khan, K. Mustafa, and S.I. Ahson, "Operation Profile-A key factor for reliability estimation", Universities Press, Gautam Das and V. P. Gulati (Eds), CIT, 2004.

35. C. Wohlin, "Estimation of Software Reliability Growth Models", Blekinge Institute of Technology, Sweden, 2007, pp. 1-3

36. F. B Abreau and R. Carapuca, "Candidate metrics for Object Oriented software within a taxonomy framework" Journal of Systems and Software, Vol. 26, No 1, July 1994, pp.1-6.

37. D. Arora, P. Khanna, A. Tripathi, S. Sharma and S. Shukla, "Software quality estimation through object oriented design metrics" IJCSNS, International Journal of Computer and Network Security, Vol.11, No.4, April 2011, pp. 100-104.

38. R. A. Khan, "Quality Estimation of Object Oriented Code: A Design Metrics Perspective" Ph.d Thesis, 2004, 2004, pp. 437-444.

39. M. Fowler, K. Beck, J. Brant, W. Opdyke and D. Roberts, "Refactoring Improving the Design of existing Code: Reading" Massaclusetts: Addission Wesley, 1999.

40. F. B Abreu, M. Goulao and R. Esteves, "Towards the design quality evaluation of object oriented software systems", Proceedings of the 5th International Conference on Software Quality, Austin Texas, 23-26 Oct, 1995, pp. 44-57.

41. M. H. Tang, M. H. Kao and M. H Chen, "An Empirical study on object oriented metrics", Software Metrics Symposium, IEEE Computer Proceedings, 1999, pp. 242-249.

42. J Bansiya and L. Etzkorn, C. Davis, and W. Li, "A Class Cohesion Metrics for Object Oriented Design", The Journal of Object Oriented Programming, Vol. 11, No. 8, 1999, pp. 47-52.

43. J Bansiya and C. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment", IEEE Transaction on Software Engineering, Vol. 28, No. 1, 2002, pp. 4-17.

44. M. Lorenz and J. Kidd, "Object Oriented Software Metrics: A Practical Guide", Prentice Hall, Englewood Cliffs, New Jersey, 1994.

45. W. Li. S. Henry, "Object Oriented Metrics that Predict Maintainability" Journal of Systems and Software, Vol. 23, No. 2, 1993, pp. 111-112

46. T. Nikolaos, and C. Alexander, "Predicting the Probability of Change in Object-Oriented Systems" IEEE Transactions on Software Engineering, Vol. 31, No.7, July 2005, pp. 601-614.

47. M. R. Lyu, "Software Reliability: to use or not to use?", 5th International Symposium on software reliability engineering, 1994. Proceedings, 6-9 November 1994, pp. 66-73.

48. H. Liu, L. Yang, Z. Niu, Z. Ma, and W. Shao, "Facilitating software refactoring with appropriate resolution order of bad smells", Proc. of the ESEC/FSE '09, pp. 265–268.

49. **P. K. Chaurasia and R. A Khan, "Classification of Software Requirement Errors: A Critical Review", International Journal of Computer Applications (IJCA), Vol. 132, No-07, pp. 09-15, 2015.**

50. Y. Singh, A. Kaur and R. Mehlotara, "Empirical validation of object-oriented metrics for predicting fault proneness models", Software Quality Journal, Springer, Science Business Media, 2009, pp. 1-35.

51. N. Sharygian, J. C Browne and R. P Kurshan, "A Formal Object Oriented Analysis for software reliability design for verification" 2011, pp. 01-15. Available at : https://www.cs.utexas.edu/users/browne/NewPapers/FASE20011.pdf

52. C. Y. Huang, S. Y. Kuo, M. R. Lyu, and J. H. Lo, "Effort index based software reliability growth models and performance assessment", Available at: www.cse.cuhk.edu.hk/~lyu/paper_pdf/A13-07_huang.pdf.

53. W.W. Schilling, M. Alam, "Modeling the reliability of existing software using static analysis" 2006, IEEE International Conference on Electro/Information Technology, 7-10 May 2006 pp. 366-371. Available at: ieeexplore.ieee.org/ document/4017728.

54. N. J. Faust, “Reliability Analysis by use of Markov Modeling”, RAC Technical Brief, March 1989. Available at : Studylib.net/doc/18130573/the-applicability-of-markov-analysis-methods-to-reliability.

55. K. Khosravi and Y. G. Aelucheneuc, “A quality model for design patterns”, 2004, pp. 1-107. Available at: www.ptidej.net/publications/Research\report \Quality\ Models\ September04.doc.

56. A. Yadav and R.A Khan, “Bridging the gap between design constructs and reliability factors”, CSI Communication, Vol. 33, No. 12 March 2010, pp. 29-32, ISSN- 0970-647X.

57. R.G. Dromey, “A model for software product quality”, IEEE Transactions on Software Engineering, 1995, Vol. 21: pp.146-162.

58. E. Emam and K, Melo, “The Prediction of faulty classes using object oriented design metrics” National research Council of Canada, November 1999, pp. 1-25.

59. A. M. A. Khouri, “Using quality models to evaluate national id systems: The case of the UAE” Proceedings of the World Academy of Science Engineering and Technology, Vol. 21, May 2007, pp. 400-414, ISSN 1307-6884.

60. C. Gygi, B. Williams, N. Decarlo and S. R. Covey, “How to measure defect rate for six sigma”, October 2012, ISBN: 978-1-118-12035-4 Available at : http://www.dummies.com/careers/project-management/six-sigma/how-to-measure-defect-rate-for-six-sigma/

61. K. Rajneesh and V. Bhattacherjee, “Class Inheritance Metrics-an analytical and empirical approach”, 13 September 2007.

62. C. Wohlin, M. Höst, P. Runeson, and A. Wesslén, "Software Reliability", in Encyclopedia of Physical Sciences and Technology (third edition), Vol. 15, Academic Press, 2001. Available at: http://www.wohlin.eu/softrel01.pdf

63. S. Skoulaxinos, “SW-HW Co-design and fault tolerant implementation for the LRID wireless communication system” Proceedings of the First NASA/ESA Conference on Adaptive Hardware and Systems IEEE, 2006, pp:305-308, ISBN: 0-7695-2614-4.

64. S. Chidamber and C. Kermer, “A metrics suite for Object oriented design” IEEE Transaction on software engineering, Vol. 20, No 6, pp. 476-493. 1994.

65. G. Booch, “Object Oriented design and Application”, Benjamin/Cummings, Mento Park, CA, 1991.

66. Dale and H. Van Der Zee, “Software Product Metrics- who needs them”, Proceedings Biometrics, 1992, pp. 31-43.

67. N. Fenton, “Software Measurement: A necessary Scientific Basis,” IEEE Transactions on Software engineering, Vol. 20, No 3, pp, 199-206, March 2006.

68. M. K. Breesam, “Metrics for Object Oriented design focusing on class inheritance metrics”, 2nd International Conference on dependability of Computer Systems, IEEE, 2007, pp. 231-237, ISBN: 0-7695-2850-3.

69. N. E. Fenton, “Software Metrics - A Rigorous Approach”, Chapman & Hall, 1992.

70. A. Macro, J. Buxton, “The Craft of Software Engineering”, Addison-Wesley, 1987.

71. D. A. Troy, S. H. Zweben, “Measuring the Quality of Structured Designs”, Journal of Systems and Software, Vol. 2, 1981.

72. D. Q. Birkmeier, “On the state of the Art of Coupling and Cohesion Measures for Service-Oriented System Design”, Americas Conference on Information Systems (AMCIS), Association for Information systems, 2010, pp. 1-11.

73. T. A. Budd, “An Introduction to object-oriented Programming” Addison Wesley, 1990.

74. D. M Ambros, M. Lanza and R. Robbes, “On the relationship between change coupling and software defects”, 16th Working Conference on Reverse Engineering, 2009, pp. 135-144.

75. **P. K. Chaurasia and R. A. Khan, “Relationship between Object Oriented Design Constructs and Design Defects”, International Journal of Computer Application and Management (IJCAM), Vol.7, No. 9, pp: 35-38. ISBN 2231-1009. Available at: http://ijrcm.org.in/article_info.php?article_id=8018.**

76. Y. Liguo, R. Stephen, K.C. Syhach, and S. Ramaswamy, “Coupling measurement in multi kernel based software with its application to Darwin”, International Journal of Intelligent Control and systems, Vol. 13, No.2, 2008, pp: 109-119.

77. M. Hitz, B.Montazeri, “Measuring Coupling and Cohesion in Object Oriented Systems”, Proceeding of the International Symposium on applied Corporate Computing, Monterrey, 25-27 October 1995, pp. 25-27 Also available at citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.409. 4862 & rep=rep1&type.

78. Li. W., S. Henry "Object-Oriented metrics that predict maintainability", Journal of Systems and Software, Vol. 23, No. 2, pp. 111-122, 1993.

79. A. Nicolaescu, H Lichter and Yi Xu, “Evolution of Object Oriented Coupling Metrics: A Sampling of 25 Years of Research” IEEE/ACM 2nd International workshop on Software Architecture and Metrics (SAM), May 2015, pp. 48-54.

80. J. F Murray, J. Davis, “General Principles of Software Validation”, U.S Department of Health and Humana Services, Vol. 1. No. 1, June 1997, pp. 1-47.

81. P. M. Shanthi and K. Duraiswamy, “An Empirical Validation of software quality metric suites on open source software for fault-proneness prediction in object oriented systems”, European Journal of Scientific Research, Vol. 51, No 2, 2011, pp. 168-181.

82. A. Marcus, D. Poshyvank, “The Conceptual cohesion of Class”, Proceeding of 21st IEEE International Conference of software Maintenance (ICSME’05) IEEE, 26-29 Sep 2005, pp. 133-142.

83. https://www.researchgate.net/publication/3551177Object-oriented_computer animation.

84. R. Lai and M. Garg, “A detailed study of NHPP software reliability models”, Journal of Software, Vol. 7, No.6, June 2012, pp.1296-1306.

85. **P. K . Chaurasia, R. A. Khan, “Exploring Object Oriented Design Metrics”, International Conference on Nano-Science and Nano-Technology, Babasaheb Bhimrao Ambedkar University, Lucknow, 22-24 Sep, 2017**.

86. V. Basili, L. Briand, and W. Melo "A Validation of Object Oriented Design Metrics as Quality Indicators", IEEE Transactions of Software Engineering, Vol. 22, No. 10 , pp. 751-761, 1996.

87. P. M. Shanti, and K. Duraiswamy, “An empirical validation of software quality metrics suites on open source software for fault proneness prediction in object oriented systems” European Journal of Scientific Research, Vol. 51, No.2, 2011, pp.168-181, ISSN:1450-216X.

88. L. Briand, K. E. Emam and S. Moraska, “Theoretical and Empirical Validation of Software Product metrics”, Technical Report (ISERN-95-03), International Software Engineering Research Network, 1995, available at: http://citeseerx.ist.psu.edu/ viewdoc/download? Doi = 10.1.1.98.1176&rep=rep1& type=pdf.

89. B. Littlewood and L. Stringin, “Software reliability and dependability : A Roadmap” International Conference on Software Engineering Proceedings of the Conference on the Future of Software Engineering, ACM New York, NY USA, 2000, pp. 175-188.

90. T. Zimmermann, “Changes and Bugs- mining and predicting development activities”, Proceeding, ICSM 2009, IEEE 2009, pp. 443-446.

91. J. Holloway, “Object-Oriented design taxonomy”, A thesis submitted to University of Wales Aberystwth, 2008, pp. 1-143.

92. C. L. Gray, “A coupling-complexity metric suite for predicting software quality”, A Thesis presented to the faculty of California Polytechnic state University, 2008, pp.1-72.

93. K. Mustafa and R. A Khan, “Quality metric development framework” Journal of Computer Science, Science Publication, Vol. 1, No.3, 2005, pp. 437-444, ISSN 1549-3636.

94. R. K Chowdhary, S. Chandra, A. Agrawal and A. Yadav, “Software fault tolerance techniques”, Proceedings of Advance Computing and Communication Technologies, 2008, pp. 34-43, ISBN: 81-87433-68-X.

95. R.A. Khan, A. Agrawal, “Software Engineering: A Practiners Approach”, Alpha Science International, 2014.

96. S.J. Keene, “Comparing Hardware and Software Reliability, Reliability Review”, Vol. 14. No. 4, December 1994, pp. 05-21.

97. D.S. Herrmann, “Software Safety and Reliability”, IEEE Computer Society Press, Los Alamitos, 1999.

98. L. Shanmugam & L. Florence, “An Overview of Software Reliability Models”, IJARCSSE, Vol. 2, Issue 10, October 2012, pp. 36-42.

99. Z. Jelinski, P. B. Moranda, “Software Reliability Research”, In Statistical Computer Performance Evaluation, W. Freiberger, Ed. New York: Academic, 1972, pp. 465-484.

100. S. Gaudan and G. Auriol, “A new structural complexity metrics applied to object oriented design reliability assessment”, 2007. Available At: www.lattis.univ-toulouse fr/~motet/papers/2007_ISSRE_GMA.pdf.

101. M. Y. Liu and I. Traore, "Empirical relation between coupling and attack ability in software systems: a case study on DOS," in Proceedings of the 2006 Workshop on Programming Languages and Analysis for Security Ottawa. Ontario, Canada: ACM, 2006, pp. 57-64.

102. K. Pemmaraju, “The Quest for Software Quality”, http://www. cigital.com/papers /download/sil-india-dec98-kp.doc.

103. K. K. Agrawal, Y. Singh A. Kaur and R. Malhotra, “Empirical study of object oriented metrics”, Journal of Object Technology, published by ETH Zurich, Vol. 5, No. 8, 2006, pp. 149-173.

104. K. Kaur, H.Singh, “An Investigation of Design Level Class Cohesion Metrics”, The International Journal of Information Technology, Vol. 9, No.1, 2012, pp. 66-73.

105. K. Kaur, H. Singh, “Exploring Design Level Class Cohesion Metrics”, Journal of Software Engineering and Applications (JSEA), 2010, pp. 384-391.

106. B.W. Boehm, J. R. Brown and M. Lipow, “Quantitative Evaluation of software quality”, pp. 592-605. Available at: csse.usc.edu/TECHRPTS/1976/usccse76-501/usccse76-501.pdf

107. H. Astudillo, “Five ontological levels to describe and evaluate software architectures”, Vol. 13, No. 1, pp. 69-76.

108. A. B. Al-Badareen, "Software Quality Models: A Comparative Study" in Software Engineering and Computer Systems, Springer, 2011, pp. 46-55.

109. ISO/IEC25010, “Systems and Software Engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Software product quality and system quality in use models. ISO2010.

110. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. M. Loingtier, J. Irwin, “Aspect-Oriented Programming”, ECCOP’97 Object-Oriented Programming, 11th European Conference, pp. 220-242.

111. I. Castillo, F. Losavio, A. Matteo, and J. Boegh, “Requirements, aspects and software quality: the REASQ model”, J Object Technology, 2010, 9(4), pp. 69-91.

112. ISO/IEC CD 25010, “Software Engineering. Software Product Quality Requirements and Evaluation (SQuaRE). Quality Model and guide, 2011.

113. A. Shaik, C. R. K. Reddy, B. Manda, C. Prakashini and K. Deepti, “An empirical validation of object oriented design metrics in object oriented systems”, Journal of Engineering Trends in Engineering and Applied Sciences (JETEAS), Vol. 1, No.2, pp. 2016-224.

114. J. K. Chhabra, R. K. Challa, “Object oriented inheritance metric-reusability perspective” International Conference on Computing Electronics and Electrical Technologies (ICCEET), 2012, pp. 852-859.

115. I. Bratko and S. Muggleton, “Applications of inductive logic programming” Communication ACM 38, 1995, pp. 65-70.

116. R. C. Martin, "Agile Software Development", Principles, Patterns and Practices, 2002.

117. M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, “Refactoring: Improving the Design of Existing Code”, Object Technology Series, Addison-Wesley, June 1999.

118. J. Perez, C. Lapoez, N. Moha and T. Mens, “A Classification Framework and survey for Design Smell Management”, Technical Report No. IT DI, 2011, pp. 1-31.

119. B. M. Pietrzyk, “Automatic Refactoring of Large Codebases” Thesis, Masaryk University, 2015.

120. M. A. Sharif, W. P. Bond, and T.A Otaiby, “Assessing the Complexity of Software Architecture”, ACMSE, April 2004, pp. 98-103.

121. T. Kamiya, S. Kusumoto, “Prediction of fault proneness at early phase in object oriented development” object oriented real time distributed computing (ISORC’99), 1999, pp. 253-258.

122. S. K. Chandran, A. Dimov, and S. Punnekkat, “Modeling uncertainties in estimation of software reliability – a pragmatic approach” Fourth IEEE International Conference on Secure Software Integration and Reliability Improvement, IEEE Computer Society, 2010, pp. 227-236.

123. L. Cheung, R. Roshandel, N. Medvidovic, and L. Golubchik, “Early prediction of software component reliability”, ACM, 2008, pp. 111-120.

124. J. W. Best, and J. V. Khan, “Research in Education”, 6th edition, PHI Publication, 1992.

125. Z.A. Rahamneh, M. Reyalat, A.F. Sheta, S.B Ahmad, and S.A Oqeili, “A new software reliability growth model: genetic programming based approach”, Journal of Software Engineering and Application, 2011, Vol. 4, pp. 476-481.

126. J. Zeng, J. Li, X. Zeng, W. Luo, “A Prototype system of software reliability prediction and estimation” 3rd International Symposium on intelligent Information technology and security informatics, 2010, pp. 558-561.

127. M. Cristescu and L. Cioviac, “Estimation of the Reliability of distributed applications”, Informatics economics, Vol. 14, no. 4, 2010, pp. 19-21.

128. B. C. Gang, J. C. Hai and C. K. Yuan, “A reliability improvement predictive approach to software testing with Bayesian Method”, Proceedings of the 29th Chinese Control Conference, July 29-31 2010, pp. 6031-6036.

129. J. Holloway, “Object Oriented design taxonomy”, A thesis submitted to University of Wales Aberystwyth, 2008, pp. 1-143.

130. A. P Wood, “Reliability metric varieties and their relationship”, Proceedings annual reliability and maintainability symposium, IEEE, 2001, pp. 110-115.

131. P. Aital, P. Sashikala, "Role of Software Reliability Models in Performance Improvement and Management", Journal of Software Engineering and Applications, 2012, pp. 737-742.

132. A Khandelwal, "Software Reliability Modeling using Fault Tree Analysis and Stochastic Petri Nets", Thesis, 2013, pp. 1-39.

133. H. Li, M. Lu, and Q. Li, "Software Reliability Metrics selecting method based on analytic hierarchy process", Proceedings of the 6th International Conference on Quality Software, IEEE Computer Society, 2006, pp. 337-346.

134. A. Wood, "Predicting software reliability" IEEE, Vol. 29, No.11, November 1996, pp. 69-77. ISSN: 0018-9162.

135. A. L. Reibman and M. Veeraraghavan, "Reliability Modeling: An overview for system design", IEEE Computer Society, Vol. 24, No. 4, 1991, pp. 49-57, ISSN: 0018-9162.

136. M.C. Kim, S.C. Jang, and J. Ha, "Possibilities and limitations of applying software reliability growth models to safety critical software", Nuclear Engineering and Technology, Vol. 39, No. 2, April 2007, pp. 145-148.

137. I. Karanta, "Methods and Problems of software reliability estimation" VTT technical Research Center of Finland, 2006, pp. 1-58.

138. M. R. Lyu, "Applying reliability models more effectively", IEEE software, 1992, pp. 43-52. ISSN: 0740-7459.

139. https://www.tutorialspoint.com/ OOAD - UML Structural Diagrams

140. **P. K Chaurasia, R. A. Khan, "Identification of Error Prone Path for Improving Software Testing: Genetic Algorithm Approach, Conference Proceeding National Conference on Recent Advances in Chemical and Material Sciences, 23-24, February, 2015, ISSN 978-93-84224-257, pp 52-54, organized by MMMUT, Gorakhpur.**

141. **P. K. Chaurasia, R. A Khan, "Software Models for Software Quality", National Conference on Information Security Challenges (NCISC-2016), organized by Department of Information Technology, BBAU, Lucknow 24th February 2016.**

142. **P. K. Chaurasia, "Security Breaches from Categorization of Software Errors", National Seminar on Emerging Trends and Advancements in Cyber Security, Organized by Department of Computer Application, Integral University, Lucknow on 04 April 2016.**

143. A. Yadav, R. A. Khan, “Reliability Estimation Framework-Complexity Perspective”, Computer Science and information Technology, Vol. 2 no. 5, 2012, pp. 97-104.

144. R. A. Khan, A. Agrawal, “Software Engineers: A Practitioners Approach”, Alpha Science International, May 2014.

145. M. Zhu and H. Pham, “A Software Reliability Model with time-dependent fault detection and fault removal”, Vietnam Journal of Computer Science, 2016, pp. 72-79.

146. R. Mahajan, S. K Gupta and R. K. Bedi, “Design of Software Fault Prediction Model Using BR Techniques”, Elsevier Procedia Computer Science, Vol. 46, 2015, pp. 849-858. Also Available at : http://www.sciencedirect.com/science/ article/pii/S187705091 5002185.

147. R. Rana, M. Staron, C. Berger, J Hanson, and M. Nilson, “Selecting Software Reliability Growth Models and Improving their Predictive Accuracy using Historical Projects Data”, Elsevier The Journal of Systems and Software, Vol. 98, 2014, pp. 59-78. Also Available at: http://www.sciencedirect.com/science /article/pii/S01641212 1400185X.

148. F. Febero, C. Calero, and M.A Moraga, “Software reliability modeling based on ISO/IEC SQuaRE, Information and Software Technology, Vol. 70, Feb. 2016, pp.18-29.

149. Y. Lian, Y. Tang, and Y. Wang, “Objective Bayesian analysis of JM model in software reliability”, Computational Statistics & Data Analysis, Vol. 109, May 2017, pp.199-214.

150. M. P. Cristescu, E. A. Stoica, and L. V. Ciovica, “The Comparison of Software Reliability Assessment Models”, Procedia Economics and Finance, Vol. 27, 2015, pp. 669-675.

151. D. Amara, L. B. A. Rabai, “Towards a new framework of software reliability measurement based on software metrics”, Procedia Computer Science, Vol. 109, 2017, pp. 725-730.

152. M. Kooli, F. Kaddachi, G.D Natale, A. Bosio, P. Benoit, and L Torres, “Computing Reliability: On the difference between software testing and software fault injection techniques”, Journal of Microprocessor and Microsystems, Vol. 50, May 2017, pp. 102-112.

153. R. Rana, M. Staron, C. Berger, J Hansson, M. Nilsson, F. Torner, W. Meding, and C. Hoglund, “Selecting Software Reliability growth models and improving their predictive accuracy using historical projects data, Journals of Systems and Software, Vol. 98, Dec 2014, pp. 59-78.

154. J. A. Dallal, “Predicting move method refactoring opportunities in object-oriented code”, Journal of Information and Software Technology, Vol. 92, Dec 2017, pp. 105-120.

155. T. T Pham, X. Defago, and Q. T. Huynh “Reliability prediction for component-based software systems: Dealing with concurrent and propagation errors”, Journal of Science of Computer Programming, Vol. 97, No. 4, January 2015, pp. 426-457.

156. L. Kumar, S. Misra, and S. K. Rath “An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes”, Journal of Computer Standards & Interfaces, Vol. 53, August 2017, pp. 1-32.

157. T. Kim, K. Lee, and J. Baik, “An effective approach to estimating the parameters of software reliability growth models using a real-valued genetic algorithm”, Journal of Systems and Software, Vol. 102, April 2015, pp. 134-144.

158. F. Febrero, C. Calero, and M.A. Moraga, “A Systematic Mapping Study of Software Reliability Modeling”, Journal of Information and Software Technology, Vol. 56, Issue8, August 2014, pp. 839-849.

159. R. Malohatra, “An Empirical framework for defect prediction using learning techniques with android software”, Journal of Applied Soft Computing, Vol. 49. Dec-2016, pp. 1034-1050.

160. I. Arora, V. Tetarwal, and A Saha, “Open Issues in Software Defect Prediction”, Procedia Computer Science, Vol. 46, 2015, pp.906-912.

161. S. Kanmani, V. R. Uthariaaraj, V. Sankaranaryanan, and P. Thambidurai, “Object-oriented software fault prediction using neural networks”, Vol. 49, Issue 5, May 2007, pp.483-492.

162. F. Huang, and B. Liu, “Software defect prevention based on human error theories”, Chinese Journal of Aeronautics”, Vol. 30, Issue 3, June 2017, pp.1054-1070.

163. E. Erturk, and E. A. Sezer, “Iterative software fault prediction with a hybrid approach”, Vol. 49, December 2016, pp.1020-1033.

164. D. Amara, and L.B.A. Rabai, “Towards a new framework of software reliability measurement based on software metrics”, Vol. 109, 2017, pp.725-730.

165. B. Walter, and T Alkhaeir, “The relationship between design patterns and code smells: An exploratory study”, Vol. 74, June 2016, pp. 127-142.

166. R. Morales, Z. Soh, F. Khomh, G. Antoniol, and F. Chicano, “On the use of developers’ context for automatic refactoring of software anti-patterns”, Journal of systems and software, Vol. 128, June 2017, pp.236-251.

UPTEC Computer Consultancy Limited

(An ISO 9001 : 2008 Certified Company)

Date : 30.10.2017

TO WHOMSOEVER IT MAY CONCERN

This is to certify that **Mr. Pawan Kumar Chaurasia,** a Ph.D Scholar from the Department of Information Technology, **Babasaheb Bhimrao Ambedkar University, Lucknow** has conducted some reliability database verification work with our organization. He has used our projects along with other details for research purposes.

The identification of these projects has concealed as per our desire and organization policy. The source data that is going into thesis is correct to the best of my knowledge and belief.

30.10.2017

DR. R.K. JAISWAL
Director

Regd. Office : 16, Rana Pratap Marg, UPTEC Crossing, Lucknow – 226 001. Phone : 2206794, 2209603, 2206021-22 ; Fax : 0522 – 2209493
E-mail : contact@uptecnet.com