IMPROVING SOFTWARE MAINTENANCE USING SOFTWARE METRICS

By

ANURADHA CHUG

Roll No. 2k11/Ph.D./Comp.Sc./02

Under the guidance of

Dr. Ruchika Malhotra

Associate Head and Assistant Professor,

Department of Software Engineering

Submitted in fulfillment of the requirements of the degree of
Doctor of Philosophy to the

DELHI TECHNOLOGICAL UNIVERSITY

(FORMERLY DELHI COLLEGE OF ENGINEERING)

SHAHBAD DAULATPUR, MAIN BAWANA ROAD, DELHI 110042

December, 2016

CERTIFICATE

DELHI TECHNOLOGICAL UNIVERSITY

(Govt. of National Capital Territory of Delhi)

BAWANA ROAD, DELHI - 110042

Date: ________________

This is to certify that the work embodied in the thesis titled **“Improving Software Maintenance Using Software Metrics”** has been completed by **Anuradha Chug** under the guidance of **Dr. Ruchika Malhotra** towards fulfillment of the requirements for the degree of Doctor of Philosophy of Delhi Technological University, Delhi. This work is based on original research and has not been submitted in full or in part for any other diploma or degree of any university.

Supervisor

DR. RUCHIKA MALHOTRA

Associate Head and Assistant Professor, Department of Software Engineering

Delhi Technological University, Delhi 110042

Declaration

I, **Anuradha Chug**, Ph.D. student (Roll No: 2k11/Ph.D./Comp.Sc./02) hereby declare that the thesis entitled **"Improving Software Maintenance Using Software Metrics"** which is being submitted for the award of the degree of doctor of philosophy in Computer Engineering, is a record of bonafide research work carried out by me in the Department of Computer Science & Engineering, Delhi Technological University.

I further declare that the work presented in the thesis had not been submitted to any University or Institution for any degree or diploma.

Date :

Place : New Delhi

Anuradha Chug
Roll No.: 2k11/Ph.D./Comp.Sc./02
Department of Computer Science & Engineering,
Delhi Technological University,
New Delhi -110042

Acknowledgment

On the occasion of submitting my thesis for fulfillment of the requirement of the degree 'Doctor of Philosophy', I would take this opportunity to express my immeasurable appreciation and deepest gratitude for the help and support to the following persons, who in one way or the other, have contributed in making this study possible.

It's a matter of great fortune and privilege to work under the able guidance of my supervisor **Dr. Ruchika Malhotra, Associate Head and Assistant Professor, Department of Software Engineering, Delhi Technological University**. Without her sound advice, excellent supervision, valuable suggestions, wise counsel and technical guidance, I would not have been able to complete the dissertation in this manner. I am deeply indebted to her for shaping my path of research by constantly supporting me with her extensive knowledge, insightful discussions and ever available help. She is one of the smartest people I've ever known and she would remain my best role model as a scientist, researcher, mentor, and teacher. I would remain obliged to her for being patient, precise and motivating me at all the times. I am really thankful to her for the consistent support and encouragement she provided me throughout the period of this research, which has brought me where I stand today. The positive vibes she transferred to me during this research work deserve admiration. I feed proud to be associated with her and look forward to carrying this relationship to greater heights.

With a profound sense of gratitude, I want to give special thanks to **Prof. Yogesh Singh, Vice Chancellor, Delhi Technological University and Director, Netaji Subhas Institute of Technology.** I cherish my old memories, when I used to attend the software engineering classes delivered by him during my M.Tech. course. It was during these classes that my love for this subject actually developed. Earlier I used to consider software engineering as a theoretical subject, however, it was his excellent delivery of the concept using numericals that changed my opinion and I felt inspired for the current study.

I would like to convey my sincere thanks to **Prof. O.P. Verma, Head, Department of Computer Science and Engineering, Delhi Technological University** for his encouragement, help and cooperation in accomplishing the current research.

I would like to thank all the **faculty members** of Department of Computer Science & Engineering, Delhi Technological University for their valuable discussion and guidance during the course of this study. I would like to thank my friends, fellow research scholars and office staff for all their direct and indirect assistance in carrying out this task.

I would like to thank my husband **Lt Col P K Chug** who has always inspired me for hard work and motivated me to focus on my research work. I am grateful to him for his much needed emotional support and unconditional love during the course of research journey apart from willing support always available at my disposal.

I would also like to thank my **parents** and **parents-in-law** for their blessings and sharing my responsibilities as a home-maker during this research work. Lastly and by no means least, I thank my children **Ms. Sezal** and **Master Naman** whose precious time I have invested in carrying this research work because often, they had to endure my absence, but they seldom complained.

Finally, I bow to the **Almighty Waheguru** who gave me the strength to carry out this work with sincerity, honesty and dedication.

Anuradha Chug

Abstract

Changes in the newly developed software are inevitable due to multiple reasons such as change in user requirements, advancement in technology and business competition pressure. One of the biggest challenges faced by the software engineers today is to develop large and complex software in stipulated time frame, under budgetary constraints, meeting the customer's demands and needs. Maintenance phase starts once the software product is delivered to the customer and during this period software have to be constantly improved/modified on the basis of the change in customer's requirements or software environment. Software maintainability means the ease with which a software can be modified to correct faults, improve the performance or adapt to a changed environment. Since the maintenance phase consumes one third of the total cost of the software development life cycle, producing a software that is easy to maintain may potentially save large costs and efforts.

Any endeavor towards increasing the maintainability of the system would eventually be helpful in reducing the overall project cost. Many studies have previously established the strong relationship between software design metrics and maintainability of the system. This means that by constantly measuring the design metrics and applying regularly certain methods, the overall maintainability of the system can be monitored and improved. It would certainly have a positive impact on the development of software and implementing the best practices for the software development community. In this study, solutions are proposed for monitoring and analyzing large software system using a set of software design metrics during the product development stage. Further, with the help of certain validated prediction models, these metrics can be gainfully applied in the software engineering processes in general and for optimizing software maintainability in particular.

It is very important to predict the efforts required to accommodate these changes during maintenance phase at an early stage of software development. It will help the managers to allocate resources more judiciously, thereby leading to the reduction of costs overruns. One of the main approach in controlling maintenance cost is to monitor software design metrics during the development phase itself. There are several object oriented metrics proposed in the literature to capture the design properties such as coupling, cohesion, inheritance, and polymorphism. It is a matter of interest for researchers to measure the software using design metrics and predict its maintenance behavior on the basis of their values. The problem of predicting the maintainability of software is widely acknowledged in the industry and much has been written on how maintainability can be predicted by using various tools and processes at the time of development with the help of software design metrics.

Several statistical and machine learning techniques have been proposed in the literature for prediction problems across a range domains such as finance, medicine, engineering, geology and physics. Most of the prediction models in the literature are built using statistical and machine learning techniques. There are very few studies which are using evolutionary techniques for predicting software maintainability using object oriented metrics. Since evolutionary techniques have different results, there is strong need to conduct more and more data based empirical studies that are capable of being verified by observation or experiments. Conducting such large empirical studies and comparing the performance of evolutionary techniques with statistical and machine learning techniques is helpful for the creation of well established theories.

Many metric suites are proposed in literature to measure the object oriented software. The current research work was undertaken to examine these various object oriented metric suites defined in literature and in this regard empirical investigations were conducted using machine learning techniques and evolutionary techniques with an aim of constructing a generalized, reliable and repeatable model to predict precise software maintainability during an early phase of software product development.

Further, large-scale bench-marking framework for the maintainability predictions of open source software system was also developed during the course of the current study. Metric based maintainability prediction model developed in this study would be helpful in software development process without escalating budgetary considerations taking least possible time frame. The framework and reference architecture in which the software systems are being currently developed world over are fast changing dramatically due to the emergence of data warehouse and data mining tools. To appropriately address this challenge, a new metric suite to redefine the relationship between design metrics and maintainability for data intensive applications is also proposed.

In this research study, we are principally concerned with various methods and measures used to improve the software maintainability by predicting maintenance effort with the help of internal quality attributes. Certain methods are also available in literature which improve the design of the code and in turn enhances maintainability such as Clean Code, Pair Programming, Lean, Crystal, Kanban, scrum etc. Refactoring is one of the important activity carried out in maintenance phase, in which the design of software is improved and complexity is reduced without affecting its external behaviour. Many refactoring methods have been suggested in the literature and each has a specific purpose and corresponding effect. However, it is so far unclear how a particular refactoring method affects the software maintainability. One of the objectives of this study was also to observe the quantifiable effects of few commonly applied refactoring methods on software maintainability. The design metrics of the software were calculated and analyzed, both before as well as after the application of refactoring method and comprehensive reports were prepared to observe the effects of selected refactoring methods on design metrics which were subsequently mapped to the maintainability of the software system. It helps in identifying the opportunities of refactoring in software comprising of a large number of lines of source code with an aim to ascertain its potential to optimize software maintainability.

Agile methodology is comparatively new method of software development to address the problem of unpredictability in business. It provides alternatives to the traditional project management techniques and nowadays more and more products are developing in software industry based on agile methodology to ensure quality, reliability and scalability of the delivered software products. Scrum is one of the most vital agile method whose impact on software maintainability has also been investigated in this study. We developed the same product using scrum method as well as Iterative Enhancement Model and compared both of them using carefully selected metrics from the maintenance point of view. It helps the project managers to create a flexible product in which defects are identified during early stages of software development life cycle thereby avoiding any cost overrun.

Contents

List of Publications

Papers Accepted/Published in International/National Journals

1. Ruchika Malhotra and Anuradha Chug, Software Maintainability: Systematic Literature Review and Current Trends, *International Journal of Software Engineering and Knowledge Engineering*, World Scientific Publishing, **Index in Science Citation Index (SCI) and Thomson Reuters, Impact Factor:0.368.** (In Printing)

2. Anuradha Chug and Ruchika Malhotra, Bench-Marking Framework for Maintainability Prediction of Open Source Software using Object-Oriented Metrics, *International Journal of Innovative Computing, Information and Control*, 1(2): pp. 615-634, 2016. **Index in SCOPUS, Compendex (Elsevier) and INSPEC (IET)** Impact Factor:0.295.

3. Ruchika Malhotra and Anuradha Chug, Software Maintainability Prediction using Machine Learning Algorithms, *Software Engineering: An International Journal*, 2(2):pp. 19-36, 2012.

4. Ruchika Malhotra and Anuradha Chug, Application of Group Method of Data Handling model for software maintainability prediction using object-oriented systems, *International Journal of System Assurance Engineering and Management, Springer*, 5(2): pp. 165-173, 2014. **Index in SCOPUS, Compendex (Elsevier) and DBLP, OCLC, SCImage**

Papers Accepted/Published in International Conferences

5. Ruchika Malhotra and Anuradha Chug, An Empirical Validation of Group Method of Data Handling (GMDH) on Software Maintainability Prediction using Object-Oriented Systems, *International Conference on Quality, Reliability and Information Technology,* 27-29 Nov 2012, New Delhi, India.

6. Ruchika Malhotra and Anuradha Chug, Software Maintenance Prediction with the help of Machine Learning Algorithms, *National Conferences on Recent advances in Software Engineering*, 20-21 April 2012, New Delhi, India.

7. Ruchika Malhotra and Anuradha Chug, Application of Evolutionary Algorithms for Software Maintainability Prediction using Object-Oriented Metrics, *8th International Conference on Bio-Inspired Information and Communication Technologies*, BIONET-ICS, 1-3 December 2014, Boston, MA, USA.

8. Ruchika Malhotra and Anuradha Chug, An Empirical Study to Redefine the Relationship between Software Design Metrics and Maintainability in High Data Intensive Applications, *Proceedings of the World Congress on Engineering and Computer Science* 23-25 October, 2013, San Francisco, USA.

9. Ruchika Malhotra and Anuradha Chug, Comparative Analysis of Agile Methodology and Iterative Enhancement Model in Assessment of Software Maintenance, *Computing for sustainable Global Development,IndiaCom, Conference ID 37465*, 15-16 March 2016, New Delhi, India.

10. Ruchika Malhotra and Anuradha Chug, An Empirical Study to Assess the Effects of Refactoring on Software Maintainability, *International Conference on Advances in Computing, Communications and Informatics, ICACCI-2016, IEEE Conference ID 38419*, 20-22 September 2016, Jaipur, India.(Accepted for publication)

Papers Published as Book Chapter

11. Ruchika Malhotra and Anuradha Chug, Metric Suite for Predicting Software Maintainability in Data Intensive Applications, Book Chapter, pp. 161-175, *Transactions on Engineering Technologies, Springer*, February, 2014.

Papers Communicated in International Conferences/ Journals

12. Ruchika Malhotra and Anuradha Chug, Estimating Software Maintenance Efforts using Evolutionary Approach, Journal, Informatica: Journal of Computing and Informatics.

List of Tables

List of Figures

Abbreviations

ABPS	**Angel Bill Printing Software**
ACC	**Average Cyclomatic Complexity**
ACO	**Average Cyclomatic Complexity**
AHF	**Attribute Hiding Factor**
AID	**Access of Imported Data**
AIF	**Attribute Inheritance Factor**
AMLOC	**Average Method size in Lines Of Code**
AMC	**Average Method Complexity**
ANA	**Average Number of Ancestors**
ANFIS	**Adaptive Neuro Fuzzy Inference System**
ANN	**Artificial Neural Network**
ANOVA	**Analysis Of Variance**
API	**Application Programming Interface**
APP	**Atom Publishing Protocol**
ARE	**Absolute Relative Error**
BPN	**Back Propagation Network**
BBN	**Bayesian Belief Network**
CA	**Cuckoo Algorithms**
CAM	**Cohesion Among the Methods of a Class**
CBO	**Coupling Between Object classes**
CBM	**Coupling Between Methods**
CC	**Cyclomatic Complexity**
CCE	**Consolidate Conditional Expression**
CD	**Critical Distance**
CF	**Coupling Factor**

CFS	**Correlation based Feature Subset-Selection**
CFT	**Cross Functional Teams**
CMC	**Class Method Complexity**
CR	**Change Request**
CTM	**Coupling Through Message passing**
CIS	**Class Interface Size**
CTA	**Coupling Through Abstract data type**
DPD	**Dynamic Polymorphism in Descendants**
DAC	**Data Abstraction Coupling**
DAM	**Data Access Metric**
DCC	**Direct Class Coupling**
DCRS	**Defect Collection and Reporting System**
DIT	**Depth of Inheritance Tree**
DOQ	**Documentation Quality**
DOD	**Definition of Done**
DPA	**Dynamic Polymorphism in Ancestors**
DT	**Decision Tree**
DSC	**Design Size in Classes**
EC	**Extract Class**
EF	**Encapsulating Field**
EM	**Extract Method**
FFNN	**Feed Forward Neural Network**
FIR	**Friedman’s Individual Rank**
FLMS	**File Letter Monitoring Software**
FP	**Function Point**
FSS	**Feature Subset Selection**
GA	**Genetic Algorithms**

GMDH	**Group Method of Data Handling**
GRNN	**General Regression Neural Networks**
GGAL	**GRNN with Genetic Adaptive Learning**
HM	**Hide Method**
HMM	**Hidden Markov Model**
IC	**Inheritance Coupling**
IDE	**Integrated Development Environment**
IEM	**Iterative Enhancement Model**
IMS	**Inventory Management Software**
ISO	**International Organization for Standardization**
IL	**Interaction Level**
IS	**Interface Size**
JERN	**Jordan Elman Recurrent Network**
LCC	**Loose Class Cohesion**
LCOM	**Lack of Cohesion in Methods**
LCOM3	**Lack of Cohesion Among Methods of a Class**
LOC	**Lines of Code**
LR	**Linear Regression**
MARE	**Mean Absolute Relative Error**
MARS	**Multiple Adaptive Regression Splines**
MBLR	**Multivariate Binary Logistic Regression**
MFA	**Method of Functional Abstraction**
MHF	**Method Hiding Factor**
MI	**Maintainability Index**
MIF	**Method Inheritance Factor**
MLR	**Multivariate Linear Regression**
MMRE	**Mean Magnitude of Relative Error**

MOA	**Measure Of Aggression**
MPC	**Message Passing Coupling**
MPMT	**Mean Preventive Maintenance Time**
MTTR	**Mean Time to Repair**
MCMT	**Mean Corrective Maintenance Time**
MaCMT	**Maximum Corrective Maintenance Time**
MRE	**Magnitude of Relative Error**
NAC	**Number of Ancestor Classes**
NDC	**Number of Descendant Classes**
NIHICP	**Non-Inheritance Information Flow-Based Coupling**
IHICP	**Information Flow-Based Inheritance Coupling**
NLM	**Number of Local Methods**
NOC	**Number Of Children**
NODBC	**Number of Data Base Connections**
NOH	**Number of Hierarchies**
NM	**Number of Methods**
NOMA	**Number of Object/Memory Allocation**
NOP	**Number of Polymorphic methods**
NPM	**Number of Public Methods**
NPV	**Number of Public Variables per class**
NV	**Number of Variables per class**
NMI	**Number of Methods Inherited by a subclass**
NMO	**Number of Methods Overridden by a subclass**
NMA	**Number of Methods Added by a subclass**
NCR	**Number of times a class is Reused**
NF	**Number of Friends of a class**
OAC	**Operation Argument Complexity**

OO	**Object Oriented**
OVO	**Overloading in Stand-alone Classes**
PCA	**Principal Component Analysis**
PF	**Polymorphism Factor**
PO	**Product Owner**
PSO	**Particle Swarm Optimization**
PM	**Number of Public Methods**
PNN	**Probabilistic Neural Networks**
QA	**Quality Assurance**
QUES	**Quality Evaluation System**
RFC	**Response For a Class**
RMSE	**Root Mean Square Error**
RQ	**Research Questions**
RSC	**Readability of Source Code**
SPA	**Static Polymorphism in Ancestors**
SPD	**Static Polymorphism in Descendants**
SCC	**Scrum Code Camp**
SCCR	**Schema Complexity to Comment Ratio**
SDLC	**Software Development Life Cycle**
SIX	**Specialization IndeX**
SLR	**Stepwise Logistic Regression**
SMS	**Student Management Software**
SOUL	**Smalltalk Open Unification Language**
SRS	**Software Requirement Specification**
SVM	**Support Vector Machine**
TCC	**Tight Class Cohesion**
UIMS	**User Interface Management System**

UOS	**Understandability Of Software**
WMC	**Weighted Methods per Class**
XP	**eXtreme Programming**

Chapter 1

Introduction and Literature Survey

1.1 Introduction

Software quality is defined as the degree of excellence in terms of the purpose for which it has been made. The major reason for developing the quality of the software upto a certain level is to make sure that the final delivered software is as per the requirements of the customer and compliance of the well established standards given by few professional organizations such as ISO, IEEE etc. Maintaining high quality of the software saves a large amount of time and money because it will have fewer defects as well as fewer requests for any change in future.

Maintainability is one of the important attribute of software quality which determine how easy it is to modify the software once it is delivered to the customer [63, 202]. Reducing the cost and efforts required to maintain the software will lead to reduction in the overall cost of the project because it consumes almost one third of the total cost of the software development life cycle (SDLC). It is proved in many research studies that many quality attributes such as maintainability, reliability, defects and fault proneness can be predicted during the early stages of software development using source code metrics and past defect detection data [34].

In this chapter, we present an overview of software quality with a special focus on software maintenance which is one of the most important software quality attribute. We present the formal definition of software maintainability and discuss the problems associated with software maintenance and other related core concepts. Various types of maintenance activities are also presented and in this regard, importance of software maintainability prediction is being discussed. In the later part of the chapter, we have presented the literature review of various software design metrics specially the Object Oriented (OO) metrics and prediction models related to software maintainability since the main focus of our research is “software maintainability prediction using OO metrics”.

In addition to the early estimation of maintenance efforts using software maintainability prediction models as mentioned above, many other ways are also suggested in literature to improve the design of the code and reduce the efforts required to carryout the maintenance work. These techniques include Clean Code, Pair Programming, Consistent Design, Lean, Crystal, Kanban, scrum, Refactoring, Agile Methodology, Exhaustive Testing, eXtreme Programming (XP) etc. Out of these, two vital techniques refactoring and agile methodology are being studied in greater details to understand their impact on software maintainability. An overview of these techniques is also presented in this chapter. In the last section of this chapter, we have presented the goals and organization of the thesis.

1.2 Software Quality

Modern civilization is heavily driven by software in all walks of life, be it an automobile, communication device or household appliance. As the software is so widely used at each and every place, maintaining the quality of the software is very important. A variety of quality models have been proposed in the literature which describes software quality and also proposed the procedures to ascertain the level of software quality by its measurement [134]. An essential part of the software development process is to measure all aspects of software quality throughout the SDLC phases and ensure its quality at desired level at all the time.

According to ANSI Standard (ANSI/ASQC A3/1978)[95], the definition of the term ‘Quality’ is given as *“Quality is the totality of features and characteristics of a product or a service that bears on its ability to satisfy the given needs”.*

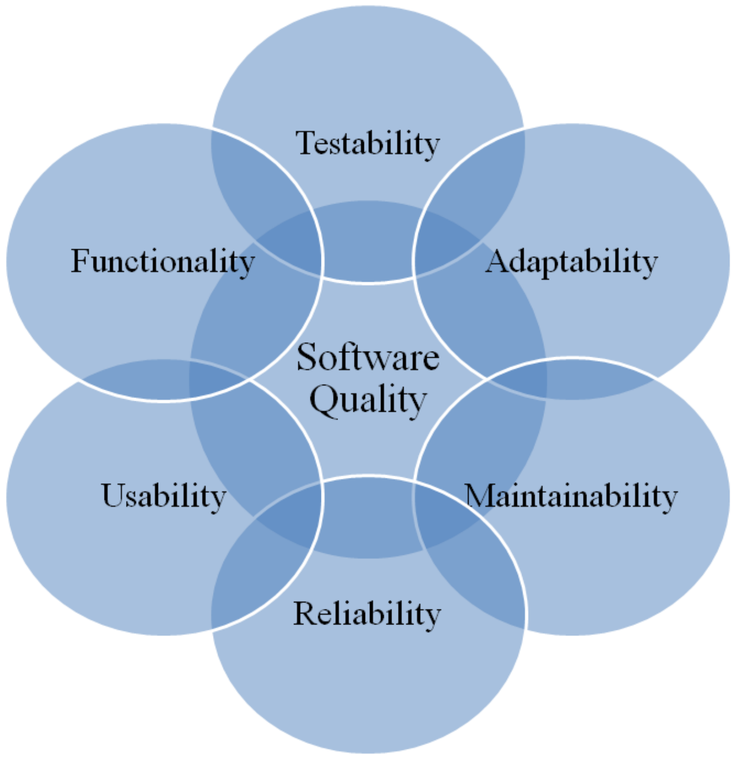

Figure 1.1: Software Quality Attribute (Source: Book, Object Oriented Software Engineering by Yogesh Singh and Ruchika Malhotra)

As per the definition, software quality is defined as the degree to which software posses a desired combination of various attributes such as Functionality, Testability, Adaptability, Maintainability, Reliability, and Usability [202]. As shown in figure 1.1, first and the main attribute is ‘functionality’ which focuses on the functions that are provided by the software product. Second,‘testability’ which measures how easy is the process of creating the test criterion for the given software. It also ensure that tester should be able to execute these tests in order to determine if the criteria are met or not. Third, ‘adaptability’ which refers to how well the software can adapt to changes in its environment or with its requirements.

Fourth, ‘maintainability’ which defines the ability to identify and fix a fault within a software component. Fifth, ‘reliability’ which means the capability of the software to maintain its service provision under defined conditions for defined periods of time. Lastly, ‘usability’ which refers to the ease of use for a given function of the specific software. These main characteristics are further broken down into sub-characteristics for their correct measurement and management purpose as compiled in table 1.1.

Table 1.1: Software Quality Attributes (Source: Book, Object Oriented Software Engineering by Yogesh Singh and Ruchika Malhotra)

Functionality	Completeness	The extent to which the software is complete.
	Correctness	The extent to which the software is correct.
	Efficiency	The extent to which the resources are required.
	Traceability	The extent to which the requirement are traceable.
	Security	Measure whether software is able to prevent unauthorized access.
Usability	Learnability	The extent to which the software is easy to learn.
	Operability	Measure how software is easy to operate.
	User-Friendliness	Interface between the software and user should be easy to understand.
	Installability	Software should be easy to install.
	Satisfaction	The extent to which the software is able to prevent unauthorized access.
Testability	Verifiability	The extent to which the software meets standards, procedures, and processes
	Validable	The extent to which the software is able to meet already set criterion.
Reliability	Robustness	The extent to which the software can perform under adverse circumstances.
	Recoverability	The ability and speed of the software to recover from crash down situation.
Maintainability	Agility	The extent to which the software can accommodate change.
	Modifiable	The extent to which the software is easy to implement and modify.
	Readability	Measure the understandability of the source code and associated documents.
	Flexibility	The extent with which the software can be easily modified.
Adaptability	Portability	The extent to which the software can be transferable.
	Interoperability	The extent to which the software is compatible.

The conceptual framework of the quality attributes was provided by McCall [157], Bohem [28] and International Organization for Standardization(ISO) [95]. First time it was introduced by McCall [157], to bridge the gap between users and developers. In this model, the software quality was divided into two levels i.e. quality factors and quality criteria [157]. There were three main representations for identifying the software quality. First is the 'product revision' that influence the ability to change the software product. Second is 'product transitions' that influence the ability to adapt the software to new environments. Third is 'product operations' that influence the extent to which the software fulfills its specification [157].

One of the main constraints of McCall quality model [157] was that it does not take into account for the performance characteristics of computer hardware. Hence, Boehm [28] introduced another quality model consisting of three levels of quality attributes i.e. primary uses, intermediate constructs and primitive constructs. Boehm Model [28] qualitatively defined software quality on the basis of a given set of attributes and metrics. The quality model proposed by Boehm [28] loosely retained the factor-measurable property arrangement and the prime characteristic of quality is what they define as "general utility". Hence, the first and foremost assertion is that the software system must be useful in order to be considered as a quality system. Mainly it concentrate on three issues: how well can I use it, how easy is it to maintain it and can I still use it if I change my environment [28]. Although both the models appear very similar due to the hierarchical structure maintained in both of them, however the McCall's model [157] mainly focuses on the precise measurement of the high-level characteristics "As-is utility", whereas wider range of characteristics are measured with special focus on maintainability in the case of Boehm's quality model [28].

1.3 Software Maintenance

Change is the inevitable characteristic of software systems which occur to eradicate deficiencies if any, to correct existing faults, to make the software compatible, to increase user satisfaction, to simplify the long code complexity, or to make the software adaptable with new hardware. As shown in figure 1.2, software needs to change after its release due to various reasons.

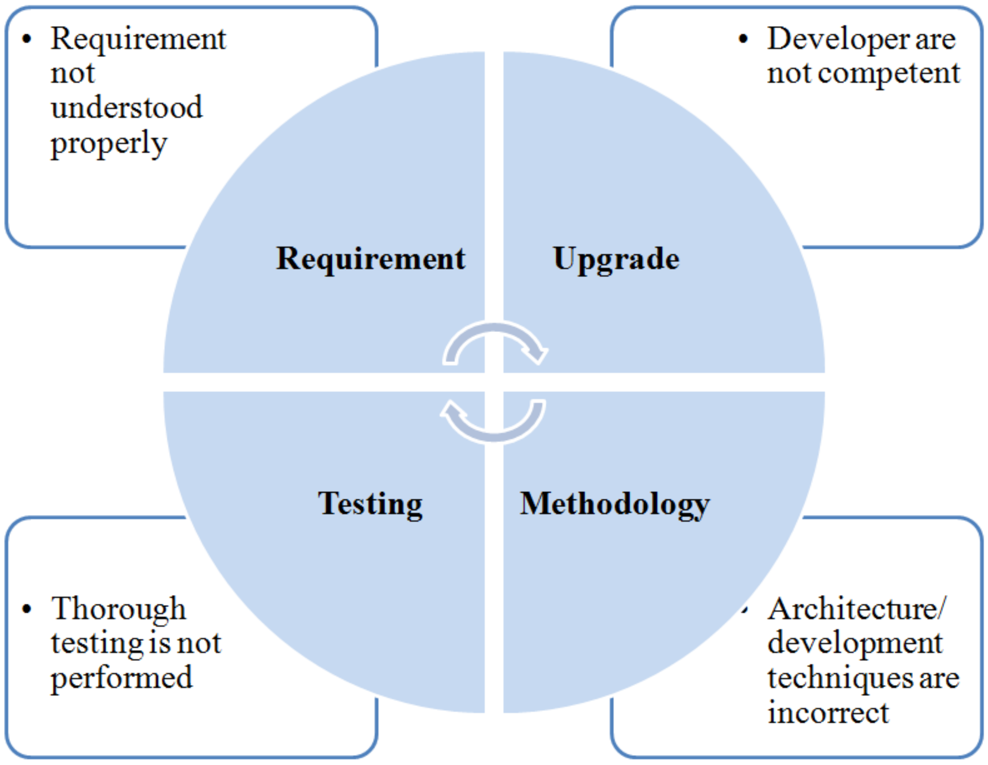
• Requirement not understood properly
• Developer are not competent
Requirement
Upgrade
Testing
Methodology
• Thorough testing is not performed
• Architecture/ development techniques are incorrect

Figure 1.2: Reasons for the Changes in Software

Among the many essential characteristics of software quality as discussed in section 1.1 such as understandability, analyzability, readability, portability etc. changeability is one of the most important characteristics. According to Belady & Lehman [19], any developed software should be able to accommodate the changes which may occur due to evolving requirements, changing platforms or any other environmental pressure. According to IEEE glossary [63], the definitions of software maintenance is given as *“Software maintenance is the process of modifying a software system or component after delivery to correct faults, improve performances or other attributes, or adapt to a changed environment*”.

The life of the software cycle consists of various activities which are broadly classified as requirement, design, implementation, testing and maintenance. Maintenance activity starts once the final product is delivered to the customer and it is considered as the most expensive part of the software development process [39]. The cost of software maintenance is rising dramatically as presented in figure 1.3.

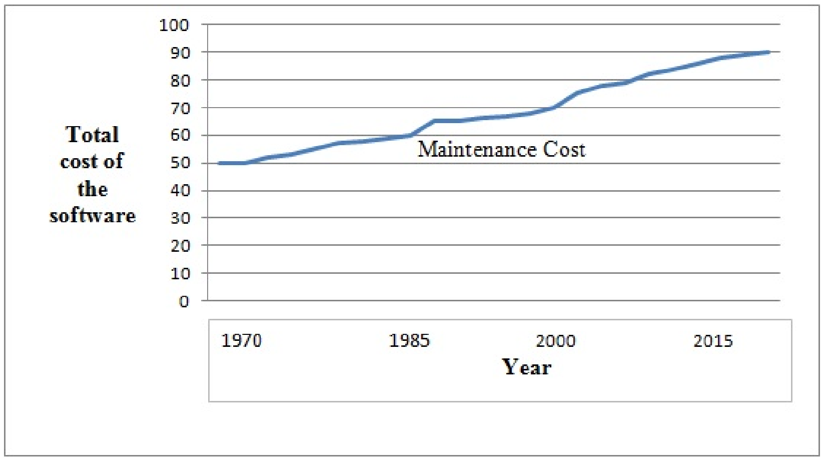

Figure 1.3: Approximate Cost Distribution of Entire Software Process Cycle Among its Various Activities (Source: Omnext White Paper, 2014)

It has been estimated in the white paper presented by Omnext in the year 2014 by Burki & Harald [39] that nowadays software maintenance accounts for more than 90% of the total cost of software, whereas it was around 50% a couple of decades ago. Systems tend to become increasingly complex and it has become extremely hard to maintain over time. Rebuilding a system is usually not an option because the system that needs to be replaced is large, the test coverage unknown and the original and modified requirements are not well documented.

Since the final cost of the product is decided based on the investments made during the maintenance phase, therefore any efforts to reduce time and cost during this phase would certainly reduce the overall cost of the product. In this backdrop, researchers are actively engaged to increase the ease of carrying out the maintenance work so that the total cost of the software product can be drastically reduced.

1.3.1 Types of Software Maintenance

Throughout the life of the software, type of maintenance activity may vary depending upon its nature. It may be just a routine maintenance tasks because some bug discovered by some user or it may be a large event in itself based on maintenance size or nature. Four types of maintenance activities based on their characteristics are shown in figure 1.4 and explained as under:

1.3.1.1 Corrective Maintenance:

Maintenance activities performed to correct faults in hardware or software. This type of maintenance tasks includes modifications and updations done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.

1.3.1.2 Adaptive Maintenance:

Maintenance activities performed to make a computer program usable in a changed environment. This maintenance task includes modifications and updations which are applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.

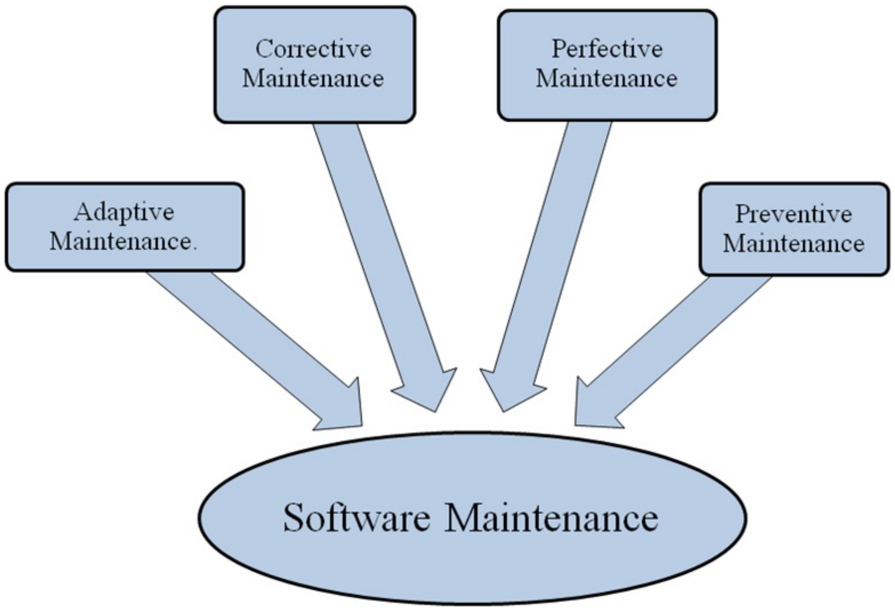

Figure 1.4: Types of Software Maintenance

1.3.1.3 Perfective Maintenance:

Maintenance activities performed to improve the quality of the software and make it perfect. This type of maintenance task includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, refactoring, new user requirements for refining the software and improve its reliability and performance.

1.3.1.4 Preventive Maintenance:

Maintenance activities that are concern with the prevention of defined issues in the software maintenance phase of a given software. This maintenance task includes modifications and updations to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

Further, it is important that while measuring software maintenance, utmost care should be taken to ensure that measurement processes are independent of the language and technology. These methods should be straight forward to avoid any chance of ambiguity. Calculation of these methods should be easy, repeatable and also understandable.

1.3.2 Software Maintainability

Software maintainability is defined as the degree to which the software can be changed in terms of Lines Of Code (LOC). For the software engineers, ability to cope up with these changes in the environment has been a challenging task and any software will undergo early demise if it does not have the capability to change. Software maintenance and evolution is a continuous process in the development of the software. If we can predict the changeability from the very beginning of the product life cycle with the help of some prediction model, it will reduce the maintenance cost due to better planning and hence improve the overall software quality. The key software maintenance issues are both managerial as well as technical. From a management view, it involves alignment with customer priorities, staffing and cost estimation while from a technical point of view, it involves an understanding of the code, impact analysis and testing.

Software maintainability is the collection of partly uncorrelated factors which should be monitored through its components rather than one single measurement [3]. Software Maintainability is a very important constituent of the Software Quality. As per the standards defined in ISO 9126 quality model [95], maintainability consists of four attributes viz Analyzability, Changeability, Stability and Testability. Software maintainability prediction is a process in which the maintenance behaviors of the software is predicted in the early phases of SDLC so that timely measures can be taken to improve the overall maintainability of the software product.

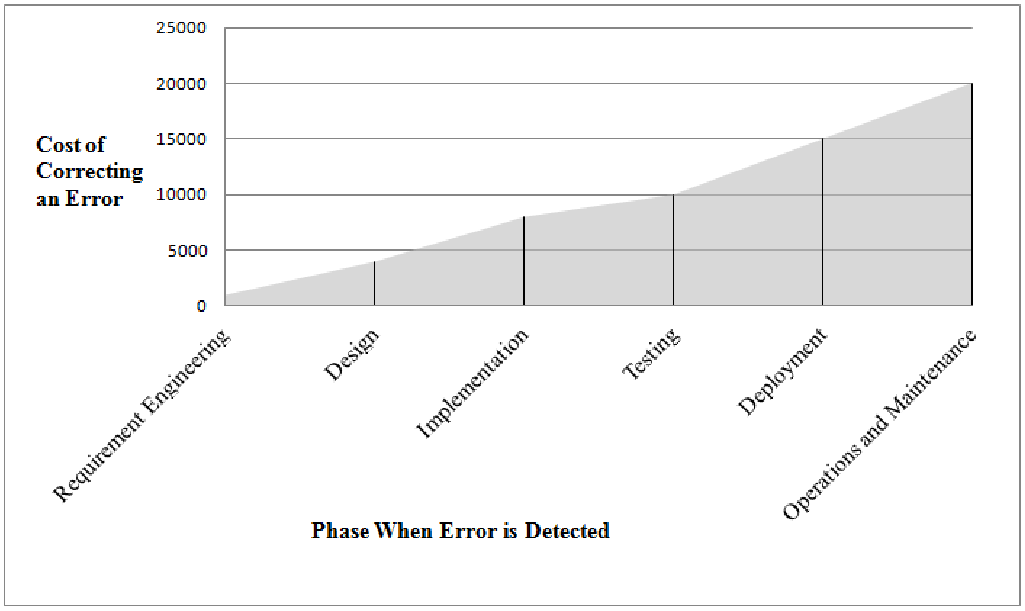

Figure 1.5: Relationship between the Cost and Time of Maintenance

As shown in figure 1.5, the amount of resource, efforts and time spent for correcting an error during maintenance is much more than doing it during the early stages of product development life cycle [171]. Hence, it is always a matter of interest for the researchers to construct prediction model for software maintainability.

1.3.3 Prediction Model for Software Maintainability

Software maintainability prediction involves the prediction of software maintainability in the early phases of SDLC. It helps in identifying those week areas of the software which can be improved to ensure overall quality of the software product. This can be carried out by constructing empirical models which can predict the external quality attributes as a function of various measurable internal quality attributes. External quality attributes are referred as target attributes (dependent variable) and the internal attributes are known as predictor attributes (independent variable).

Once the prediction models are constructed, it allows the user to measure the internal quality attributes using design metrics (discussed in next section 1.4) and estimate the required maintenance efforts in the early phase of the software development. As shown in figure 1.6, construction of prediction model consists of three phases i.e. training, testing and predicting the dependent variable.

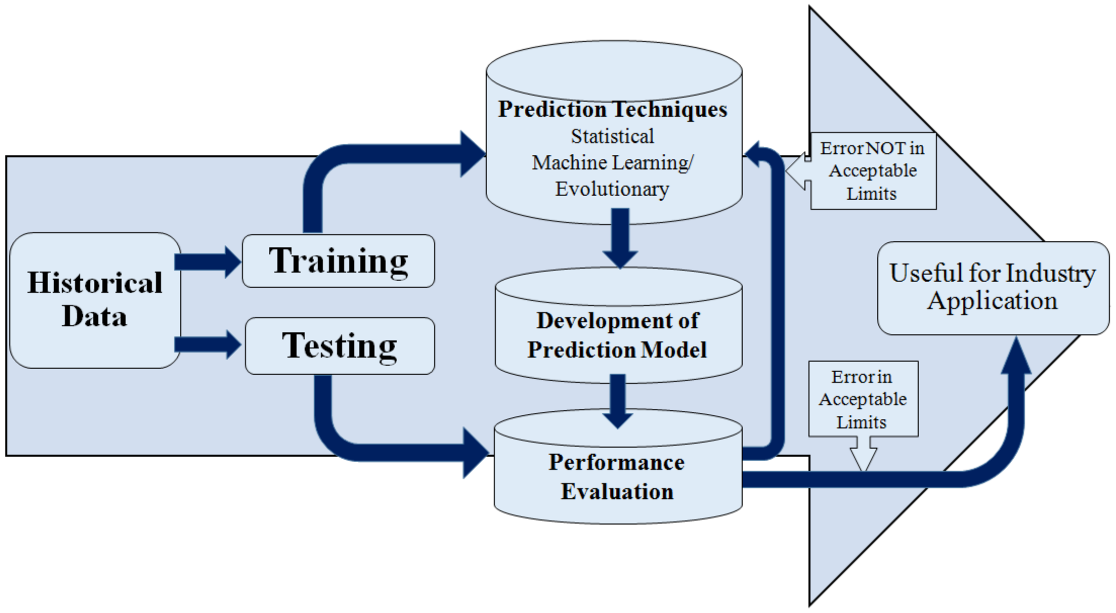
Historical Data
Training
Testing
Prediction Techniques
Statistical
Machine Learning/
Evolutionary
Development of Prediction Model
Performance Evaluation
Error NOT in Acceptable Limits
Error in Acceptable Limits
Useful for Industry Application

Figure 1.6: Creation of Software Maintainability Prediction Model

The first phase, i.e. the training or the learning phase uses the training data (for which the values of the target attribute are known) for building a model. Different classification techniques are applied on a part of the training data to build the prediction model [135]. The technique for making prediction model construction starts by dividing the data into two parts training and testing usually in 70:30 ratios. The training data is used during model development in which the model is trained by identifying the relationship between independent variables and dependent variable. These relationships are further captured in the model in the form of various classification rules which can actually be used during prediction process. In the next phase which is called as testing phase, these learned classification rules are applied on remaining 30% of the data to obtain the predicted value of the dependent variable [135].

This predicted value is further compared with the actual value of the dependent variable to determine the predictive capability of the model. If the error of the model is in the acceptable range then this model enters into the third phase i.e. prediction phase where it can be used on new data in which independent variables are known and dependent variable is unknown. Once the generalized model is ready after testing and validations, it can further be used for industry applications. On the basis of the classification rules learned in the first phase, the value of the dependent variable can be predicted using independent variables. The prediction model development process consists of the following steps in brief:

1. History dataset is divided into two parts i.e. training and testing
2. From the available independent attributes, only relevant independent attributes are selected using correlation analysis discussed in chapter 2.
3. Model is developed using statistical and machine learning techniques
4. Model is validated using various techniques such as holdout cross-validation method, n-fold cross-validation method and Leave one out cross-validation method (Discussed in section 2.12).
5. With the help of various statistical tests, hypothesis testing is performed.
6. Results are interpreted and accuracy of the model is determined.
7. If the errors is in acceptable range, model is used for the industry applications.

Various empirical studies have been conducted in past on this aspect and there has emerged a strong link between software design metrics and its maintainability. It has also been found that these software metrics can be used as predictors of maintenance effort. Figure 1.7 display benefits of predicting software maintainability to various stakeholders.

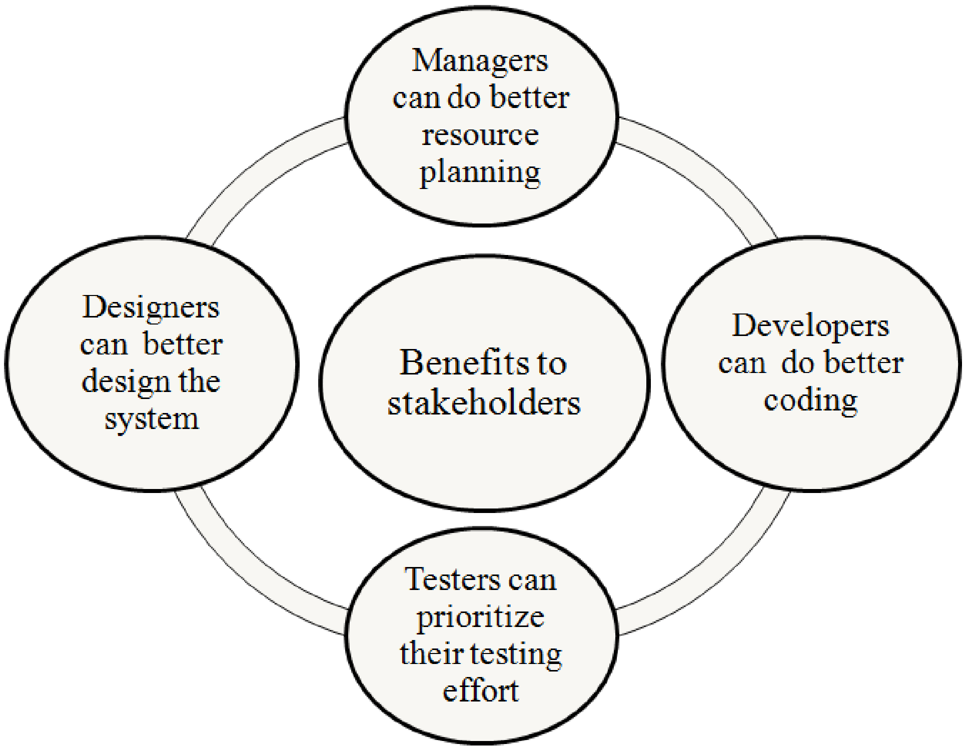
Managers can do better resource planning
Designers can better design the system
Benefits to stakeholders
Developers can do better coding
Testers can prioritize their testing effort

Figure 1.7: Advantages of Predicting Software Maintenance

Further, the advantages of maintainability prediction are summarized as follows:

- It helps project managers in comparing the productivity and costs among different projects.
- It provides managers with information so that the available resources can be planned more effectively.
- It helps managers in taking a vital decision regarding staff allocation.
- It guides about the efficiency of the maintenance process.
- It helps in keeping future maintenance efforts under control.

- It enables the developers to identify the determinants of software quality so that they can improve design and coding.
- It helps practitioners to improve the quality of software systems and thus optimize maintenance costs.

Software maintainability prediction helps to analyze and thus take corrective actions to improve the life of the software. By better understanding of software maintainability prediction, we can work on to reduce system repair times, thereby reducing downtime and increasing our system availability. Efficient software maintainability prediction models allow us to define our repair tasks and freely reuse this information throughout in an effort to improve our design.

1.4 Software Metrics

Software metrics are used for monitoring and improving various processes and products in software engineering. The rationale arises from the notion given by DeMarco [148] that "you cannot control what you cannot measure". While developing any software project, it is very important that we measure the quality, cost and effectiveness of the project and the processes, otherwise we would not be able to successfully complete the product within allocated budget in the specified time limits. Controlling the software projects and improving the quality of the software can be done by means of software metrics.

The formal definition of the software metric given by Goodman [79] as *"The continuous application of measurement based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products."*

The above definition explains the importance of software metrics and insists that they should be collected right from the initial phases of software development to measure the cost, size, and effort of the project and they can be used to ascertain and monitor the progress of the software throughout the SDLC. Achieving the software quality is a key task for any software industry which is hard to achieve because the complexity of the software tends to be high. The main aim of a software metric is to understand and analyze the quality of a given software product at each and every phase by its measurement. The main advantages of software metrics are summarized as under:

- Project status can be easily tracked.
- Helps in the detection of the defect in early phases of the software development.
- Overall cost reduces due to early removal of defects.
- Effective and informed decision can be taken by project managers.
- Project management as well as process management can be improved.
- Statistics can be presented in an organized manner to the management.
- Testers can compare the values of software metrics with respective thresh hold values established by researchers.
- Reduces the risk for managers.
- Back-tracking of the changes in order to remove bugs becomes very easy.
- Historical data can be collected which further helps the testing process to be more effective
- Change control can be easily accomplished without confusion.

1.4.1 Characteristics of Software Metrics

Measuring the structural design properties of a software system such as coupling, cohesion or inheritance and deploying them for early quality assessment for a given software is suggested by many researchers. There are several metrics proposed in the literature to capture the quality of design and code and choosing the right metrics is equally important. Seven criteria were suggested by Abreu and Carapuca [2] while selecting the best design metrics such as size-independence, down-salable, easily computatable, language independent, early obtainable and dimensionless. A metric is only relevant if it is easily understood, calculated, valid, and economical. Following are the important characteristics of software metrics [135]:

- Quantitative: The metrics should be expressible in values.
- Understandable: The way of computing the metric must be easy to understand.
- Validatable: The metric should capture the same attribute that it is designed for.
- Economical: It should be economical to measure a metric.
- Repeatable: The values should be same if measured repeatedly, that is, can be consistently repeated.
- Language independent: The metrics should not depend on any language.
- Applicability: The metric should be applicable in the early phases of software development.
- Comparable: The metric should correlate with another metric capturing the same feature or concept.

1.5 Literature Survey

This section is divided into five parts wherein the first section provides the summary of traditional software metrics. Next section provides the details about a current state where OO metrics are used to predict the respective maintainability of the software. In the next section, we describe the literature work regarding software maintainability prediction for OO software. In the next section, refactoring techniques and their effects on software maintainability are presented. In the last section, review of the use of agile methodology for improving software maintenance is presented.

1.5.1 Traditional Metrics

Research in the field of software metrics was carried out on the basis that we can improve quality and productivity of the software only by measuring its characteristics. LOC was the very first and simplest metric proposed in literature. McCabe [156] proposed another metric to quantify the complexity of the code for procedural languages known as Cyclomatic Complexity (CC) and it became quite popular among researchers. Many other metrics were also proposed from the year 1970 to the year 1990 such as Halstead Software Science Metrics [83], information flow metrics by Henry and Kafura [90], and Maintainability Index (MI) by Oman and Hagemeister [168]. MI was defined as a compositely variable consisting of Halstead Volume (HV), Number of unique operators and operands in source code, CC of the code, LOC per module in the source code and COMMENT(COM) and it becomes very popular during that period. Sneed and Meray [204] presented number of features which were reportedly associated with software maintainability such as impact rate, efforts, error rate and subjective evaluation. In the year 1990, Rombach [186] described through controlled experiment that the software systems written in OO language are more maintainable than the software systems written in a conventional language. A survey conducted by Daly [55] et al. in the year 1996 showed the same results. Hence, number of metrics that measure OO characteristics such as coupling, cohesion, inheritance and abstraction were proposed. We briefly describe these metrics in the next section.

1.5.2 Object Oriented Metrics

Many metric suites have been proposed to measure OO software including Chidamber and Kemerer [43] metric suite, Li and Henry [127] metric suite, Wei Li [126] metric suite, Chen and Lum [42] metric suite, Lee et al. [123] metric suite, Abreu and Carapuca [2] metric suite, Lorenz and Kidd [132] metric suite, Tang et al. [216] metric suite, Bansiya and Davis [15] metric suite and Henderson-Sellers [89] metric suite.

The first and very popular OO metric suite was proposed by Chidamber and Kemerer in year 1991 [43] famously known as C&K metric suite. It consists of six design metrics to measure various characteristics of OO paradigm such as inheritance, coupling, cohesion, abstraction etc. These six metrics are Coupling Between Object classes (CBO), Response For a Class (RFC), Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number Of Children (NOC), and Lack of Cohesion in Methods (LCOM).

Li and Henry [127] performed the empirical investigation of the metric suite proposed by Chidamber and Kemerer and pointed out that this metric suite does not take into account the structural complexity of the software. Hence, they added five metrics to the Chidamber and Kemerer metric suite namely Message Passing Coupling (MPC), Data Abstraction Coupling (DAC), Number of Methods (NOM), SIZE1 and SIZE2.

In 1998, Li [126] proposed a new metric suite to measure various features of OO paradigm and further performed their empirical validation. These are Number of Ancestor Classes (NAC), Number of Local Methods (NLM), Class Method Complexity (CMC), Number of Descendant Classes (NDC), Coupling Through Abstract data type (CTA) and Coupling Through Message passing (CTM). All metrics except NDC had been already empirically validated in a previous study conducted by Li and Henry [127] in predicting maintenance effort.

Abreu and Carapuca [2] proposed another metric suite famously known as MOOD metric suite. Original MOOD metric suite consists of six metrics Method Hiding Factor (MHF), Attribute Hiding Factor (AHF), Method Inheritance Factor (MIF), Attribute Inheritance Factor (AIF), Polymorphism Factor (PF) and Coupling Factor (CF).

Bansiya and Davis [15] made few enhancements into the MOOD metrics suite proposed by [2] and subsequently proposed QMOOD metric suite. It contains eleven metrics i.e. Design Size in Classes (DSC), Number of Hierarchies (NOH), Average Number of Ancestors (ANA), Data Access Metric (DAM), Direct Class Coupling (DCC), Class Interface Size (CIS), Measure of Aggregation (MOA), Cohesion Among Methods of class (CAM), Measure of Functional Abstraction (MFA), Number of Polymorphic methods (NOP) and Number Of

Methods (NOM).

Lorenz and Kidd [132] proposed metric suite consisting of eleven metrics to quantify software quality evaluation. The proposed metrics were categorized according to three groups i.e. class size metrics, class inheritance metrics and class internals metrics. Lorenz and Kidd's metrics were defined to measure the static characteristics of software design, such as the usage of inheritance, the amount of responsibilities in a class etc. The ten metrics were Number of Public Methods (PM), Number of Methods (NM), Number of Public Variables per class (NPV), Number of Variables per class (NV), Number of Methods Inherited by a subclass (NMI), Number of Methods Overridden by a subclass (NMO), Number of Methods Added by a subclass (NMA), Average Method Size, Number of times a class is Reused (NCR) and Number of Friends of a class (NF).

Bieman and Kang [25] introduced two new metrics to measure the cohesion between the class: Tight Class Cohesion (TCC) and Loose Class Cohesion (LCC). While calculating TCC and LCC, only those methods were considered which were visible. The impact of TCC and LCC on software quality was also established that higher the values of TCC and LCC, the classes would be more cohesive classes which lead to better software quality. They also consider the methods pairs using instance variables in common. Two methods are directly related if they both use either directly or indirectly a common instance variable. TCC is defined as the percentage of methods pairs, which are directly related. LCC is defined as the percentage of methods pairs, which are either directly or indirectly related .

Li and Henry [127] redefine LCOM in the year 1993 and named it as LCOM3. It is defined as the number of disjoint sets of methods. Each set contains only methods that share at least one attribute. The lack-of-cohesion in methods, LCOM3, is defined as the number of connected components in the graph. The model used in the LCOM3 metric is extended in by adding an edge between a pair of methods if one of them invokes the other.

Hitz and Montazeri [92] define a measure LCOM4 to measure the cohesion in terms of the connectivity between classes. This connectivity based metric is similar to LCOM3 metric proposed by Li [128] and applied when the graph has at least one connected component. In this case, some additional edges are added by taking into account the number of edges between connected component for method invocations.

Lee et al. [123] identified the difference between inheritance and non-inheritance based coupling and emphasized that since information flow-based coupling is based on method invocations, it takes polymorphism into account. Overall coupling was calculated as the sum of Non Inheritance information flow-based coupling (NIHICP) and Information flow-based inheritance coupling (IHICP).

In 1999, Tang et al. [216] presented a new set of metrics consisting of four metrics, Inheritance Coupling (IC), Coupling Between Methods (CBM), Average Method Complexity (AMC) and Number of Object/Memory Allocation (NOMA). The IC provides the number of parent classes to which a given class is coupled. The CBM provides the total number of new/redefined methods to which all the inherited methods are coupled. NOMA measures the total number of statements that allocate new objects or memories in a class. The AMC provides the average method size for each class. Pure virtual methods and inherited methods are not counted.

In 1996 to measure the cohesion between classes, Henderson-sellers [89] redefined the LOCM3 variable proposed by Li et al. [128] and proposed another metric LCOM5 which considers the number of methods referencing each attribute. This metric was based on the number of referenced instance variables. A class is considered as more cohesive when a large number of its instance variables are referenced by a method. In 1997, Number of coupling metrics for OO software has been proposed by Briand et al. [35] and three dimensions of coupling were identified, Interaction, Inheritance and Component coupling. These metrics takes into account the different OO design mechanisms provided by the C++ language: friendship, classes, specialization and aggregation.

In 1999, Benlarbi and Melo [20] defined a suite of five polymorphism metrics to measure run time polymorphism as well as compile time polymorphism as Overloading in stand-alone classes (OVO), Static Polymorphism in Ancestors (SPA), Static Polymorphism in Descendants (SPD), Dynamic Polymorphism in Ancestors (DPA) and Dynamic Polymorphism in Descendants (DPD).

Critical analysis of such metric suite were also conducted by many researchers such as Grady [80], Hitz [92] and Mayer and Hall [155]. They also carried out empirical investigations to verify and validate the proposed metric suite in terms of their effect on software maintainability. In one of the another study conducted by Tang et al. [216], Chidamber and Kemerer metric suite was analyzed using three C++ based industrial applications and found none of the metrics to be significant except RFC and WMC. Several OO metrics are captured using the design properties such as cohesion, coupling, polymorphism, abstraction and inheritance to predict important quality attributes such as fault-proneness, defect prediction, testing efforts, maintenance effort and productivity. Empirical analysis of OO metrics is performed and evidence are collected in order to establish strong relationship between OO metric and software quality by Malhotra [134].

1.5.3 Software Maintainability Prediction

Belady and Lehman [19] first introduced this word in the year 1969 stating that maintenance phase consists of all those activities which are performed once the development is over and product is handed over to the customer. It was a common perception that maintenance means fixing the errors, however, they demonstrated that maintenance is evolutionary development and software system gets complex over a period of time. Apart from fixing the errors during the maintenance phase, some activities are also needed to reduce the complexity of the code such as restructuring of the code, removal of duplicate code etc.

Between the period from 1969 to 1990, researchers like Yau and Collofello [232] and Yau et al. [233] concentrated more on good programming rather than designing to make the software more maintainable. Swanson [215] also pointed that maintainability of the software system is important as it consume the largest part of the total project cost. If the maintainability of the software system is not good, it cannot accommodate changes quickly which means business opportunities would be lost forever. Problems in maintenance process were also identified by Martin and McClure [150] and Nosek and Palvia [167]. A survey was conducted by Lientz and Swanson [129] in which they exposed that very high fraction of total project cost is expanded on software maintenance. They also categorized the types of maintenance into four different activities as Adaptive, Perfective, Corrective and Preventive maintenance as discussed previously in section 1.3.1.

A model was proposed by Banker et al. [14] for a maintenance project in which the software written in COBOL was studied. They found that apart from Function Point (FP) and LOC, the maintenance also depends upon many factors such as skills of the programmer, expertise and experience of the programmer, the use of a structured design methodology, the availability of a fast turnaround programming environment, density of the non-benign GOTO statements, Method used by the programmers and the response of the programming environment.

The metric suite proposed by Chidamber and Kemerer [43] and Li [126] became quite popular and evaluated analytically in several studies by many researchers including Aggarwal et al. [6], Briand et al.[34], Bandi et al. [13], Dagpinar and Jahnke [53], Elish and Elish [62], Kaur et al. [106], Koten and Gray [118], Malhotra and Chug [137], Tang et al. [216], Thwin and Quah [219], and Zhou and Leung [237]. In all of these empirical studies, it is observed and empirically validated that the OO metrics can be used to measure the structural quality of a code in general and maintainability in particular.

Li and Henry [127] attempted as early as in the year 1993 to validate Linear Regression (LR) model using two proprietary datasets for evaluating the relationship between Chidamber and Kemerer metric suite and maintainability, theoretically as per the framework proposed by Kitchenham et al. [111]. Their results indicated that approximately a total of 90% total variance in maintenance effort is accounted by Chidamber and Kemerer metrics. In their study, the datasets of two proprietary software systems User Interface Management System (UIMS) and Quality Evaluation System (QUES) was used.

Stavrinoudis et al. [210] explored the relationship between metrics and maintainability by conducting a large-scale survey in five major projects. Questionnaire was generated consisting of every factor pertaining to maintainability with an aim to measure the external attributes. Programmers also evaluated each module based on various attributes such as consistency, simplicity, conciseness, expandability, correctability etc. Internal measurements were also collected using automatic tools based on software measurement and metrics environment. They found high correlation between the software metrics score and the opinion from the programmers with regard to maintainability.

In 2002, Aggarwal et al. [5] proposed multiple parameters for software complexity measurement consisting of three factors i.e. Readability of Source Code (RSC), Documentation Quality (DOQ) and Understandability of Software (UOS). Fuzzy based model was created to measure these three factors and they were found to be highly correlated to software maintainability by the authors.

Fioravanti and Nesi [68] presented a framework for metric analysis and identified that which metric would be better ranked for its impact on the prediction of adaptive maintenance of OO systems. They observed that as the traditional metrics neglect the information about class specialization in OO paradigm, hence not fit for use. In order to find the complexity of a given class, they defined a new Metric Class Complexity (MCC) as the sum of External Class Description (ECD) and Internal Class Implementation (ICI). ECD is the complexity of the class due to class definition including method interface definition whereas ICI is the complexity of the class due to method implementation.

Bandi et al. [13] conducted an empirical study to validate three complexity metrics namely Operation Argument Complexity (OAC), Interface Size (IS) and Interaction Level (IL) and verified that these three metrics can be used successfully to assess respective maintenance time.

Software maintenance for sure plays a determinant role in finding the total project cost of any software. As per the definition of IEEE Standard 828-1998 [63], we can measure maintainability of the software during operational phase only, however, because of the fact that precaution is better than cure, it's more desirable to determine the maintainability during the development phase itself. In this regard, as suggested by Jorgensen [101] and Lucia et al. [133], most important method to crack this issue is by developing prediction models which can be deployed during the early phases of the software project development using OO metrics. Jorgensen [101] also suggested that prediction of software maintenance reduces the future maintenance efforts because developers can improve the design or coding by identifying the determinants of software maintainability. It also provides warning signs to managers well in advance with information for making effective planning and taking corrective actions using their valuable resources more judiciously. Lucia et al. [133] specifically worked on corrective maintenance where the data was collected from five maintenance projects carried by industry. A prediction model was constructed using Multivariate Linear Regression (MLR) technique to estimate the costs of a project by taking into account various task associated with corrective maintenance work only.

In two research studies Jin & Liu [100] and Misra [162], datasets developed from the students' software were used to validate the prediction models. In the first study by Jin and Liu [100] prediction model was validated using the datasets collected from the software systems developed by graduate students. Their results show that when Support Vector Machine (SVM) is combined with clustering for the purpose of maintenance effort predictions, the correlation between Chidamber and Kemerer metric suite and maintainability was found to be as high as 0.769 which is statistically quite significant. In the study conducted by Misra [162] the datasets were collected from pools of 50 C++ programmes and deployed them in LR techniques. Intuitive analysis based model using more than twenty design and code measures was also used. He concluded that two most important measures should always be kept in mind while coding. Firstly, any functions (modules) should not be more than two screens long. Secondly, spaghetti code which looks exactly like spaghetti with the presence of goto like unstructured statement should be as avoided as much possible. He also found out that it's the Average Method size in Lines Of Code (AMLOC) metric which works as significant predictor of the subsequent maintainability of any given software. Strong negative influence of Method Hiding Factor (MHF) on maintainability was also reported in his study which means as the level of abstraction increases in a class, consequently, its maintainability decreases. Interestingly this observation was found to be just contrary to the intuitive analysis performed by the author. It was hypothesized that increasing method hiding in a class will result in less maintenance effort in the future, i.e. improve its maintainability. However, this may not be the case when the level of abstraction is too high such that important aspects of the design are lost. As a consequence, maintenance effort will be expensive in the system evolution phase.

Dagpinar and Jhanke [53] suggested that instead of design level metrics of structure languages, OO metrics should be used for precise capturing while making any prediction model. Further, they also recorded the significant impact of direct coupling metric and size metric on software maintainability instead of other metrics such as cohesion, inheritance and indirect coupling.

Xia and Srikanth [227] suggested a change impact dependency measure for predicting the maintainability of source code. They suggested that dependency is an essential aspect to consider for designing the architecture of complex systems and it directly affects the maintainability and many another quality attribute of software. They further added that the notion of dependency is vague and subject to different interpretations. For some it's static while for others it is considered as dynamic.

Aggarwal et al. [4] proposed a fuzzy model to measure the software maintainability. They stated that apart from other metrics proposed earlier, maintainability also depends on two other factors unnoticed i.e. average number of live variable and average life span of a variable. They proposed a model that considers the effect of these two factors along with two traditional variables i.e. Comments Ratio and Average Cyclomatic complexity (ACC). One of another empirical study conducted by Aggarwal et al. [6] suggested that Artificial Neural

Network (ANN) is a very useful technique for maintainability predictions. Metric suite proposed by Li and Henry [127] is used in their study and it was found that 72% accuracy can be achieved while using ANN. The study also suggested that since the performance of ANN is mainly dependent on the data on which it is trained, it is the matter of availability of the suitable dataset on which accuracy of the prediction model depends.

Zhou and Leung [237], employed comparatively new modeling technique named Multiple Adaptive Regression Splines (MARS) to build software maintainability prediction model. They compared their results with artificial network models, regression tree models, and support vector models. They found that the proposed MARS model could achieve the value of R-square as 0.837 with QUES dataset and 0.656 with UIMS dataset which is quite competitive.

Koten and Gray [118] further validated Bayesian Belief Network (BBN) (also known as Bayes Net, Causal Probabilistic Network, Bayesian Network or Belief Network) using 10-cross-validation on the UIMS and QUES dataset given by Li and Henry [127] and found it to be significantly better model in terms of determining prediction accuracy. They observed that for the UIMS dataset, the BBN model outperformed the regression tree model as well as multiple linear regression model. For the QUES dataset, even though BBN was not as good as it was for the UIMS dataset, but still worked out be reasonably accurate against other regression models. They concluded that performance of BBN mainly depends upon the characteristics of datasets.

Prasanth et al. [176] stated that maintainability could be improved through risk analysis and introduced a risk-based approach to find and fix the most important problems as quickly as possible. In their study, risk was characterized by combination of two factors, the severity of a potential failure event and the probability of its occurrence.

Elish and Elish [62] corroborated relatively new technique, TreeNets for software maintainability predictions. They compared their results with other prevalent models and found them to be very cost effective with more prediction accuracy on UIMS and QUES datasets.

The results were analyzed using various prediction accuracy measures such as Magnitude of Relative Error (MRE), Mean Magnitude of Relative Error (MMRE), Mean Absolute Relative Error (MARE) and Prediction accuracy with less than 25% error known as Pred(0.25).

Kaur et al. [106] conducted another study and analyzed the prediction capability of the Adaptive Neuro Fuzzy Inference System (ANFIS) technique on UIMS and QUES dataset using Holdout validation and the outcome showed better performance as compared to previous studies for making an efficient maintenance effort prediction model.

Again one of the important empirical studies was conducted by Ping [172] for the purpose of software maintainability prediction using Hidden Markov Model (HMM). Health index of the product is adjudged by assigning the weight on the process of maintenance behavior invested over time and further related to software quality metrics using prediction model. Every time probability of corrective maintenance time is compared with the threshold value to determine the point when the software model is considered as obsolete.

A model based on multiple classifiers combination was proposed by Ye et al. [234] which has three parts: attribute selection, model training and model interpretation. In this model, genetic algorithm (GA) for attribute selection was used followed by Decision Tree (DT) algorithm for rule extraction during the training process. More than 300 classes of open source C++ software system were downloaded and used for the validation process.

Statistical comparison of various modeling techniques for software maintainability was performed by Kaur and Kaur [105] using different regression and machine learning techniques. Commonly used datasets UIMS and QUES proposed by Li and Henry [127] and used by many researchers [53, 62, 106, 118, 137, 237] was also used in their study. Friedman test was also used to rank all prediction modeling techniques on the basis of prevalent accuracy measures such as MMRE, RMSE, Pred(0.25) and Pred(0.30).

Apart from traditional metrics, maintainability prediction was performed using new predictor metrics by Kaur et al. [107]. Four open source software namely Lucene, JHotdraw, JEdit, and JTreeview were used for conducting the empirical study. Five important machine learning techniques, multi layer perceptron, naive bayes, logistic regression, bayes network and random forest classifiers were used to identify the software modules that are difficult to maintain.

The researchers are always constrained against non-availability of genuine datasets to conduct their validation studies and test newer prediction models of determining maintainability. However, the datasets of two proprietary software systems UIMS and QUES made public by Li and Henry [127] opened the doors for additional research studies to validate the maintainability prediction models and therefore majority of researchers used this datasets in their experiments. The datasets used in proprietary software also have a certain constraint on the generalizability of the results because each system has been developed in different language with different environment settings. The functioning of various machine learning techniques also differs significantly because they produce contrasting results on different software datasets. Apart from this, the generalization also cannot be done as different prediction accuracy measures are used in different studies, which leads to the contrary results. To overcome this problem Myrtveit et al. [165] suggested that more reliable research procedures must be developed before believing on the outcome of any one software prediction model.

In open source software, practitioners across the globe are allowed to change, expand and redistribute the newly created version without any requirement of the license [196]. Well known successful stories of the open source software systems include Linux operating system and Apache web server as reported by Samoladas et al. [192]. Changes in such software are made continuously by a large number of software developers in order to improve their functioning and usefulness. Estimating the maintainability of open source software systems is very challenging task due to the lack of technical support and the absence of documentation. Studies on observing the maintainability of open source software systems are very limited. Only one study by Zhou and Xu [238] is found on open source software systems and that too with the application of statistical predictive modeling. Even though Ramil et al. [180] has compiled many empirical studies which were conducted on open source software systems, however, all those studies stress on intuitively judging the software evolution based on the characteristics instead of creating a prediction model based on mathematical functions.

The evolutionary techniques came in to existence in the early 1960s when four scientists namely G.E.P. Box, G.J. Friedman, W.W. Bledsoe and H.J. Bremermann, developed biological evolution-inspired techniques for function's optimization and machine learning independently [7, 125]. Over the next two decade, evolutionary techniques were refined and by mid-1980s their usage was successfully proved in broad range of subjects' right from bin-packing, graph coloring to more technical such as pattern recognition, classification and optimization. The evolutionary technique leads to the evolution of populations consisting of potential solutions which are optimized and better suited to their environment [159].

Excellent bio-inspired techniques such as Ant Colony Optimization (ACO), Bees Algorithms, Cuckoo Algorithms (CA), and Particle Swarm Optimization (PSO) etc [7, 11, 125, 159] are proposed in literature. Some of them are used effectively in many areas of software engineering like the prediction of development effort [12], prediction of maintenance effort [16], prediction of preventive maintenance [213], and identifying effective software metrics [222], however, their use in software maintainability prediction is found to be extremely limited.

Evolutionary technique was used by Basgalupp et al. [17] for predicting software maintenance efforts. In this study, numerous shortcomings of multivariate linear regression technique were highlighted and empirically it was proved that the results of evolutionary induced decision trees are better then greedily induced decision tree. Two problems were also highlighted in the study, first, if experiments were carried out on small or medium-sized systems, relationships cannot be validated for industrial environment. The second, that if large and complex systems cannot be analyzed, we will not be able to calculate their metrics values. Conclusion is drawn that without solving these difficulties, the set of metrics cannot be applied for software maintainability predictions.

1.5.4 Refactoring to Enhance Software Maintainability

The term refactoring was introduced by Opdyke [170] as the process of restructuring software code in the context of the OO paradigm. Mens and Tourwe [158] had compiled a number of empirical studies in which the effects of selected refactoring methods on various software quality attributes were investigated. Bravo [32] had developed an automated tool in Smalltalk Open Unification Language (SOUL) which provides an interface and helps the developer in identifying bad smells present in the code and subsequently selecting and applying respective refactoring methods. Stroggylos and Spinellis [212] have studied four open source software and analyzed the effect of refactoring on various software metrics. They concluded that although it is expected from refactoring process to improve quality but when it is measured through the OO metrics in real life systems, the results are not as expected.

‘Refactoring’ word is originated from ‘factor theory’ of mathematics i.e. when any expression is converted into its equivalent factor, it becomes clearer. Refactoring is a kind of reorganization of the code in such a way that beginning and end products must be functionally identical. Code becomes simpler, cleaner and elegant which further enhances the quality of the software. Bois et al. [29] proposed the impact of refactoring on the quality of the software code and set few guidelines to be considered while refactoring the system. Bois and Mens [30] observed the effect of refactoring process on cohesion and coupling while adhering to those guidelines.

Fowler [69] specified 72 different refactoring methods in his book ranging from simple changes such as Extract Local Variable to more complex one such as Extract Class. Kataoka et al. [104] studied effects of refactoring on coupling metrics in detail with an example program.

Moser et al. [163] strongly recommended refactoring as it improves reusability. They validated their statement through an empirical study using XP development environment.

Geppert et al. [77] empirically investigated the effects of refactoring on changeability using three factors comprising of efforts, defect rate and scope for change. Wilking et al. [226] also conducted an empirical study to evaluate the effects of refactoring process on modifiability and maintainability. Although few studies have not directly evaluated the effects of each refactoring method separately on internal quality attributes or external software quality attributes, nonetheless their cumulative effects are being discussed. Xing and Stroulia [228] have discussed most frequent types of refactoring and high-level design requirements which are mandatory for refactoring process using well known Integrated Development Environment (IDE) platform Eclipse. During the maintenance phase, the code is modified many times and in this process the code quality deteriorates [69].

Refactoring is a process during which, its internal structure is improved, complexity is reduced and external behaviour remains the same [170] for any given software. Various advantages of refactoring include enhancement of new features and improvements in understandability, readability and maintainability by enforcing fine-grained encapsulation into the code.

1.5.5 Effect of Agile Methodology on Software Maintainability

Royce [187] presented a research study highlighting the problems associated with sequential developments wherein, the varying characteristics between software and automobile assembly line were compared and he asserted that the software should not be treated like typical automobile assembly line production. Over a period of time, many models such as Iterative Enhancement Model (IEM), incremental model, spiral model, prototype model etc were thus evolved by various researchers. Though introduced in the later part, however, agile methods instantly gained tremendous success in the commercial industry from the late nineties onward in software development process[103, 214].

The word “Agile” has its origins from the Latin word “agilis” meaning a swift movement. When applied to the software development domain, it represents a development approach which focuses on the addition of new features in the software in a planned and sequential manner that is delivered in small demonstrable segments and not altogether in the end. Agile development is based on seven core values i.e. empower the team, eliminate waste, see the system as a whole, deliver as fast as possible, build integrity in the team, decide as late as possible, amplify learning and create the inherently cohesive system. The requirements and their respective solutions gradually develop through a systematic alliance between cross-functional teams that can ably organize themselves without external help. It also encourages planning which can accommodate changes, a development that evolves with time, early delivery and uninterrupted enhancement. Further, it persuades quick and flexible response to change.

Many agile based research studies have been extensively conducted in academia and industry on medium and large-scale software projects using critical success factors [44]. The agile manifesto was written by renowned software engineers across the globe [70]. Various methodologies covered under the agile manifesto were XP by Beck [18], Crystal by Cockburn [47], design patterns by Cunningham [52], refactoring by Fowler [69], clean code by Martin [152], and scrum by Schwaber and Sutherland [199]. The main statement was written in agile manifesto to represent the gist of it is as follows:

"Individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, responding to change over following a plan".

Cohn and Ford [48] have discussed the positive changes observed by an organization when the agile methods were introduced. Boehm [28] elaborated the ways the agile needs to be implemented. Soundararajan et al. [206] asserted that as customer needs are quickly evolving, the vendor must introduce flexible methods of software development. Knippers [113] investigated the effects of the agile methods on software maintainability and concluded that the agile reduces defects as well as program complexity and supports long-term maintainability.

Rising and Janoff [185] specifically introduced scrum method of the agile methodology into an organization wherein teams were further divided into small teams to observe the enhancement in software quality. When and how the scrum should be introduced by any organization was illustrated by famous scrum trainer Michael James [97]. Svensson and Host [214] proposed the use of agile process based XP in an evolutionary and maintenance software development environment. Importance of Definition of Done (DOD) in a scrum sprint was illustrated by Davis [56] which empirically proved that DOD brings all the members of a team on a common page and understanding which ensures that when a team moves a task from the in-progress section to the done section, all the highlighted points should have been taken care of thoroughly. Pries-Heje et al. [178] proposed new methodology called as Scrum Code Camp (SCC) and emphasized that cost should not be a criteria for choosing an agile team over the other; instead team work spirit, technical and functional experience, and the attitude of approaching a problem are more important factors which make a difference.

Very few studies were found which were aimed to observe the effects of the agile methodology on maintainability [75, 112, 113]. One such study was conducted by Knippers [113] who investigated the effects of agile methods on software maintainability and concluded that agile reduces defects as well as program complexity and supports long-term maintainability. Further, there is no empirical study which quantitatively compares the product developed using the agile methodology as compared to traditional IEM development model.

1.6 Goal of the Thesis

Jones [40] has given a comparison that in the year 1950, out of 1100 people deployed in software industry only 100 (roughly 10%) were engaged in maintenance but by the year 2025 out of 5,500,000 people potentially deployed in software industry approximately 4,250,000 people would be engaged in maintenance task (roughly around 77%). This really gives the researchers an inspiration to think as to how we can make the software more maintainable so that we can reduce the personnel engaged in it to the extent possible. There is a large shortage of software personnel due to the burst of mass update maintenance work required.

It is a bitter fact that in software engineering industry, the majority of the time and effort is consumed in software maintenance actions rather than on development. Somehow, if we can predict the maintenance effort of developed software on the basis of the characteristics of the product, we would be better able to manage the product and make its optimum use.

With reference to the problem cited above, this thesis focuses on exploring various techniques, models and methods for improving software maintainability. In other words, we have constructed various maintainability prediction models by establishing a relationship between OO metrics and maintainability in this research study. Besides this, we have also explored the application of various methodologies such as agile and refactoring on software maintenance for identifying their utility in enhancing the software maintainability. For the purpose of empirical validation, we have used a number of software systems obtained from proprietary systems as well as open source software repositories.

The immediate goal of this research is to measure the structural attributes of the OO software using many important concepts such as coupling, cohesion, encapsulation, abstraction, inheritance and polymorphism and relate it to the external quality attributes software maintainability. The main motivation behind this goal is to be able to assess quality of software in the early phases of the SDLC and use them for predicting maintainability which would greatly facilitates proper resource planning well in advance. The objective of this thesis is also to analyze at various aspects of software maintenance, hence, this thesis identifies various procedures that are traditionally meant to improve the software maintainability such as agile methodology, refactoring, program comprehensions, re-engineering, reverse engineering and impact analysis.

We conducted extensive empirical studies to understand the effects of two methods i.e. agile methodology and refactoring on software maintainability. One of the main properties of the software in a real-world environment is that it continuously evolve during the maintenance phase when new requirements are added. The code becomes more complex due to this small and unplanned repairs. Hence, the quality of the software deteriorate as it drifts away from its original design. Even the use of good software development model also cannot solve this problem because their increased capacity is basically used to implement new requirements within the same time frame.

In order to cope up with this kind of spiral of complexity, there is an urgent need for refactoring which reduces the software complexity by incrementally improving the internal software quality. In this thesis, the central hypothesis is developed after detailed deliberation as it is extremely vital to formulate a correct hypothesis in an endeavor to ensure correct outcome. The central hypothesis of this thesis is specified as follows:

"The classes of given software, which needs more maintenance efforts can be identified as a function of internal quality attributes using an efficient prediction model, thereby assist project managers in better planning of resources to enhance efficiency and reduces the maintenance cost. The effects of various prediction techniques to augment software maintainability can be investigated, thereby suggesting project managers to utilize them in an optimum manner in order to increase the overall performance and efficiency."

Thus, the aim of this research has been three folds as shown in figure 1.8. First, is to develop an efficient prediction model to predict software maintainability for different kind of datasets such as proprietary, open source, data intensive applications etc. Second, is to identify OO metrics which are more effective for the prevalent data intensive applications. Third, the goal of this research has been to study the effect of various techniques such as the agile methodology, refactoring etc. on software maintainability. Generally, we analyze the predictive capability of machine learning and evolutionary techniques for predicting maintenance efforts. Hence, the summary of the goals of the work is divided into following parts:

1. Perform an extensive review of existing literature to identify the association between OO metrics and maintainability with following objectives:

    - Study the use of prediction models for maintainability in the early phases of development.

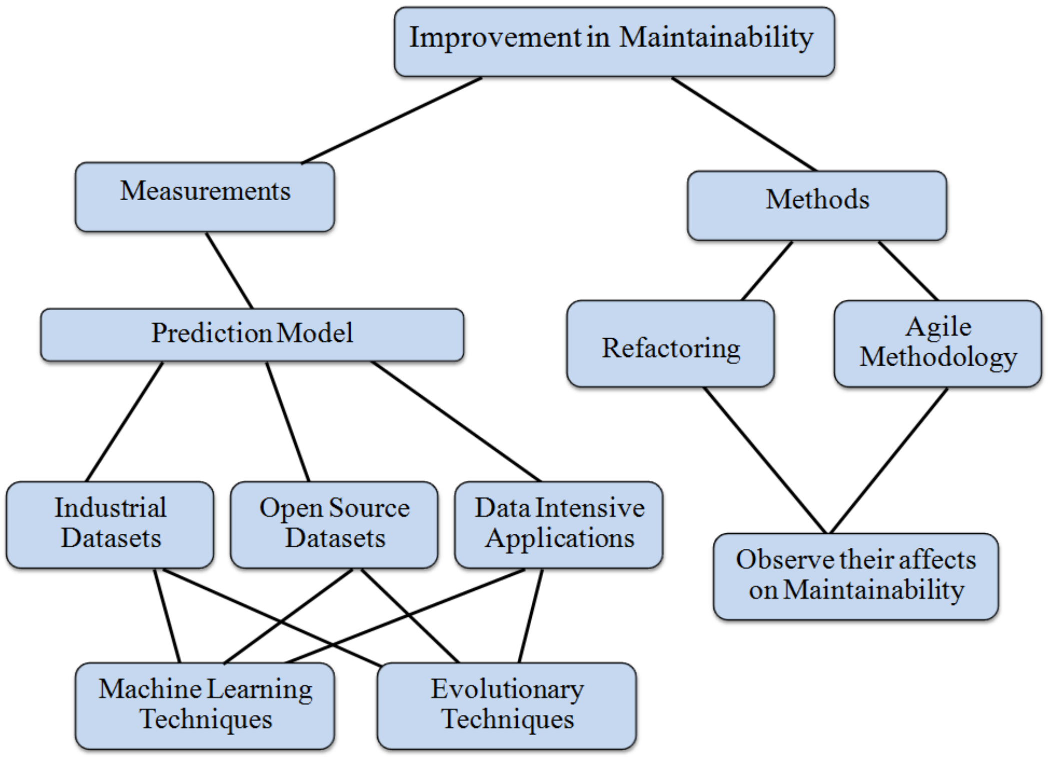

Figure 1.8: Ways of Improvements in Software Maintenance

- Identify maintenance process efficiency to keep the maintenance cost under control.
- Comparing the performance of various maintainability prediction models in terms of accuracy.
- Identify the advantages and disadvantages of various prediction models over each other.
- Identify the software metrics which can be used in prediction model making process.
- Identifications of the existing gaps for future prospect of research in the field of software maintainability.

2. Develop new models for prediction of software maintainability using machine learning techniques and evolutionary techniques with following objectives:

- Empirically validate the relationship between design metrics and subsequent maintainability.
- To ascertain empirically if evolutionary techniques can be applied for software maintainability predictions.
- Identify whether evolutionary techniques perform better or worse than statistical and machine learning methods.
- To propose the use of a relatively new hybrid technique Group Method of Data Handling (GMDH) and compare its prediction accuracy with other machine learning models.

3. Study most commonly used design metrics and determine the most suitable metric suite for data intensive applications with following objectives:
    - Study the most commonly used OO metrics.
    - Identify the suitability of OO metrics for data intensive applications.
    - Develop new metrics based on data base connections and schema complexity to capture the design characteristics of databases intensive applications.
    - Empirically validate the effectiveness of following two new metrics.
4. Investigating the maintainability prediction of the open source software using machine-learning techniques with following objectives:
    - To study the impact of OO metrics on maintainability in the context of open source software systems?
    - The comparative performances of machine learning techniques for maintainability prediction using open source software systems?
    - To determine which pairs of machine learning techniques are performing statistically better from each other techniques in terms of maintainability prediction with the help of post-hoc analysis.

5. Investigating the effects of refactoring on software maintainability with the following objectives:
    - To identify the bad smell present into the source code and identify suitable refactoring method.
    - Apply respective refactoring method to remove the particular bad smell present into the source code.
    - Assess the change in the values of OO metrics due to the application of specific refactoring method.
    - Finally, map the change in the values of OO metrics to maintainability using prevalent software quality models.
6. Understanding the impact of the agile methodology with following objectives:
    - To study one non-agile method, IEM and one agile method, scrum for product development.
    - Compare and analyze the performance of scrum with traditional IEM for product development.
    - Validate the results on industrial applications.

1.7 Organization of the Thesis

The objective of the thesis is to build an efficient software maintainability prediction model, explore the utility of design metrics for emerging applications and analyze the effectiveness of various methods and techniques for the improvement in software maintainability. This section presents the organization of the thesis. **Chapter 2** discusses the framework of research methodology adopted in carrying out this work. A literature review of existing studies and the prevailing research gaps are identified in **Chapter 3** lead us towards the scope of this research. Next, **Chapter 4** presents the use of machine learning techniques for software maintainability prediction. **Chapter 5** identifies the reasons why the currently used OO metric suite is not suitable for highly data intensive applications and to suggest new metrics for the same. In this regard, few commercial applications are used for the validation of new metric suite. In **Chapter 6** a bench-marking framework for maintainability prediction of open source datasets is presented. Application of evolutionary techniques for constructing the maintainability prediction model is presented in **Chapter 7**. There are certain methods which are applied to enhance the maintainability of the software. Refactoring is also one of the technique which is the part of maintenance phase. In this technique, the design of software is improved and complexity is reduced without affecting its external behaviour. In order to understand the effects of these methods, **Chapter 8** presents various refactoring methods and subsequently analyzes its effects on software maintainability. **Chapter 9** presents the agile methodology and its effects on software maintainability by comparing the methodology with the traditional existing methodology. Finally, conclusions obtained from the thesis are described in **Chapter 10**. Next subsections present the brief description about each chapter.

Chapter 2: Research methodology followed in the current study is explained in detail in this chapter. In the beginning, the research problem is defined and the goals are fixed. Further independent and dependent variables used in this study are specified. Various OO metrics are described which are used in order to capture various attributes of OO paradigm such as cohesion, coupling, abstraction, polymorphism and inheritance. For the purpose of empirical validation, datasets used in the current study are also explained in the later part of the chapter. The ten cross-validation techniques used in this study for validating the prediction models have also been explained. Various prediction accuracy measures selected to evaluate the outcome of prediction models are also discussed. A range of statistical tests followed by post-hoc analysis used in this research in order to confirm the reliability of empirical experiments are summarized in this chapter. Various techniques such as agile and refactoring used for the improvement in software design are also discussed in brief at the end of this chapter.

Chapter 3: Systematic review of the existing studies related to software maintainability since January 1991 to October 2015 has been conducted and reported in this chapter. Many Research Questions (RQ) were formed in order to address various issues related to software maintainability. All the available studies are being compiled in structured form and analyzed through numerous perspectives such as the use of design metrics, prediction model, tools, data sources, prediction accuracy etc. Review results are analyzed, compiled and research gaps are identified providing further directions.

Next, **Chapter 4**: In an attempt to quantitatively address the issue of software maintainability, main purpose of this chapter is to propose few machine learning techniques with an objective to predict software maintainability and its evaluation. The prediction model is constructed using machine learning techniques, GMDH, GA and Probabilistic Neural Network (PNN) with Gaussian activation function. The results were compared with prevalent prediction models using various prediction accuracy measures such as MRE, MARE, MMRE and Pred(0.25).

Chapter 5: In this chapter, new metric suite for highly data intensive applications is presented. Certain metrics are developed where main focus is to measure the amount of database handling. Importance for understanding software as well as database is highlighted. In this chapter, we propose a new metric suite and redefined the relationship between design metrics with maintainability. The proposed metric suite is evaluated, analyzed and empirically validated using five proprietary software systems.

Chapter 6: A bench-marking framework for maintainability prediction of open source datasets is presented in this chapter. In this chapter, we have analyzed the effectiveness of machine learning techniques for the maintainability prediction using open source software. Large-scale empirical comparisons of thirteen classifiers over seven public domain datasets were conducted followed by extensive statistical tests to establish the confidence on the performance of one machine learning technique over another.

Chapter 7: The application of evolutionary techniques for constructing the maintainability prediction model is presented in this chapter. The significance of the evolutionary techniques has substantially increased in recent time due to their capability of maximizing the quality function. An empirical study for exploring the application of evolutionary techniques for software maintainability predictions is discussed. The performance of evolutionary techniques with traditional ones have been compared extensively in this chapter.

Chapter 8: The objective of this chapter is to examine the effects of refactoring process on maintainability. In this regard, few important refactoring methods are classified on the basis of their effects on software maintainability using the OO metrics. Five real-time systems were taken and their OO metrics were calculated and analyzed before the application of refactoring as well as after the application of refactoring. In this chapter, opportunities of refactoring in large code is identified and fine line is drawn to maintaining a perfect balance between re-engineering and over engineering.

Chapter 9: This chapter provides an insight into another approach agile methodology. In this chapter, the impact of the agile framework using scrum on the deliverable to the customer is compared to the IEM. The same software product is developed using scrum methods as well as IEM model and various metrics were used to compare both the products quantitatively as well as qualitatively from the maintenance point of view such as Number of defects identified, Time and stage of SDLC when the defect was identified, Number of change requests received, and features rolled using both methods.

Chapter 10: Finally, in this chapter, application of this research work is presented. In the end, we present conclusions along with directions for future research to be carried out in this field.

Chapter 2

Research Methodology

2.1 Introduction

Research methodology is the process to systematically investigate and solve the undertaken problem. In order to validate the relationship that exists between software design metrics and maintainability in our research, the first step is to formulate the research methodology so that empirical studies can be conducted accordingly. In this chapter, various aspects of research methodology are presented like theoretical foundation of the procedures, well-defined inputs to the empirical studies, prediction modeling, accuracy measures, cross-validation techniques and various statistical tests to establish the confidence into the received outcome.

The chapter is organized as follows: Section 2.2 presents the research process followed in this work to carry out various empirical studies. Section 2.3 defines the research problem. Development of hypothesis is presented in section 2.4. Independent and dependent variables are defined in Section 2.5. The process of empirical data collection and various sources for data collection are presented in Section 2.6. Section 2.7 describes the pre-processing and data analysis process. Features subset selection techniques are presented in section 2.8. Criteria for the selection of prediction modeling techniques is presented in section 2.9. Various learning techniques used for creating prediction model are discussed in Section 2.10. Section 2.11 briefly explain various prediction accuracy measures used in this research for performance evaluation of the created prediction models and Section 2.12 describes cross-validation methods used to get the unbiased results. Section 2.13 provides various statistical tests which are used to check the significance of the outcome in this research and finally discussion is provided in section 2.14.

2.2 Research Process

This section presents the process adopted in this research to carry out various empirical studies. The summarized process is shown in figure 2.1 which begin with identifying the underline research problem and formulation of the central hypothesis. Independent and dependent variables are selected and the sources of data collection are finalized. The collected data goes through the rigorous process of cleaning, filtering and noise removal before it can be used for making prediction model. Redundant features are subsequently removed using feature subset collection methods so that the reduced size of the data enhances speed and accuracy of the prediction model. Proper machine learning techniques are chosen based on certain characteristics and the prediction model is developed. Appropriate performance measures are selected which can be applied in the research in order to evaluate the effectiveness of the results. Models developed using various machine learning techniques are subjected to statistical analysis followed by Post-hoc evaluation to verify the significance of the outcome. Finally as in the case of any empirical study, conclusions are drawn from obtained evidences.

2.3 Define Research Problem

The very first step is to define the research problem as it helps in determining the exact research methods and procedures to be used. This section presents the research problem in the form of questions pertaining to research in the context of software maintainability. Following RQs are addressed in this study:

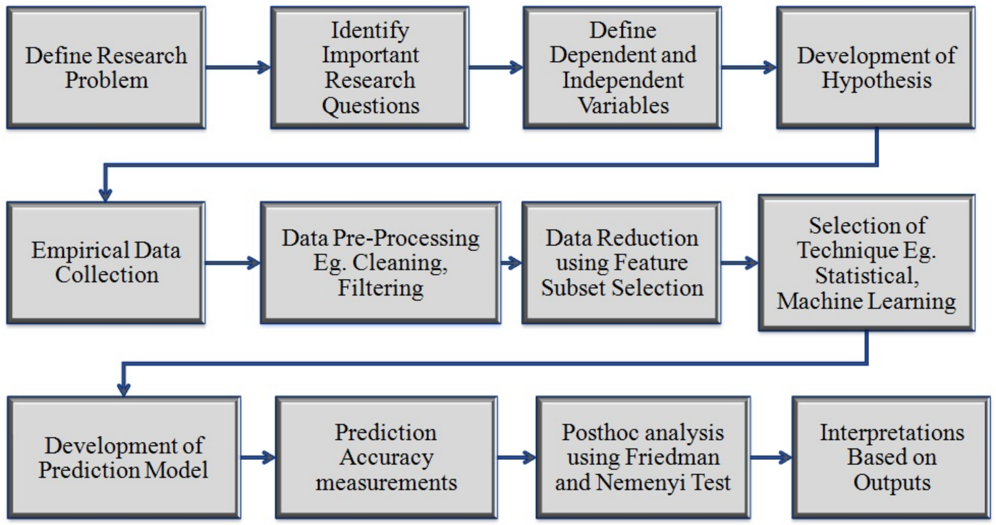
Define Research Problem
Identify Important Research Questions
Define Dependent and Independent Variables
Development of Hypothesis
Empirical Data Collection
Data Pre-Processing Eg. Cleaning, Filtering
Data Reduction using Feature Subset Selection
Selection of Technique Eg. Statistical, Machine Learning
Development of Prediction Model
Prediction Accuracy measurements
Posthoc analysis using Friedman and Nemenyi Test
Interpretations Based on Outputs

Figure 2.1: Research Methodology

- What is the current state of research on software maintainability prediction for improving quality of software?
- What is the qualitative performance of different machine learning techniques in prediction modeling with reference to software maintainability.
- Which metric suite is more suitable for database intensive applications?
- What is the performance of model prediction using open source and widely used software?
- What is the effective way of validating the prediction models once they are trained?
- Can we explore the impact of the agile methodology on software maintainability.
- Investigate if the effects of well known refactoring methods are positive or negative on software maintainability ?

2.4 Development of Hypothesis

After literature survey, research should carefully state the hypothesis to be tested in the study. Hypothesis is tested on the sample data. On the basis of the result from the sample, a decision concerning the validity of the hypothesis (accept or reject) is made. In this thesis, two hypothesis will be tested. First is concerned with the prediction technique used in creating prediction models and second is concerned with the methods/treatments provided into the code to improve software maintainability.

2.4.1 Hypothesis 1

For each prediction technique M (such as GMDH, GGRN, GGAL etc), the following hypothesis will be tested related to data analysis methods:

- Null Hypothesis: Prediction technique M does not outperform the other techniques in predicting maintainability.
- Alternate Hypothesis: Prediction technique M outperform the compared technique in predicting software maintainability.

2.4.2 Hypothesis 2

For each method T (such as Refactoring, Scrum etc), the following hypothesis will be tested related to the improvement in design to ease the software maintenance process:

- Null Hypothesis: Method T does not improve the software maintainability.
- Alternate Hypothesis: Method T improve the software maintainability.

2.5 Define Variables

In order to measure the OO software system, guidelines provided by Chidamber and Kemerer [43] and Li and Henry [127] are used in this study. The OO metrics capturing the features pertaining to cohesion, coupling, abstraction, inheritance and size are taken into account in this study. In the next two subsections, we define the independent variables and dependent variables selected in this study.

2.5.1 Independent Variables

As we wanted to measure OO features, size as well as complexity of the system, various metrics were carefully selected from the metric suites proposed by various researchers Chidamber and Kemerer [43] metric suite, Li and Henry [127]metric suite, Bansiya and Davis [15] metric suite, Henderson-Sellers [89] metric suite, Martin [151] metric suite, and Tang et al. [216] metric suite.

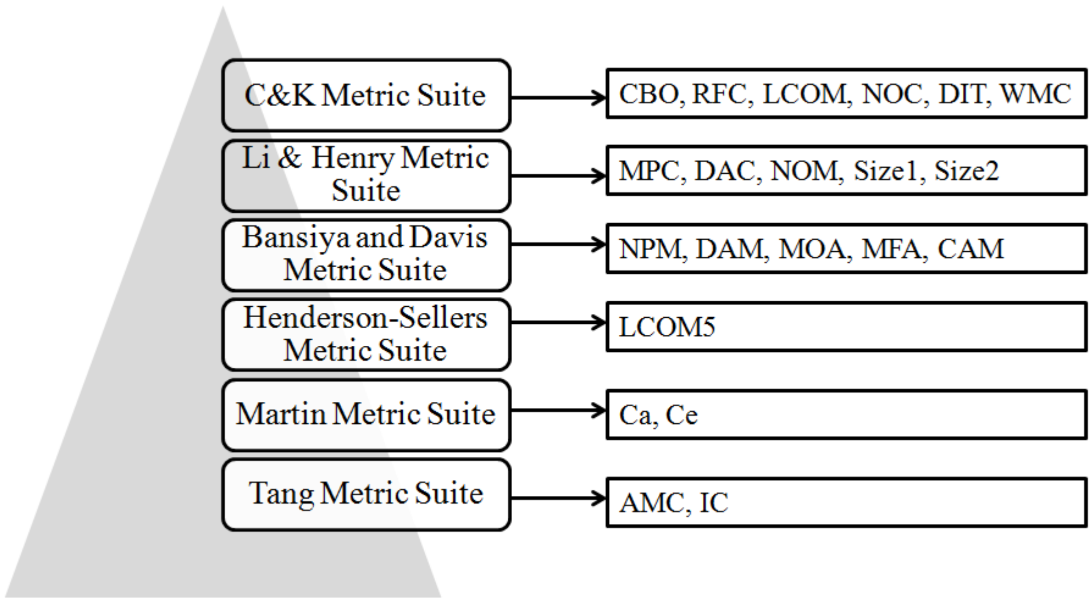
C&K Metric Suite
CBO, RFC, LCOM, NOC, DIT, WMC
Li & Henry Metric Suite
MPC, DAC, NOM, Size1, Size2
Bansiya and Davis Metric Suite
NPM, DAM, MOA, MFA, CAM
Henderson-Sellers Metric Suite
LCOM5
Martin Metric Suite
Ca, Ce
Tang Metric Suite
AMC, IC

Figure 2.2: Sources of Independent Variables

All the independent variables selected in the current research study and their sources are depicted in figure 2.2. One traditional metric LOC is also selected to keep the evolution of software in history perspective. Definitions of all the metrics are summarized as follows:

The Chidamber and Kemerer [43] metrics suite is given below:

1. **Coupling Between Objects**: CBO represents the number of classes to which the given class is coupled. Any two given classes are coupled if the methods declared in one

class uses either the methods or instance variables which are defined in another class. During the counting both types of relationship 'uses' and 'used-by' relationships are taken into account, however only once. Other types of references for example use of constants, calls to API declares, handling of events, user-defined types or the instantiation of the object are simply ignored. High CBO is undesirable as the excessive coupling between object classes is detrimental to modular design and prevents any reuse.

2. **Response For a Class**: RFC counts the number of local methods plus the number of non local methods called by local methods. As shown in equation (2.1), it's the sum of M and R where M is the number of methods in the class and R is the number of remote methods directly called by methods of the class.

$$RFC = M + R \tag{2.1}$$

3. **Lack of Cohesion of Methods**: LCOM counts the number of disjoint sets of local methods where each method in a disjoint set shares at least one instance variable with at least one member of the same set. Consider a class C_1 with n methods M_1, M_2. . . ., M_n. Let I_j is the set of all instance variables used by method M_i. There are n such sets $\{I_1\}$,. . . . . . .$\{I_n\}$. Let P= $\{(I_i, I_j) \mid I_i \cap I_j = 0\}$ and let Q= $\{(I_i, I_j) \mid I_i \cap I_j \neq 0\}$. Then the value of LCOM is |P| - |Q| if the value of |P| is greater than |Q|, otherwise it is considered as 0.

4. **Number of Children**: NOC counts a number of immediate sub classes of a class in a hierarchy.

5. **Depth of Inheritance**: DIT counts the depth of a class within the inheritance hierarchy. It is the maximum number of steps from the class node to the root of the tree and is measured by the number of ancestor classes.

6. **Weighted Methods per Class**: WMC represents the sum of McCabes's cyclomatic complexities of all local methods in a class. As shown in equation (2.2), let a class K_1 with method M_1... M_n, that are defined in the class. Then the WMC of the class is defined as the sum of all the WMC of each method.

$$WMC = \left(\sum_{i=1}^{n} Ci\right) \tag{2.2}$$

The Li and Henry [127] metric suite is given below:

7. **Message Passing Coupling**: MPC is computed as the The number of messages sent out from a class.

8. **Data Abstraction Coupling**: As the name implies, DAC counts the number of instances of another class declared within a class.

9. **Number of Methods**: NOM counts the number of the number of methods in a class.

10. **SIZE1**: As the name implies, it counts the number of lines of code excluding comments.

11. **SIZE2**: It counts the total count of the number of data attributes and the number of local methods in a class.

The Bansiya and Davis [15] metric suite is given below:

12. **Number of Public Methods**: NPM is computed by counting the number of public methods in a given class

13. **Data Access Metric**: DAM is calculated as the ratio of private + protected attributes of the said class to the total number of attributes defined in that class.

14. **Measure Of Aggression**: MOA counts the percentage of user defined data declared in the said class.

15. **Method of Functional Abstraction**: MFA is counted as the ratio between the inherited methods and the total number of methods in the said class.

16. **Cohesion Among the Methods of a Class**: CAMC is calculated based on the signatures of the methods. This metric computes the similarity among various method of the said class.

    The Henderson-sellers [89] metric suite is given below:

17. **Lack of Cohesion Among Methods of a Class**: LCOM5 metric is proposed by Henderson-sellers [89] to remove some of the disadvantages of LCOM and LCOM3. It was observed that if there are some variables in the class which are not accessed by any of it's methods, for example when the variables are accessed outside the class, LCOM consider it in the formula which is a major design flaw. In order to remove this discrepancy, the class variables should be encapsulated with accessor methods or properties as shown in equation (2.3). Further, it was also stated that its value should vary between 0 and 2 wherein value greater then 2 is considered as alarming.

$$LCOM3 = \frac{\left(\frac{1}{N}\sum_{i=1}^{n}\mu(D_i) - m\right)}{1-m} \tag{2.3}$$

    Where m represents the number of methods, N represents the number of instance variables (attributes) and μ represents the number of methods which access each datum.

    Martin [150] metric suite is given below:

18. **Afferent Couplings**: As the name implies, AC counts the incoming coupling by counting the number of classes in other packages that depend on classes within the package. So basically it is counted as the number of classes calling the said class.

19. **Efferent Couplings**: EC is counted as the number of other classes called by said class.

    The Tang et al. [216] metric suite is given below:

20. **Average Method Complexity**: AMC is computed as the average of McCabe Cyclomatic Complexity of all method.

21. **Inheritance Coupling**: As the name implies, IC counts the outgoing coupling by counting the number of parent classes to which a given class is coupled.

22. **Lines Of Code**: LOC counts the The number of lines in the source code excluding comments.

2.5.2 Dependent Variable

The dependent variable is CHANGE and it refers to the number of lines of code of a given class that were changed during the maintenance period. A single deletion or addition of a line in the source code of a class was counted as one change, however, any modification in one line of code was counted as two changes as it represented a deletion followed by an addition. Thus, the metric CHANGE, measures the maintainability attribute of the classes in these software systems. It is a continuous and non-negative integer.

One of the many difficult aspects of software maintenance phase is the estimation of maintenance effort needed to correct or enhance a software system. Organizations need to investigate which factors influence their maintenance or development process most so that they can do proper planning. Measuring and monitoring of maintenance efforts can be achieved by focusing on the measurements of coding related activities. Hence, in order to measure the maintenance efforts, we closely observe the maintenance history of the software and count the number of lines in which any kind of change is performed. As the change in any line of code may occur due to correction, prevention, perfection or adaptation activity, this thesis predict all kind of software maintainability using various machine learning and evolutionary techniques.

2.6 Empirical Data Collection

Data collection is a very important aspect to carrying out any empirical study in the field of software engineering. Inaccurate data collection not only put negative impact on the results, but such types of studies ultimately lead to inaccurate output.

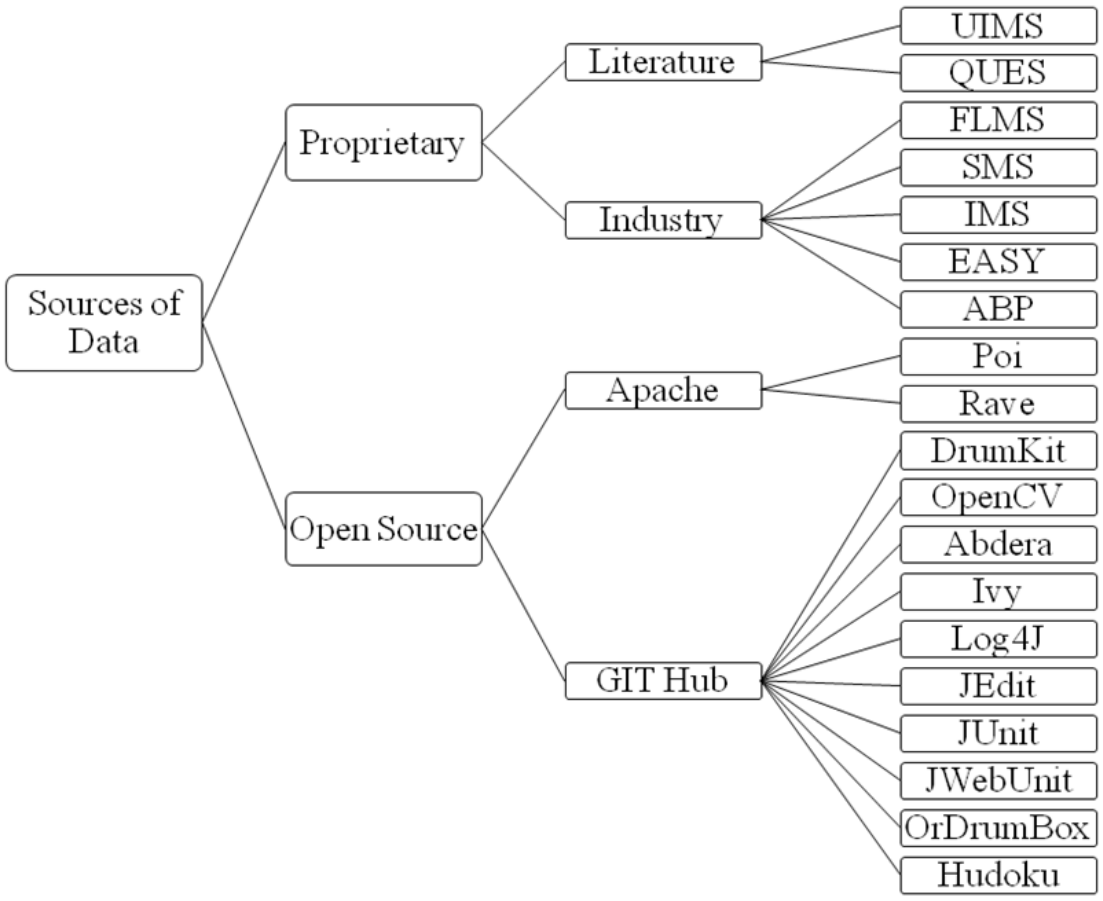

Figure 2.3: Sources of Empirical Data

The researchers in the field of software engineering are always constrained against non-availability of genuine datasets to conduct their validation studies and test newer prediction models. In one of the studies conducted Li and Henry [127], datasets of two proprietary software systems namely UIMS and QUES were made public. Many researchers used this datasets in order to validate their respective prediction technique such as Dagpinar and Jahnke [53], Elish and Elish [62], Kaur et al. [106], Koten and Gray [118] and Zhou and

Leung [237].

In this study, as shown in figure 2.3, data is collected from two sources, proprietary as well as open source to validate the maintainability prediction models. For the proprietary software datasets, apart from the UIMS and QUES dataset [127], five medium and large systems are used. Their details are given in the following subsection.

2.6.1 Proprietary dataset

In this category, we began our research by taking the proprietary dataset available in literature. We identified and decided to use one of the famous dataset given by Li and Henry [127] and also used by many researchers [53, 62, 106, 118, 137, 237]. In the next phase, dataset were collected from five proprietary software systems which are real life applications and maintained from the last three years. All software systems were large sized software and developed and maintained by professional in Microsoft Visual Studio .NET software using C# Language. The details of seven proprietary software systems used in the current research such as versions, release date, size, number of classes etc are summarized in the table 2.1. The process of collecting the data is explored in figure 2.4. However, brief description of their functioning is mentioned in following subsection.

2.6.1.1 User Interface Management System

UIMS is developed by Software Productivity Software Inc for creating an interface for user. Implemented in Classical Ada, the UIMS dataset contains class-level metrics data collected from 39 classes of a UIMS and metric values of each class was made public by Li and Henry [127].

2.6.1.2 Quality Evaluation System

QUES is also developed by Software Productivity Software Inc for evaluation of the quality. Implemented in Classical Ada, the QUES dataset contains class-level metrics data collected from 71 classes of a QUES system and metric values of each class was made public by Li and Henry [127].

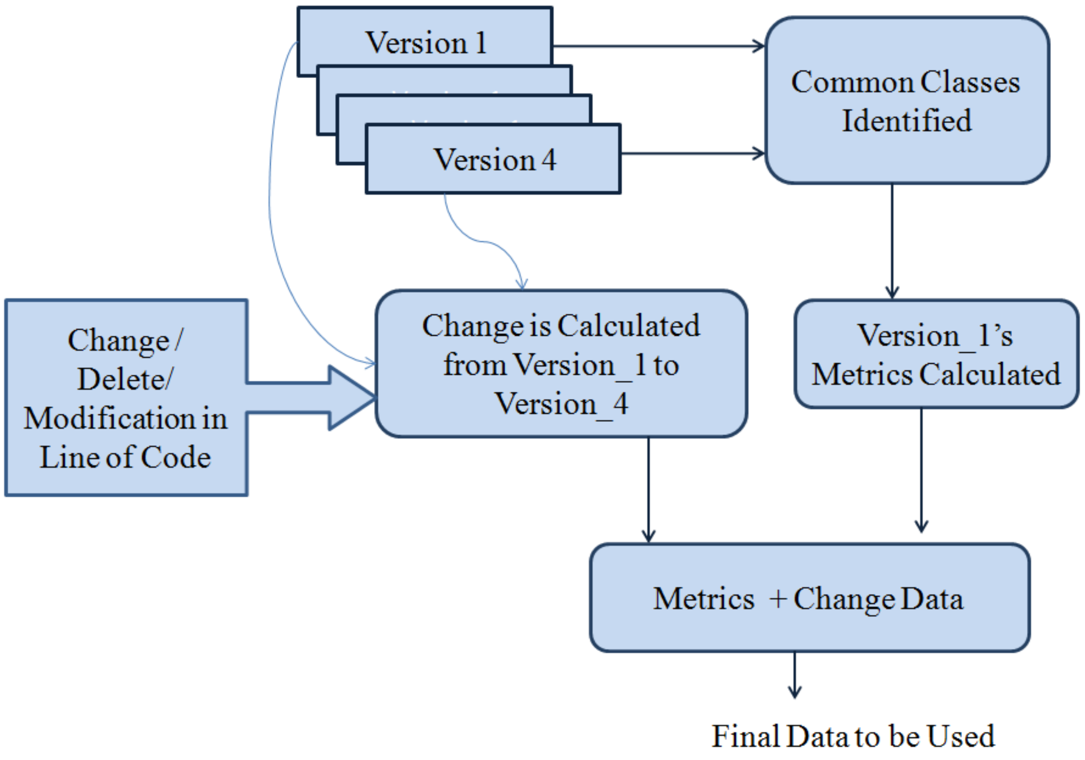
Version 1
Version 4
Common Classes Identified
Change / Delete/ Modification in Line of Code
Change is Calculated from Version_1 to Version_4
Version_1's Metrics Calculated
Metrics + Change Data
Final Data to be Used

Figure 2.4: Change Data Collection

Table 2.1: Characteristics of Proprietary Software

S. No.	Name of the Proprietary Software	Version	Release Date	Number of Classes	Percentage Change
1	User Interface System (UIMS)	1.0 to 2.0	01 Apr 2013	101 classes	20.34
2	Quality Evaluation System (QUES)	1.0 to 2.0	18 Jun 2012	143 classes	16.29
3	File Letter Monitoring (FLM) software	1.2 to 1.3	08 Feb 2013	686 classes	61.88
4	EASY software system	2.0 to 2.3	12 July 2012	614 classes	74.76
5	Student Management System (SMS) software	1.2 to 2.3	21 Feb 2011	351 classes	34.50
6	Inventory Management System(IMS) software	1.0 to 2.2	11 Feb 2012	417 classes	24.96
7	Angel Bill Printing (ABP) software	4.1 to 4.11	15 Nov 2011	251 classes	14.644

2.6.1.3 File Letter Monitoring Software

File Letter Monitoring Software (FLM) System is a customize software to track the movement of files between different department of an organization. It is a web application which monitors the file/letter movement in an office or organization. Files can be dispatched within office or other branch offices. The software maintains a log file to trace history of files movement. It also keeps a scanned copy of the letters which are kept in the database along with many other attributes such as dispatch number, dispatch date etc.

2.6.1.4 EASY Software system

EASY Classes Online Services is a web portal for an Educational Institute in Bawana, Delhi wherein study material is provided online for students in two ways. Firstly, users can freely register on the website as ‘Beginner’ and get a link to download free study material. In second way, users can register themselves as ‘Premium users’ after paying some fixed amount to the EASY Classes Ltd. They can also access online video tutorials, objective test paper, and subjective test paper regarding some specific topic. After solving the test paper, he can submit them, whose reports are generated to judge their performance in tests. Comparative performance of the students is also made available through visual charts for each topic.

2.6.1.5 Student Management Software

Student Management System (SMS) is a windows based application which maintains the record of students and teacher for private educational institute. In this software, there are two mode of receiving fee payment from the student i.e. installment and onetime payment. Salary is issued to the teachers as per their basic salary and other emoluments such as dearness allowance, house rent allowance, transport allowance etc by the administrator. Various reports can be generated such as teacher wise salary payments, student wise fee payments, institute’s balance sheet, running expenditures etc. and the print outs of these reports can also be taken as and when required.

2.6.1.6 Inventory Management Software

Inventory Management (IMS) software is a windows based application which maintains inventory of a company at different branch offices in geographically apart cities. It provides many services such as serial number tracking, bar code printing, tracking of expiration date for perishable items etc. It also provide procurement service which retrieves item costs for purchase orders and create vouchers for purchased goods. In addition to the solution for item identification within the Inventory Management system, general accounting features are also added such as payments, taxes, cash receipt processing revenue etc. Inventory management software uses a web-based interface to search, retrieve and display inventory data to the stock manager client.

2.6.1.7 Angel Bill Printing Software

Angel Bill Printing (ABP) software is a windows based compatible application for maintaining bills with backup and restores facility. It is software in which after adding information, it can maintain fully editable items list by client itself with generation of a common bill format. Invoices are delivered to the client after calculating, formatting, printing and e-mailing of bills. ABP software is very easy to operate and generate easy-to-read invoice templates which include facilities such as ability to show the amount in words, round off the total amount etc. Comprehensive reports and payment history are also available with every invoice in order to run the business efficiently.

2.6.2 Open Source dataset

Open source software are developed by different users located at geographically different places around the globe. It has altogether different approach and methodology for development. There are many advantages related with open source software such as they are freely available and does not involve copyright issues. Anyone can easily customized them as per their own specifications and use without paying any license fee.

In the second category, two open source repositories http://www.github.com and http://www.sourceforge.net are explored for collecting the empirical data from open source software. Two important characteristics were kept in mind for identifying the ideal open source system. Firstly, it should follow OO paradigm and secondly, it should have a high number of downloads in recent times (last 12 months) as it is a clear indication that there are active users contributing constantly.

Process of Empirical Data Collection

In order to collect the data for conducting the empirical studies, the procedure as shown in figure 2.5 is adopted in the current research. Initially the source code of initial version as well as modified version for the same software were collected. Git repository mining tool 'Defect Collection and Reporting System' (DCRS) developed in the Java language at Delhi Technological University by Malhotra et al. [147] was used for the purpose of empirical data collection. It processes the repositories and read the change descriptions such as time stamp of committing the incurred change, unique change identifier, type of change (defective, perfective or corrective), change descriptions etc. It pre-processes current version and previous version of the same software and extract only those classes which are common in both the versions.

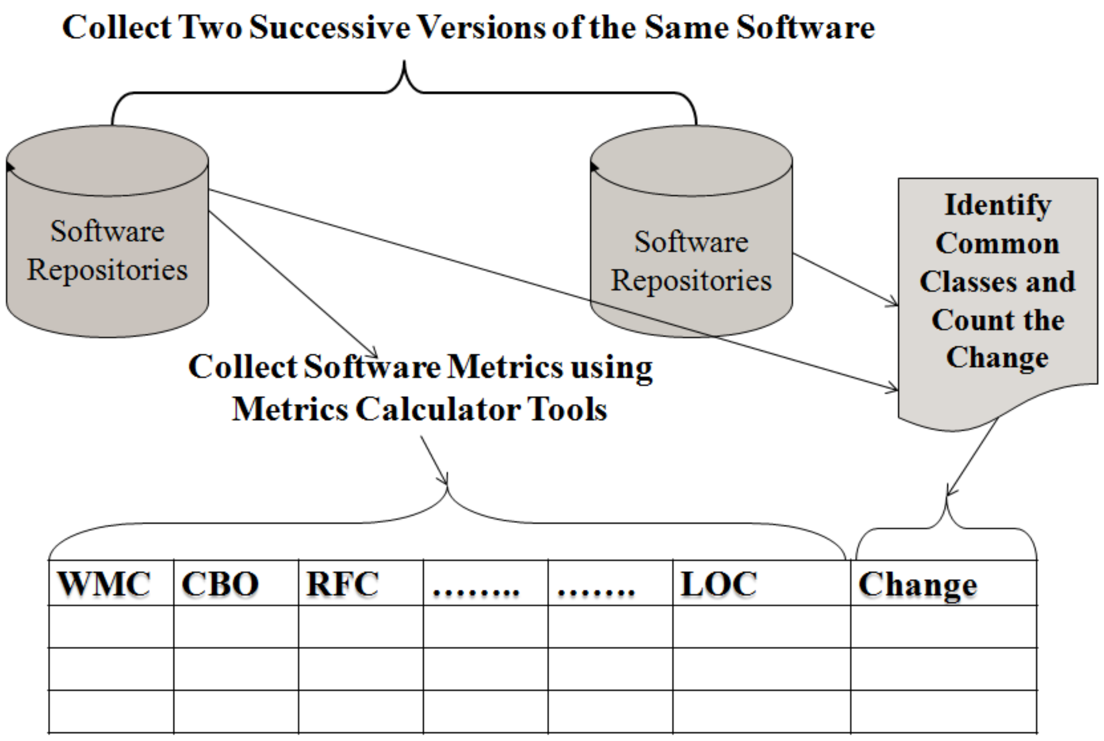
Collect Two Successive Versions of the Same Software
Software Repositories
Software Repositories
Identify Common Classes and Count the Change
Collect Software Metrics using Metrics Calculator Tools
WMC
CBO
RFC
……..
…….
LOC
Change

Figure 2.5: Process of Empirical Data Collection

Changes in the common classes were calculated in terms of the LOC. Any line added or deleted in the new version with respect to the old version is counted as one change whereas any lines of code modified in a new version with respect to the old version is counted as two changes. DCRS compares both the versions of the software and prepare the list of changes by counting the lines of code where changes took place and generate the reports containing detailed information for each class. It also calculates values of each of the OO metrics for each and every class and provides insight such as cloning of Git Repositories and Self Logging's. Similarly, processing the classes in older version generates the class wise values of each OO metric. Finally, data points for each class are generated where values of OO metric are considered as independent variables and the value of 'change' is considered as a dependent variable.

For example, if Abdera 1.1.2 (older version) and Abdera 1.1.3 (newer version) are being analyzed in DCRS, processing the change logs generates the number of changes performed from older version to newer version per class. There are 686 classes in this software; so the values of OO metrics for each class were collected for the older version i.e. Abdera 1.1.2. Finally, data points are combined for each class where OO metric values are considered as the independent variable and the value of change considered as a dependent variable.

Description of Open Source Datasets

Twelve open source datasets were identified and their source codes were collected in this study. All the details of these software systems such as versions, release date, size, the number of classes etc are summarized in table 2.2.

2.6.2.1 Drumkit

Drumkit is a Java Mobile based game on JAVA-JME platform (https://github.com/nokia-developer/drumkit-jme). It creates a virtual drum kit and allows the user to play drum by just tapping on the screen. It also gives facility to record the beats for future use and compatible with series 40 devices. Acoustic kit and pad kit views are available in latest updated version which gives multi touch support for series 40 full touch devices. Graphics drawing and audio playback are some of its important features which had made this software very popular. The application is open source which means anybody can download and make changes without license. One problem is yet open for the open source community developers that its graphics have yet not been optimized for Symbian devices.

2.6.2.2 OpenCV

OpenCV (Open Source Computer Vision) is a set of programmed modules in java with primary focus is to provide real time vision to electronic devices. Initially it was developed in Intel's research center for real-time ray tracing, but now its available at (http://sourceforge.net/projects/opencvlibrary) and free for use under the open-source BSD license. Library of the OpenCV is cross-platform for basic vision infrastructure. In the year 2000, very first version of OpenCV was released and since then it has undergone many changes such as C++ interface, safe patterns, better implementations etc. In the year 2012, support for OpenCV was taken over by a non-profit foundation OpenCV.org and now every six months, its update versions are released.

2.6.2.3 Abdera

Abdera is an open source atom parser generator used in client scripts as well as server scripts to build high quality designed documents on internet. Atom stands for web related standards and atom syndication format is an XML language used for web feeds. Adbera is developed on the standard of Atom Publishing Protocol (APP) which is a simple HTTP based protocol for web resources creation. Although its code was initially developed by IBM in the year 1998, later on it is donated to the Apache Software Foundation in the year 2006. Nowadays its code is available free for use under the open-source BSD license at https://git.apache.org/abdera for the open source developers community.

2.6.2.4 Ivy

Integrated with Apache Ant, Ivy is a set of open source libraries and programs that allow applications to broadcast information through text messages, with a subscription mechanism based on regular expressions. Its very simple to use and provides powerful OO Java dependency management. It is considered as most popular Java build management system based on Apache design principles. It generates two primary report types i.e. HTML reports and graph reports to help in understanding immediate dependencies, transitive dependencies and conflicts of the projects. It is also free for use under the open-source BSD license and available at https://git.apache.org/ivy.

2.6.2.5 Log4j

It quite common practice to insert log statement while debugging any software because debuggers are not always available or applicable. It is even more significant in the case of multi threaded and distributed applications. However, it has a drawback that it can slow down the running of an application. Log4J is a software specially designed to address these concerns of the debugging process. It is reliable, fast, extensible and simple to understand and use. With arbitrary granularity, it allows the developer to control which log statements are output and when. It has three main components i.e. loggers, appenders and layouts which work together to enable developers to log messages according to message type and level, and to control at run time reports. It is fully configurable at run time using external configuration files. It is also free for use under the open-source BSD license and available at https://git.apache.org/Log4j

2.6.2.6 JEdit

JEdit is a highly customizable text editor written in Java and runs on any operating system. Originally developed by Slava Pestov, it is donated to open source community in the year 2006. It can be extended with macros written in scripting languages. Apart from simple facilities such as code folding, text folding, text wrapping etc., it also supports advance facilities such as spell checker, auto-complete, etc. using more than 150 patches available free online at https://jedit.svn.sourceforge.net/svnroot/jedit

Table 2.2: Characteristics of Open Source Software

S.No.	Software	Version	Release Date	Number of Classes	Percentage change
1.	Drumkit	1-0.5.0 to 1-0.6.0	25 Apr 2014	101 classes	20.34
2.	OpenCV	2.4.10 to 3.0	12 Sept 2014	143 Classes	16.29
3.	Abdera	1.1.2 to 1.1.3	08 Jan 2014	686 Classes	61.88
4.	Ivy	2.2.0 to 2.3.0	21 June 2013	614 Classes	74.76
5.	Log4j	1.2.16 to 1.2.17	31 Mar 2010	351 classes	34.50
6.	JEdit	5.1 to 5.2	28 July 2013	417 classes	24.96
7.	JUnit	4.10 to 4.11	29 Sept 2011	251 classes	14.644
8.	HuDoKu	2.0 to 2.2	01 Aug 2012	45 classes	11.63
9.	JWebUnit	1.2 to 3.0	08 Oct 2012	230 classes	24.37
10.	OrDrumBox	0.6.5 to 0.9.8	07 Jan 2012	218 classes	24.96
11.	Apache Poi	3.9 to 3.10	01 Feb 2014	940 classes	16.44
12.	Apache Rave	0.21.1 to 0.22	09 July 2013	672 classes	33.29

2.6.2.7 JUnit

JUnit is an open source framework for writing and executing unit tests and defining test suite. As JUint is compatible with almost all IDEs and it also has inbuilt test drivers, use only need to write the test case. It is one of the most popular unit testing tools used to run and tests the scripts based on the defined annotations. It also allows testing the specific module and prove quick results very efficiently. The test scripts can be written in short span of time due to its simplicity. It also provides the facility of test case prioritization on the basis of user's requirements. It is also free for use under the open-source BSD license and available at http://sourceforge.net/projects/junit/.

2.6.2.8 OrDrumBox

OrDrumBox is a open source software which has audio sequencer function written in java to facilitates creation of online songs. There are many features provided for creating song online such as audio mixer, piano roll, rendering, automatic matching for sound and track, composing of song automatically, allows soft synthesizers etc. Midi files and wav files can be imported and exported through this software which makes it more compatible with existing music files. Powerful graphic user interface allows the writing of pattern on blank working space right from scratch, add several new tracks, addition of a new note for creating the main beat and auto generation of the patterns. As many sounds as required can be created by changing the tempo and volume before playing the song. The sequencer allows assembling the patterns in a song and configure the program to repeat each pattern several times. It is also free for use under the open-source softonic license and available at http://www.freewarefiles.com/OrDrumBox_program_105766.html

2.6.2.9 HuDoKu

HoDoKu is open source software based on java framework to generate, solve, train and analyze the sudoku game in multiple languages. It allows five difficulty levels which either can be solved by the user or he can ask the software to demonstrate solving procedure. Many humans style of solving techniques such as fish, wings, coloring, chains, LCs, uniqueness and subsets are supported in this software. It a great tool for users those who wants to learn the solving procedure of sudoku. It is such a powerful tool that allows the user to automatically set a point and solve the sudoku till that point only. It also has the facility of analysis where the user can view all available solution options at a particular point and select one. New version allows to create a save point so that if some guess work is there, user can revert to the save point in case the guess proves to be wrong in later on stage. It is also free for use under the open-source software licensed under the GPLv3 and available at http://sourceforge.net/projects/hudoku/.

2.6.2.10 JWebUnit

JWebUnit is an open source software developed in Java framework to facilitate the web application testing. It evolved from JUnit open source software in which various refactoring treatments are provided followed by the code modification using several test engines to make it more suitable for creating acceptance test cases for a given web applications. Application Programming Interface (API) are also provided to navigate the site under testing. Finally, set of assertions are asked to verify the correctness of application. It is also free for use under the open-source BSD license and available at http://sourceforge.net/projects/jwebunit/.

2.6.2.11 Apache Rave

Apache Rave is the web based data integration application software. It's a light weight java platform to host the widgets to manage open social gadgets. It combines the data, presentations or functionality from two or more sources to create new services, for example Flicker. It's actually not a portal but a mash up which is customizable and support various platforms. Apache Rave is available at https://rave.apache.org/.

2.6.2.12 Apache Poi

Apache Poi stands for 'Poor Obfuscation Implementation'. As the name suggests whenever the communication is confusing or harder to interpret, this free and open source java library is used for reading and writing Microsoft document formats. Written in java language, it can be used for creating Microsoft Office file formats such as Excel, PowerPoint, MS Word etc. It provides excellent support for Microsoft Excel and able to handle all formats. Apache Poi available online at http://poi.apache.org/Visio.

2.7 Data Analysis and Pre-Processing

2.7.1 Descriptive Statistics

Descriptive statistics are used as a tool for analyzing the research data in order to identify whether the data is skewed or normalized. Numerical facts are presented in tables as well as in graph form for the purpose of description and decision making. If the data is simply presented in raw form, it is really difficult to visualize what the data is conveying and the task becomes much more difficult especially when the data is large. Graphical presentations of descriptive statistics are very helpful to effectively communicate a message which is easily understood by almost everyone. Data is reduced to a state where it can be easily used for further analysis.

Five main descriptive statistics measures used in this research are maximum, minimum, mean, median, and standard deviation with respect to each metric. Their values are further used for the interpretation, for example if the mean value of the attribute used for measuring the coupling is less than three, then we interpret that low coupling is imparted in the software. These five measures are also used to compare the characteristics and skewness of different datasets used in our empirical studies. For example if software A has mean value of NOC less than the mean NOC value of software B, we interpret it that inheritance is comparatively not much used in systems A. Similarly if the maximum value of the LCOM is greater in software A than in software B, then its interpreted as cohesion is high in software A.

2.8 Feature Subset Selection

One of the main problems while making prediction model is to identify a representative set of features from which the classification model can be constructed for prediction task. In order to address the problem of feature sub selection, many techniques are suggested in literature with underlying goal to identify those features out of all given attributes which are highly correlated with the dependent variable, yet uncorrelated with each other [135]. The advantages of FSS are:

- Results are more accurate, compact and quick.
- Since the dataset is reduced, measurement cost is also reduced as less data needs to be collected.
- Understandability of the model increases.
- Time used to train the model decreases because of fewer data.
- Due to the simplification of models, it becomes much easier to interpret by researchers.
- Enhanced generalization could be achieved because data contains more of quality and less of quantity.

- It reduces the problem of over-fitting in to the data during the training process.
- Reduction of the variance due to presence of only core data.

Hence by reducing the dimensions of the data in the first stage, machine learning techniques are able to operate faster and more effectively.

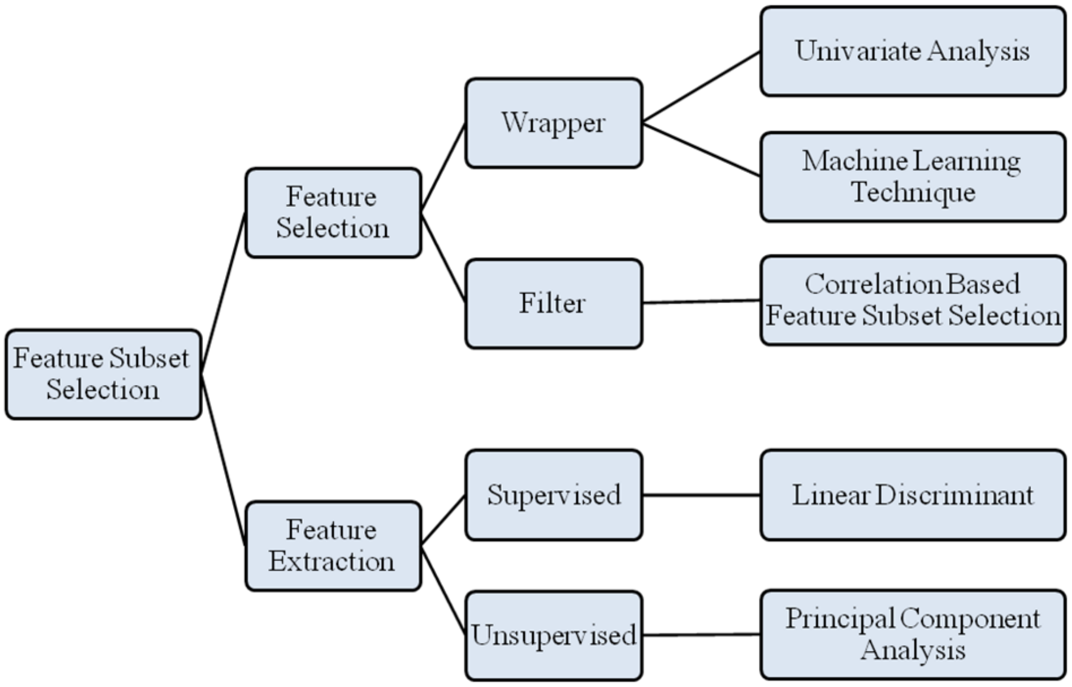

Figure 2.6: Classification of Features Subset Selection Methods

FSS can be achieved in two ways as shown in figure 2.6, first method is known as "attribute selection" which eliminate those independent variables that have little or no information in them. Second method is known as "attribute extraction" in which several attributes are combined into new set of attributes. Both the methods are further subdivided into subcategories.

2.8.1 Attribute Selection

In one of the study conducted by Kohavi and John [115], feature selection further categorized into two types i.e. Wrappers and Filters. While evaluating the most valued subset from a given set of attributes, filter method does not require the classification algorithms whereas wrappers method needs them. The wrapper feature subset evaluation conducts a search for a good subset using the learning algorithm itself as part of the evaluation function.

2.8.1.1 Univariate Linear Regression

The univariate analysis is the process in which individual effect of each independent variable on dependent variable is identified. The choice of methods in the univariate analysis depends on the type of dependent variables being used. Since in our research study the dependent variable is continuous, univariate analysis using linear regression is performed to find the individual effect of each of the OO metric described in section 2.5.1 on dependent variable change.

Performing the univariate analysis using linear regression helped us in two ways in this research study. Firstly, it reduces the dimensions of the data because the OO metrics which are not related to dependent variable change can be screen out based on the characteristics of dataset. Secondly, as in our study we have proposed new metrics, so univariate analysis helped us to identify the individual effects of newly proposed metrics on dependent variable change based on the characteristics of the dataset. Four values are received in the output of univariate analysis i.e. estimated coefficient, standard error, the t-ratio and p-value. The value of Sig (p-value) represents amount of significance of the each of the independent metrics on change. If in the outcome p-value is received as 0.000, that means both independent and dependent variables are significantly correlated.

2.8.1.2 Correlation Based Feature Subset Selection

Filter methods are faster than wrapper method, however, some times it may fail because the selected attributes are not tuned for particular prediction mode. Correlation based Feature Subset-Selection (CFS) is widely used filter method. CFS calculates the worth of a subset of attributes on the basis of individual predictive capability of every feature and the degree of redundancy among them [82]. While using this algorithm, subsets of features that are highly correlated with the class but have low inter-correlation are chosen. The technique is based on a heuristic algorithm to evaluate a subset of attributes by balancing them after identifying the redundancy present in them.

2.8.2 Attribute Extraction

No attributes is removed in this method, instead a new attribute is produces by assigning weights to the original attributes as per their impact and combining them. Two types of attribute extraction methods are used, supervised in which output variable is known and unsupervised in which output variables are not known. Principal Component Analysis (PCA) is most widely used unsupervised attribute extraction technique explained as under.

2.8.2.1 Principal Component Analysis

In PCA technique, principal components are created which are linearly uncorrelated variables. These variables are created from the set of independent variables which have a little correlation among themselves. First of all in this technique covariance matrix of the original variables is calculated then it is followed by the calculation of Eigen vectors. The principal components have the important property that final information content is represented using few best features so that remainder attributes can be discarded.

2.8.2.2 Linear Discriminant Analysis

The objective of Linear Discriminant Analysis (LDA) is to perform dimensionality reduction while preserving as much of the class discriminatory information as possible [99]. LDA is an instance based ranking technique. This technique estimates the quality of attributes according to the way they differentiate between instances of different classes that are very close to each other. For example, it picks up randomly any instance and search for the nearest neighbor from the same class and call it hit and nearest neighbor from the different class and call it ‘miss’. Depending upon the values of each attribute, the relevance score is updated and the process is repeated for a specific number of times. Finally, if two or more attributes having high relevance among them, then only one is selected.

2.8.3 Discussion

PCA is a widely used method for attribute sub-selection when used with neural networks whereas univariate analysis is used for pre-selecting the important metrics as per their significance [82]. CFS is found to be fastest technique when it is used with machine learning techniques [135]. In this research study, relatively new technique, GA are used for FSS. As suggested by Yang and Honavar [231], it identifies and screen out noisy, irrelevant and redundant features, and identifies relevant features that does not strongly depend on other features. On natural domains, GA typically eliminated half of the features and permits its scaling to larger datasets.

2.9 Criteria for the Selection of Prediction Modeling Techniques

While analyzing the quality of the software, various design metrics must be used together to measure all of its aspects because single metric alone cannot reveal its various characteristics [135]. In the field of software engineering, researchers have already witnessed the application of machine learning models for predicting external qualities such as software development cost, testing efforts, maintenance efforts, correction cost, software defects, software reliability etc. using internal quality attributes such as coupling, cohesion, inheritance, polymorphism, abstraction etc. Many machine learning techniques have been explored such as decision tree, ANN, BBN, SVM, genetic programming etc. in the empirical studies conducted by various researchers throughout the globe [135]. In the next sections all the prediction modeling techniques are described in brief. Apart from understandability, scalability and simplicity, the most desirable factor is the accuracy of prediction model. The cost, effort and time needed to train any machine learning model must always be as less as possible. While the results of the comprehensible prediction model should be easily interpretable, it should manage the tradeoff between variance and biasness and should try to minimize both of them.

There are several models and metrics proposed in the literature to predict the maintainability of the software systems. These methods vary from simple statistical models such as regression analysis to complex machine learning technique such as neural networks etc. The most important aspect while creating any prediction model is the selection of particular technique to be used for training. After discussing the desirable basic characteristics of every prediction model, next obvious question is picking up of a particular machine learning technique, which depends on many factors such as data type, the availability of time, type of dependent variable, the number of independent variables, the size of the dataset etc. Figure 2.7 summarizes various factors that must be taken into consideration for selecting the particular machine learning techniques.

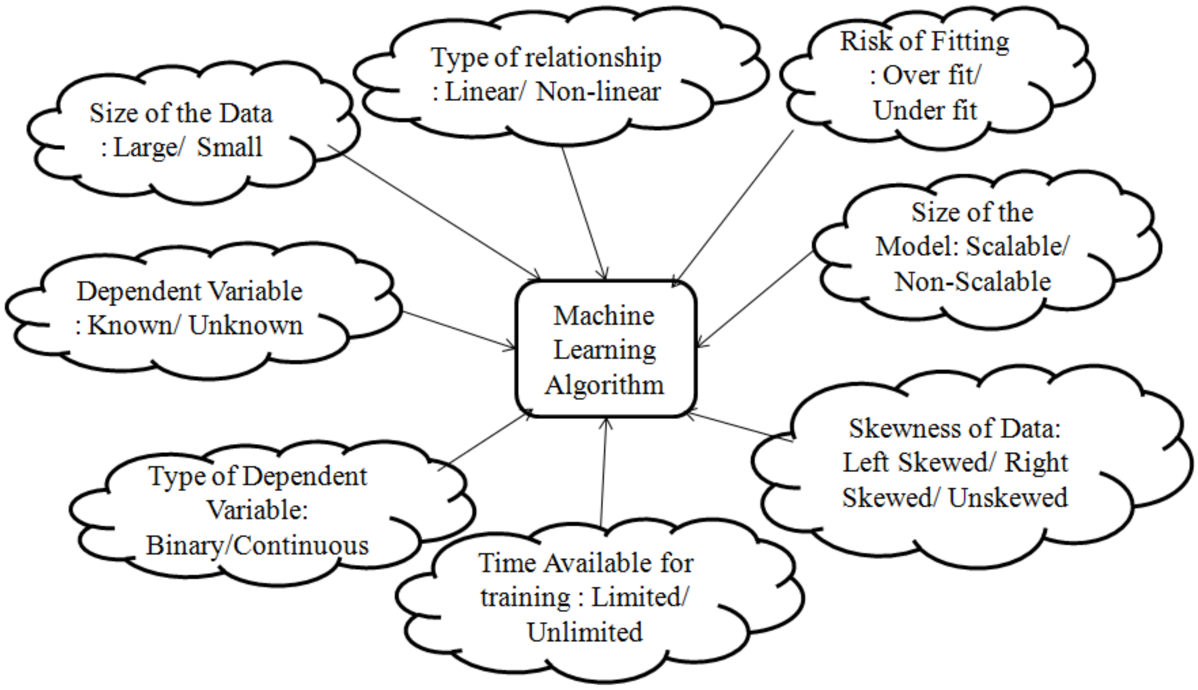
Type of relationship : Linear/ Non-linear
Risk of Fitting : Over fit/ Under fit
Size of the Data : Large/ Small
Size of the Model: Scalable/ Non-Scalable
Dependent Variable : Known/ Unknown
Machine Learning Algorithm
Skewness of Data: Left Skewed/ Right Skewed/ Unskewed
Type of Dependent Variable: Binary/Continuous
Time Available for training : Limited/ Unlimited

Figure 2.7: Factors Affecting Selection of Prediction Techniques

Logistic Regression should be used if the dependent variable is binary and the training dataset is large because it is a scalable prediction model. Naive Bayes is found to be a good choice when the dependent variable is multi-category whereas training dataset is finite. Both

Naive Byes and Logistic regression gives a probabilistic interpretation to the output. If the relationship between independent and dependent variables is linear, a decision tree is the best choice because they are less prone to outliers, easy to understand and non-parametric in nature. The only disadvantage with decision tree is that sometimes it over-fit the data; however, this type of problem does not occur with ensemble learners and random forest method. All the three techniques i.e. decision tree, random tree and ensemble learners are scalable and easy to learn. The kernel of SVM gives a theoretical guarantee that the model would not over-fit the data. Hence, if accuracy is the top most priority, memory intensive SVM should be selected. The only disadvantage with SVM is that they are not scalable. Neural Networks does not perform statistical training, hence, they are good for the complex non-linear relationship between independent and dependent variables. Much complex computational burden and proneness to over-fitting are the disadvantages of neural networks.

It is also observed that if the numbers of independent variables are more than six, and the relationship between independent variable with dependent variable is complex and non-linear, then GMDH technique should be selected. It is so because GMDH technique breaks the network into sub-network and after every training iteration, it sorts all the sub-network on the basis of an error in decreasing order and removes those sub-network units which have a large error with respect to the accuracy measure. In our research, since the dataset was large (mostly all datasets have more than 600 classes), relationship is very complex as well as non-linear and the number of independent variables are more than six (Twenty one independent variables considered in this research as discussed in section 2.5.1), we have successfully explored the applications of GMDH technique.

2.10 Machine Learning Techniques

In this section, we explain the various machine learning techniques used for making the prediction models as well as to ascertain the relationship of design metrics with maintainability.

2.10.1 Linear Regression

The linear regression model is so old that it exists even in the pre-computer age of statistics. They are still significant in this computer era as they are the foundation of most of the advanced methods. If the training dataset is small, it is found that they even out-performed other non-linear models used for predictive analysis. Linear regression uses statistical method to estimate the response by assuming that the regression function is linear with the given inputs $X_1 + X_2 + X_3$....... X_n. In one of the research study conducted by Hoffmann and Shafer [93], it was found that linear regression model has low signal-to-noise ratio because first it determines the percent of variance occurring in the dependent variable due to each independent variable separately and then, later on, uses this knowledge to predict the dependent variable.

2.10.2 Multivariate Analysis

In multivariate analysis method, weighted linear combination of each of the independent variables is identified in such a way that they should be able to predict the dependent variable with minimum possible error. It can be performed using many techniques depending on the type of dependent variable; for example linear regression is used when the dependent variable is continuous and logistic regression is used when the dependent variable is categorical. MLR is the most commonly used technique for fitting a linear equation on observed data. Further in multivariate linear regression, there are three methods used for identifying and picking the subset of important metrics from the set of independent variables i.e. forward selection, backward selection and stepwise selection. In this study, stepwise selection method is used as it guarantees to provide optimum and most significant subset of independent variables. Multivariate analysis assign the weight to each of the independent variable in such a way that the final predicted value should be as much close to the actual value as possible.

2.10.3 M5 Rules

M5 Rules technique is capable of predicting linear model which nearly maps each example to a different value. This technique uses rules to explain the data, hence, instead of single output value in the node, it has a collection of rules in the each node. M5 Rules technique is considered as one of the most flexible technique because it is able to map the examples covered by one rule on many different outcome values which is not the case without classifiers having a single target value. In one of the research study by Kohavi and Sommerfield [116], accurate and compact rule sets are generated using separate-and-conquer paradigm. They concluded that M5 Rules based classifiers always have a much lower number of rules and conditions than other techniques such as linear regression because of its capability to balance the trade-off between generality versus consistency of the attribute.

2.10.4 Bayesian Belief Networks

BBN have gained popularity only in the last decade in number of applications such as bioinformatics, text mining, natural language processing, signal processing, speech recognition, error-control codes etc [118]. It belongs to the family of probabilistic graphical models and often used to represent knowledge about an uncertain domain. BBN are both mathematically rigorous and intuitively understandable. They are represented using a graphical structure known as directed acyclic graph. Mathematically the network is defined by a pair B = (G,E), where G consists of nodes X1, X2, X3......Xn in the graph to represent random variables and E represents the edges between the nodes to depict the direct dependencies between these variables [118].

2.10.5 Decision Tree

Decision Trees classifier is one of the most popular technique used in various disciplines such as pattern recognition, data mining, machine learning etc. Various splitting criteria and pruning techniques are proposed and validated by many researchers [116]. A decision tree is created based on the concept of entropy and information gain. During the construction as per the pre-specified splitting criterion, the most qualified independent variable is selected at the node which consists of nodes that form a rooted tree, originate from a root which has no incoming edges. It is a classifier expressed as a recursive partition of the instance space. Other than the root node, each node has exactly one incoming edge. All those nodes, which have exactly one outgoing edge is called internal nodes and those which does not have out going edges are called external nodes. Each internal node behaves as a splitting node and splits the instance space into two or more sub-space depending upon the value of that attribute. Each leaf node represents one class with the most appropriate target value. Instances are classified by navigating them from the root of the tree down to a leaf, according to the outcome of the tests along the path. Each node is labeled with the attribute it tests, and its branches are labeled with its corresponding values.

2.10.6 Support Vector Machine

Originally developed for solving the binary classification problems, SVM are explicitly based on a theoretical model of learning which was later extended to solve the regression problems. In this method, a hyper-plane is created to separate the data into the nonlinear region and finally with the help of kernel function data points are mapped into different dimensional space. This technique was first introduced in 1992, but very soon it became very popular among researchers because of its success in handwritten digit recognition. It allows the user to separately implement and design their components because of its modular design and provide theoretical guarantees about its best performance.

SVM is a quadratic optimization problem subject to linear constraints with the unique minimum. In one of the research study conducted by Cortes and Vapnik [51], difficult task to detect and exploit complex patterns in datasets are explored using SVM technique. In the category of the supervised learning techniques, they are found to be the best in terms of accuracy and efficiency. SVM maximizes the margin around the separating hyper-plane which do not get affected by local minima, hence, its solutions are unique especially when they are global. Support vectors are actually nothing but the data points that lie closest to the decision surface which works as decision function to be fully specified by a subset of training samples.

2.10.7 K Star

K Star is an instance based learner which uses entropy as a distance measure to improve the accuracy in every iteration. It provides a consistent approach for handling the independent attributes which are either having real values, or the symbolic values. Proposed by Lee and Song [122], it uses similarity function from the training set to classify the test set. Attributes having missing values are filled very carefully in order to attain the maximum accuracy. They are treated as if they were drawn at random from all instances in the dataset. Missing values are taken as the mean of the probability of transforming to each of the attribute values which are not missing in the dataset. Finally, by taking the average of the column entropy curves, the missing value is determined.

2.10.8 Ensemble Learning

Over the years, ensemble learning algorithms have become extremely popular because, even though it generates multiple base models using traditional machine learning techniques, but finally it combines them into single ensemble model by taking their aggregate. Since most ensemble algorithms operate in batch mode, and repeatedly read and process the entire training set again and again, they are always considered better than single classifier. In this technique, first, a set of classifiers are constructed and in the very next step, final classification is performed by taking a weighted vote of their predictions. Originally only Bayesian averaging method was developed under this category, but later on, bagging, boosting methods were also developed and became very popular because of their good empirical results and theoretical foundations [33]. They are described as follows:

2.10.8.1 Boosting

Boosting is considered as a general method for improving the accuracy of any given learning algorithm by Freund et al. [71]. It does not suffer from over fitting. There are basically two types of boosting techniques, Logitboost and Adaboost. Logitboost is one of the popular releases of boosting which uses regression method as the base learner and it performs additive logistic regression. Adaboost combines a number of weak hypotheses to get better classification performance. For this, equal weights are assigned to all the training examples and then the weights of the incorrectly classified examples are increased on each round so that a weak learner is forced to focus on the hard examples in the training set [71].

2.10.8.2 Bagging

Developed by Leo Breiman [33] to increase the accuracy of regression models, bagging reduces the variance and helps to avoid the problems associated with over-fitting. The idea is to build various similar training sets and train a new function for each of them. In this study, meta learning based bagging method is used to predict the number of changes which might occur in a class based on OO metrics. The bagging method under the category of ensemble learners are well established and recognized as the best method for obtaining highly accurate classifications.

2.10.8.3 Non Linear Boosting Projection

Schapire and Singer [182] identified two scenarios where there are chances that boosting may fail, first when there is insufficient training data relative to the complexity of the base classifiers and second when the training errors of the base classifiers become too large too quickly. Poposed by Garcia et al. [74], in Non Linear Boosting Projection (NLBP) approach, instead of random space, constructive non linear projections are created using neural networks and further combined with the philosophy of boosting to handle noise present in the data.

2.10.8.4 Discussion

Ensemble learning is the process of combining multiple classifiers to solve a computational problem. When the data available is too large for a single classifier to be trained, we can partition the data into subsets and allow different classifiers to be trained on each subset and combine the results using some specified rules. On the other hand, when the data is too small, we can use bootstrapping mechanism, wherein we draw the data with replacement and apply a classifier to each sample. Among number of classifiers available, it is difficult to make a choice of the appropriate classifier for our problem. Combining multiple classifiers helps to reduce the chances of making a poor or wrong selection. It may or may not improve the performance over a single classifier, but it certainly reduces the risk of poor selection, hence, in this research, we have used NLBP ensemble learners.

2.10.9 Artificial Neural Network

Inspired from the natural functioning of the brain, ANN is the electronic models which are developed to provide machine solutions in non technical manner. They are originally developed to mimic basic biological neural systems particularly the neurons present in the human brain. Although computer capable of doing rote things very speedily such as solving the large complex mathematical equations, however doing the easy job such as recognizing even simple patterns are very troublesome for computers as shown in figure 2.8.

ANN deals with the process of storing information as patterns and utilizing those patterns for solving problems based on past learning. ANN does not involve traditional programming practices, instead, it involves the creation of massively parallel networks and the training of those networks to solve specific problems. This field also utilizes words very different from traditional computing like behave, react, self-organize, learn, forget and generalize.

Advantages of ANN: One of the important advantages of ANN is that it can learn by observing the datasets using approximation method. These types of random function approximation methods are very cost effective. ANN saves both time and money by considering data samples rather than entire datasets to arrive at solutions. They are simple mathematical models which enhance the existing data analysis technologies. Many different versions of ANN model have been used in this research study as mentioned below.

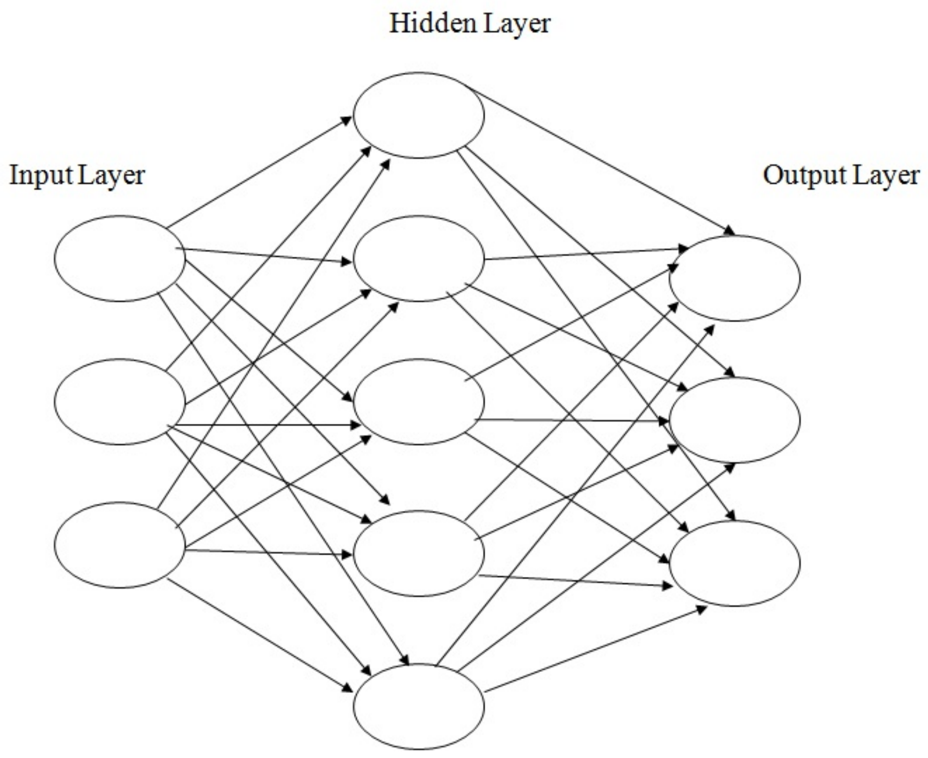

Figure 2.8: Architecture of Artificial Neural Network

2.10.9.1 Back Propagation Network

Although Back Propagation Network (BPN) is originally invented by Hu [94] in 1964, however it came into use only in 1994 by Rumelhart et al. [188] when it was used as supervised learning technique. Training data in BPN consists of a pair of the vector (input vector and target vector). During the training process, an input vector is presented to the network for the learning process. Output vector is generated from these learning vectors and compared with the actual target value. If there is any difference in the values, the weights of the network are readjusted to reduce this error and the process is repeated until the desired output is produced.

2.10.9.2 Kohonen Network

Proposed by Kohonen [117], Kohonen Network is best known as self organizing networks as they learn to create maps of the input space in a self-organizing way. Although, Kohonen Network is invented to provide a way of representing multidimensional data in much lower dimensional spaces, a network is created that learn the information such that any topological relationships within the training set are maintained without supervision.

2.10.9.3 Feed Forward Neural Network

In Feed Forward Neural Network (FFNN) [38], information moves in only one direction i.e. forward from input nodes to output nodes through hidden nodes and there are no loops in the network. The number of hidden neuron selected as 10 for the sample data collected from these five real life applications.

2.10.9.4 General Regression Neural Networks

Proposed by Specht and Shapiro [207], GRNN is a very powerful network as it needs only a fraction of the training samples during the learning process and finishes the learning process in a single pass. Due to the highly parallel structure, it performs well even in the case of noisy and sparse data and the over fitting problem does not arise as neither do they set the training parameters during the commencement of learning process, nor they define the momentum. Once network finished the training process, the only smoothing factor is applied to determine how tightly the network matches its prediction.

2.10.9.5 Probabilistic Neural Networks

PNN is a special type of FFNN created by Specht [208] which is based on BBN and Kernel Fisher discriminate analysis. In PNN, the operations are organized into a multi-layered feed forward network with four layers namely Input layer, Hidden layer, Pattern layer/Summation layer and an Output layer. The first layer is input layer where one neuron is present for each independent variable. The next layer is the hidden layer which contains one neuron for each set of training data. It not only stores the values of the each predictor variables but also stores each neuron along with its target value. Next is the Pattern layer, one pattern neuron is present for each category of the output variable. The last layer is output layer wherein weighted votes for each target category is compared and selected. Unlike BPN, which require feedback of errors and subsequent adjustment of weights and many presentations of training patterns, training a PNN network is very fast because it requires that each pattern is presented to the network only once during training. During the training session, we can see the number of learning events completed during training which is also called as ‘epoch’. Training can be done in real time since training is almost instantaneous. When data is sparse, training is superior to other network types. The success of PNN networks is dependent upon the smoothing factor. The adaptive PNN network is very powerful as during the building of neural networks, it uses genetic techniques.

2.10.9.6 Jordan Elman Recurrent Network

Jordan Elman Recurrent Network (JERN) is special kind of recurrent network in which hidden layers are fed directly into the input layers [122]. Even though JERN are slower but this type of recurrent network has the ability to learn the sequences. JERN network combine the past values of the context unit with the present input to obtain the present net output. In JERN, context unit acts as low pass filter and creates an output by giving average value to some of its most recent past outputs. Output of the network is obtained by summing the past values multiplied by the scalar parameter. The input to the context unit is copied from the network layer, but the outputs of the context unit are incorporated in the net through their adaptive weights. JERN is a very powerful learning as the hidden layer is fed back into the input layer, so features detected in all previous patterns are fed into the network with each new pattern.

2.10.10 Genetic Algorithms

GA is an adaptive system motivated by biological system proposed in Charles Darwin’s evolution theory. It is a high level simulation. The GA starts with a set of solutions (represented by chromosomes) called population. Genetic technique is a search heuristic and it mimics the process of natural evolution. This heuristic is routinely used to generate useful solutions to optimization and search problems. Best solutions from one population are then taken and used to form a new population which will be better than the old one. While choosing the solutions, their fitness function is evaluated. Those solutions which are more close to fitness function have more probability to be selected. We say that the more suitable solutions have more chances 'to survive'. This process is repeated until some condition is satisfied such as achievement of the best solution. Hence the population is improved over generations to accomplish the best solution.

2.10.11 GRNN with Genetic Adaptive Learning

GRNN with Genetic Adaptive Learning (GGAL) is a hybrid technique in which the GRNN technique mentioned in previous section is modified by introducing the genetic adaptive learning during the training session. As suggested by Specht et al. [207], by simulating the biological evolution, this genetic inspired neural network method has the ability to search large and complex spaces to determine near optimal solutions in time and space efficient manner.

2.10.12 Group Method of Data Handling

Russian Scientists Ivakhnenko and Koppa [96] introduced this technique in the year 1970 for constructing an extremely high order regression type model termed as GMDH. This technique builds a multinomial of a degree in hundreds, whereas standard multiple regression bogs down due to computation and non-linear dependence. GMDH can predict the outcome even with smaller training sets. The computational burden is reduced with GMDH model because it automatically filters out input properties that provide little information about location and shape of hyper surface.

2.11 Prediction Accuracy Measures

When building prediction models, the primary goal should be to make a prediction model as much accurate as possible. The model should be able to accurately predict the desired target value for new data. An important question that needs to be asked of any prediction model is 'How accurate is its predictions'. The difference between the actual and predicted value is often measured in terms of error. There are many prediction accuracy measures suggested in literature which find out the effectiveness of any prediction model.

Based on the two values namely actual value and predicted values, researchers have stated various methods to evaluate the quality of predictions [50, 65, 110]. As presented in equations (2.4) to equation (2.5) in the next section various measures to adjudge the prediction accuracy are presented. We can develop a relationship between how well a model predicts on new data (its true prediction error and the thing we really care about) and how well it predicts on the training data (which is what many models in fact measure).

In our research study, we evaluated and compared the OO software maintainability prediction models quantitatively with other proposed models. The measures used in the current study are discussed in succeeding sections.

2.11.1 Magnitude of Relative Error

It is a normalized measure of the discrepancy between actual values and predicted values as proposed by Kitchenham et al. [110]. It is measured as shown in equation (2.4) below:

$$MRE = \frac{\textit{Actual Value } - \textit{Predicted Value}}{\textit{Actual Value}} \tag{2.4}$$

MRE value is calculated for each class and further many prediction accuracy measures are derived from this measure. Maximum of MRE for all data points is used as MaxMRE, minimum MRE of all data points is used as MinMRE and Mean of MRE is used as MMRE in this research study. Their definition is provided as follows:

2.11.1.1 Mean Magnitude of Relative Error

MMRE measures the average relative discrepancy. It is equivalent to the average error relative to the actual effort in the prediction. In our study we have expressed MMRE as actual values even though in some of the studies it is expressed in percentage form (%). As shown in equation (2.5), MMRE is calculated as the mean of MRE.

$$MMRE = \frac{1}{N}\sum_{i=1}^{N} MRE \tag{2.5}$$

MMRE has been regarded as a versatile assessment criterion and has a number of advantages such as it can be used to make comparisons across datasets and all kinds of prediction model types and it is independent of measuring unit and scale independently.

2.11.1.2 Maximum of Magnitude of Relative Error

MaxMRE measures the data point which achieve maximum difference between actual values and predicted value. As shown in equation (2.6), it is calculated by taking maximum of all MRE values.

$$MaxMRE = Maximum(MRE_1, MRE_2.........MRE_{\mathrm{n}}) \tag{2.6}$$

2.11.1.3 Minimum of Magnitude of Relative Error

MinMRE measures the data point which achieve minimum difference between the actual values and predicted value of the dependent variable. As shown in equation (2.7), it is calculated by taking minimum of all MRE values.

$$MinMRE = Minimum(MRE_1, MRE_2.........MRE_{\mathrm{n}}) \tag{2.7}$$

2.11.2 Mean Absolute Relative Error

Mean Absolute Relative Error (MARE) is a normalized measure to detect the discrepancy between actual and predicted value of dependent variable (maintenance effort in this case). As shown in equation (2.8), first the difference between the actual and predicted value is calculated and the result is divided by the actual value. Then, the absolute value of each data point is summed and is divided by the total number of data points. MARE is defined as follows:

$$MARE = \frac{1}{N}\left(\sum_{i=1}^{N}\frac{|\,\textit{Actual Value} \;-\; \textit{Predicted Value}\,|}{(\,\textit{Actual Value}\,)}\right) \tag{2.8}$$

Ever since it is proposed by the Kitchenham et al. [110], it has become the de facto standard to measure the accuracy of software maintainability prediction.

2.11.3 Root Mean Square Error

Another measure used to compare the machine learning techniques is the Root Mean Square Error (RMSE) defined as the square root of the variance of the residual value. As shown in equation (2.9), the difference between the predicted values with actual values for each class is squared, then averaged and finally the square root of this average value is taken. The RMSE measure is defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\,\textit{Actual Value} \;-\; \textit{Predicted Value}\,)^2} \tag{2.9}$$

This method gives comparatively high weightage to large errors as the differences are squared before they are averaged and it is chosen when large errors are most undesirable. Lower values of RMSE indicate better fitness of the model used for predictions.

2.11.4 Pred

Proposed by Fentom and Bieman [65], it measures the proportion of the predicted values which have MRE values less than or equal to specified value. As shown in equation (2.10), it is calculated in percentage form.

$$Pred(q) = \frac{K}{N} \tag{2.10}$$

Where q is the specified value, K is a number of cases whose MRE is less than or equal to q, N is the total number of cases in the datasets. In the current study we have used most commonly values such as pred(0.25) and pred(0.30) in the field of software maintainability prediction so that we can compare our results.

2.11.5 R-Square

It is a measure of the quality of fit because it measures how well the variation in the output is explained by the targets. It is formally defined as the ‘fraction by which the variance of the errors is less than the variance of the dependent variable’. It is called as R-squared because in a simple regression model it is calculated by just squaring the correlation between the dependent and independent variables which is calculated as shown in equation (2.11) and usually denoted by ‘r’.

$$r = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2)]\,[(n\sum y^2 - (\sum y)^2)]}} \tag{2.11}$$

It is found in the research studies that if this number is equal to 1, then there is a perfect correlation between targets and outputs [65]. It is calculated by square of the correlation coefficient and 100% R-square means perfect predictability.

2.11.6 P-Values

In statistics, the p-value is used to measure how extreme are the observations, hence, it’s a function of the observed sample results. The p-values are used for testing the hypothesis of no correlation. Each p-value is the probability of getting a correlation as large as the observed value by random chance, when the true correlation is zero. If p is small, say less than 0.05, then the correlation i.e. R is significant.

2.11.7 Underestimate and Overestimate

Sometimes even if the prediction model is producing good results for MMRE value, it might happen due to the phenomenon that there are many variations between the actual value and predicted value in opposite direction and during the averaging process they nullify each other. Hence, this accuracy measure tries to find out whether there is any kind of biasness present in the model by identifying the number of underestimates and overestimate in percentage form.

2.11.7.1 Percentage of Underestimate

Sometimes the predicted value is less than actual value and sometimes the predicted value is greater than the actual value. Hence, in this accuracy measure, we count how many times the predicted value is less than actual value when total data points are provided to us. The number of underestimate values are calculated in percentage form using equation (2.12).

$$Underestimate = \frac{X}{N} * 100 \tag{2.12}$$

Where X is a number of observations in which the predicted value is less than the actual value while N is the total number of observations.

2.11.7.2 Percentage of Overestimate

In order to calculate the overestimate observations, the number of observations where the predicted value is higher than the actual value is counted and finally it is calculated in percentage form using equation (2.13) as follows:

$$Overestimate = \frac{Y}{N} * 100 \tag{2.13}$$

Where Y is a number of observations in which the predicted value is greater than the actual value while N is the total number of observations. This accuracy measure also helps in comparing the performance of multiple classifiers by presenting the visualization chart for underestimate and underestimate and provide us the much-needed information about the biasness if it at all present in the model.

2.12 Cross-Validation Methods

In order to obtain unbiased and generalized results from any empirical study, it is essential that validation of the prediction model should be carried out on different datasets other than the one used for training. Three very commonly used methods for the validation are holdout cross-validation method, n-fold cross-validation method and leave one out cross-validation method. Although, in this study n-cross-validation technique is used with setting the value of n as 10, however we have discussed all three methods in the following sub sections:

2.12.1 Holdout Cross-Validation Method

Functioning of holdout validation method is very simple as in this technique, the dataset is divided into three portion m:n:p. During the prediction process, the m portion is used for training purpose, the n portion is used for the testing purpose and finally, the p portion is used for validation purpose. In normal practice approximately 67% of the data is used for training purpose and remaining 33% is used for testing and validation. If the data size is comparatively small, the cost of setting aside a significant portion of the dataset for validation is too high. Hence, this technique is adopted only if the dataset is comparative large in size.

2.12.2 N-Fold Cross-Validation Methods

In the N-Fold cross-validation proposed by Stone [211], it works by splitting the dataset into sets of n folds. Cross-validation is very similar to the holdout method but the main difference is that in this method, each data point is used both to train the models and to test a model, but never at the same time. In this method, the data is divided into n folds where each time n-1 parts are used for training purpose and one part is used for validation purpose. This procedure is repeated n times and the results from each fold are combined to produce the model validation results. A useful feature of n-fold cross-validation is that it gives the estimates of the variability of the true error. One of the important questions of cross-validation is what should be the value of n. If the value of n is too small, we get more bias error estimates, however, the variance is less. However, if the value of n is very high, the error estimate is unbiased but on the other side, the variance could be higher. Hence, there is a trade-off between biasedness and variance which has to be taken into consideration while selecting the value of n. Another important factor while deciding the value of n is the computational time which is directly proportional to the number of folds. For each fold, we have to train a new model, so it takes more time. Generally, as a normal practice, in any empirical studies either 5-fold or 10-fold cross-validation is used.

2.12.3 Leave One Out Cross-Validation Method

Leave one out cross-validation method is an extreme case of n-fold cross-validation where the value of n is one less than the total number of data points. For example, for each dataset we just keep one fold i.e. one data point for validation and remaining n-1 data points are used for training. The process is repeated n times and in every ith iteration, the ith data point is used for validation. If the size of the dataset is very limited, this technique is adopted.

2.12.4 Discussion

If the size of the data is limited, n-fold cross-validation is preferred to the holdout method. Since in this study, the size of the dataset is greater than 80 data points, hence leave-one-out method has not been used. Hence, in this study, n-fold cross-validation method is used and the value of n is set to 10. That means in our research for each of the dataset, for an example of 100 data points; we can create 10-folds each containing 10 data points. During the prediction model building, the process is repeated 10 times irrespective of the training method used. Each time nine groups are combined resulting in 90 data points and used to train the model and 10th group of 10 points that was not used to construct the model is used to estimate the actual prediction error. In the case of 10-fold cross-validation, the final result of the error estimate is actually average of each iteration. Hence, we obtain a more robust prediction model which provides us true prediction error.

2.13 Test for Significance

Statistical tests are generally performed for comparing the prediction performance of one model with other model and identify whether one technique has really outperformed the other technique or it's just coincidental. Various statistical significance tests are suggested in literature and broadly divided into two categories i.e. parametric and non-parametric. Parametric tests make assumption that the data is normally distributed. Some examples of the parametric test are paired t-test and Analysis Of Variance (ANOVA) test. Non-parametric tests are also called as distributed free tests because they do not make any kind of assumption regarding the distribution of data. Some examples of the non-parametric test are Friedman Test, Wilcoxon Test, Nemenyi Test etc.

In this research, in order to analyze the statistical significance of the outcomes provided by various models, Friedman test followed by post-hoc analysis using Nemenyi test and Wilcoxon test are performed. We have not used paired T-Test and ANOVA test to check the significance because we were neither guaranteed that the accuracy differences are normally distributed, nor we were sure about the presence of variance in the residual error. We have followed the guidance provided by Demvsar [59] that non-parametric tests are very effective way to check the significance of the results if the dataset does not follow a normal distribution. These tests are further explained in a subsequent sections.

2.13.1 Friedman Test

The Friedman's test [72] is a kind of non-parametric distribution-free statistical test, which is used to find if there exists significant difference among the performance of machine learning techniques and rank them accordingly. As suggested by Demvsar [59], when the performance of multiple classifiers needs to be compared on multiple datasets, it is one of the best techniques. This test evaluate the performance of each classifier on given multiple dataset using prediction accuracy measure such as MRE, MMRE and RMSE and further allocate mean rank to all classifiers accordingly. The technique whose performance is adjudged as best on multiple datasets, is assigned at the lowest rank. If there are n techniques, Friedman test considers n-1 degree of freedom using the chi-square table. The following hypothesis is formed before conducting the Friedman test on results.

Null Hypothesis (H_0): There is no significant difference among the performance of participant machine learning techniques.

Alternate Hypothesis (H_1): There exists significant difference among the performance of participant machine learning techniques.

The Friedman statistic is calculated based on the formula given in equation (2.14) as follows:

$$X_r^2 = \left(\frac{12}{Nk\left(k+1\right)} \sum_{i=1}^{k} R^2 \right) - 3N\left(k+1\right) \tag{2.14}$$

Where R_i is the rank allocated to the i^{th} technique by the Friedman test, N is the number of datasets, k is number of machine learning techniques considered for ranking. The value of ✗$_{calculated}$ is calculated from the given equation (2.14) and compared with ✗ $_{tabulated}$ using chi-square distribution table. If the value of Friedman Measure i.e. ✗ $_{calculated}$ lies in the critical region, Null hypothesis is rejected and alternate hypothesis is accepted and it is concluded that there exists a significant difference between the performance of participant machine learning technique. Otherwise Null hypothesis is accepted and alternate hypothesis is rejected and concluded that there does not exist significant difference between the performances of participant machine learning technique. Further, if Friedman test found that there exists significant difference between accuracy achieved from various prediction techniques, follow-up tests such as Nemenyi test and Wicoxon tests needs to be conducted in order to evaluate and compare each pair of prediction technique. Both Nemenyi test as well as Wicoxon tests are also distribution free tests and used for post-hoc analysis. Individually every technique is ranked using Friedman's Individual Rank (FIR) as shown in equation

(2.15) as:

$$Friedman's\ Individual\ Rank\ (FIR) = \frac{C}{N} \tag{2.15}$$

Where C is the cumulative rank and N is a number of datasets. FIR for each technique is calculated and the technique which scores lowest value of FIR is considered as the best performer and the technique achieving the highest rank is termed as worst performer.

2.13.2 Post-Hoc Analysis

There are two statistical tests used for conducting post hoc analysis viz Wilcoxon Signed - Rank Test and Nemenyi Test. If the results based on the mean rank achieved using FIR for both the performance measure MMRE or RMSE is found to be significant, it is advisable to check whether the difference in mean rank is statistically significant or not by means of post hoc analysis using Nemenyi Test.

2.13.2.1 Wilcoxon Signed Rank Test

Wilcoxon Signed rank test is used to find whether there exists statistically significant performance difference among each pairs of learning technique. Like the paired t-test for correlated samples, this test is also applied to two-sample designs which involve repeated measures. As proposed by Bland and Altman [27], if there is paired value of MRE say X_a and X_b, this test will first find the absolute difference between X_a and X_b for each pair and omit those cases when X_a and X_b are equal. Next, it assigns a positive and negative rank to each pair i.e. '+' sign when X_a is greater than X_b and '-' sign when X_a is less than X_b. As shown in equation 2.16 and 2.17, it calculates the value of W for the Wilcoxon test, which is equal to the sum of the signed ranks.

$$W = \left| \sum_{i=1}^{N} (\ sign(\ X_1 - X_2).R) \right| \tag{2.16}$$

$$Z = \frac{W - 0.5}{\sqrt{\frac{n(n+1)(2n+1)}{6}}} \tag{2.17}$$

$$\alpha_{\text{new}} = \frac{\alpha_{\text{old}}}{NumberofComparisons} \tag{2.18}$$

Although, Friedman test is used to find if there exists significant differences between the performances of techniques, however, Wilcoxon Test is used to find the individual differences between given pair on same dataset if at all it exists. Hence, if the number of datasets are less than five, Friedman test is unable to do post hoc analysis and we stick to Wilcoxon test. There is only one disadvantage with Wilcoxon test that sometimes it generate Type I error, however in order to overcome this type of problem, Bonferroni adjustment is used [135]. As shown in equation (2.18), it suggests that value of significance level i.e. α should be revised and divided by k where k is the number of times Wilcoxon signed test is conducted [27]. For example, if the significant level is 0.05 and it is repeated on six datasets, the new adjusted value of α becomes (0.05/6)= 0.008 after Benferroni adjustment.

2.13.2.2 Nemenyi Test

Post hoc analysis using Nemenyi test is conducted in case significant results are yielded by the Friedman test in other words if the null hypothesis of the corresponding test is rejected in Friedman test. When the sample size is equal and the data is not normalized, Nemenyi Test is a very powerful tool for post hoc analysis [124]. It is used to compare the performance of various classifiers for finding the existence of statistically significant difference among them. First of all Critical Distance (CD) is calculated using equation (2.19) which depends upon the number of techniques, number of datasets and the level of significance as shown under:

$$Critical\ \ Distance\ \ (CD) = q_{\alpha}\sqrt{\frac{k\,(k+1)}{6N}} \tag{2.19}$$

Where k means the number of technique and n is the number of the data sample. The value of q$_{\alpha}$ is based on Studentized range statistics for a given level of significance as defined by Demvsar [59]. While comparing the performance of two machine learning techniques during post hoc analysis, the difference between their respective FIR values is calculated. If this difference is greater than or equal to the value of CD, it is concluded that the performance of two machine learning techniques is statistically significant at the selected significance level α. If the difference is less than the value of CD, it is concluded that the difference between their performances is not statistically significant.

2.14 Discussion

There are many important facet of any software such as its source code, documentation, archived communications, maintenance efforts, change management and defect tracking. This information is vital for researchers as they can mine these data repository and extract knowledge for improving overall software quality. Model can be created based on the available data for the purpose of predicting many aspects of software such as change, defect prediction, fault proneness, maintainability etc. While creating a prediction model researchers, practitioners and academicians need to understand empirical concepts and techniques [135]. In this chapter we have presented the steps to be followed for conducting an empirical study and creating a prediction model for software maintainability. We have also presented various statistical tests to be conducted to verify the significance of the results and finally interpretation of the results.

Chapter 3

Systematic Literature Review

3.1 Introduction

If the maintainability can be measured in early phases of the software development, it helps in better planning and optimum resource utilization. Measurement of design properties such as coupling, cohesion etc. in early phases of development, often leads us to derive corresponding maintainability with the help of prediction models. In this chapter, we performed a systematic review of the existing studies related to software maintainability from January 1991 to Oct 2015 based on the guidelines provided by Kitchenham et al. [111] for conducting a systematic review in the field of software engineering. In total, 96 primary studies were identified out of which 47 studies were from journals, 36 from conference proceedings and 13 studies from other sources. All studies were compiled in structured form and analyzed through numerous perspectives such as the use of design metrics, prediction model, tools, data sources, prediction accuracy etc.

The chapter is organized as follows: The systematic review begins with Section 3.2 discussing the background of the subject of research i.e. Software Maintainability. Section 3.3 describes the motivation of undertaking this review. Section 3.4 discusses the review methodology; Section 3.5 presents detailed planning which includes identifications of keywords, raising the RQ and retrieval of studies. Section 3.6 presents various activities performed while conducting this review which includes the synthesis of information, a list of qualified studies, inclusion and exclusion criterion and assigning the identifier to each of the shortlisted studies. Reporting of reviews and answer of each of the RQ are presented in Section 3.7. Section 3.8 discusses current trends and loopholes as it sets the goals and directions for future, and finally section 3.9 presents the discussion.

The results of this chapter have been reported in [145].

3.2 Background

In the current systematic review, all research papers, review articles, white papers, reports, and conferences proceedings known to authors since 1990 to date have been collected, scrutinized, compiled and analyzed. The objective of the systematic literature review is to organize empirical evidence in comprehensive form on the following aspects:

- Various factors that affect maintainability.
- Different means and methods to improve maintainability.
- The use of prediction models for maintainability in the early phases of development.
- Comparing the performance of various maintainability prediction models in terms of the accuracy.
- Identify the advantages and disadvantages of various prediction models over each other.
- Identify the software metrics which can be used in prediction model making process.
- Identification of the existing gaps for future prospect of research in the field of software maintainability.

In order to achieve the above mentioned aims, nine digital libraries were extensively searched and 96 primary studies were identified for inclusion in the study based on criteria discussed in later part of this chapter. After preliminary investigations in the previous literature, RQ were raised and detailed comprehensive reports were generated.

3.3 Motivation

While undertaking the current review on software maintainability, the two obvious questions arise as under:

- Why is it the right topic for a review? and
- Is there any recent review carried in this area?

Recently a survey conducted by Jones [40] has reported that during the 1950s only 10% of the total professionals deployed in software industry were engaged in maintenance work and by the year 2025, this figure would rise to 77%. Further, it claims that acute shortage of software personnel is also due to the burst of maintenance work. Hence, practitioners are trying hard to make maintainable software so that the overall project cost can be controlled and the product can be managed optimally. As part of the better planning, often developers predict maintainability of the software on the basis of its designed characteristics. This motivates us to compile all the studies in the said field in order to identify how much has been achieved as well as the potential areas of research based on existing gaps. Many reviews have been conducted in the past to compile the studies related to software maintainability by Ghosh at al. [78], Koh et al. [114], Riaz et al. [183], Saraiva [194] and Saraiva et al. [195], but the current review is different from all of them in three aspects. Firstly, the current review is carried out as per the guidelines provided by Kitchenham et al. [111] for conducting a systematic review in the field of Software Engineering. Secondly, in this review, empirical studies on software maintainability prediction are shortlisted and analyzed both qualitatively as well as quantitatively in tabulated form for easy understanding. A thorough analysis was conducted to cover various aspects of software maintainability predictions such as the prediction techniques, software design metrics, datasets, tools, and prediction accuracy of the models etc. Thirdly and most importantly, none of the previous reviews [78, 114, 183, 194, 195] are as comprehensive as the current one in which more than 96 studies published from the year 1991 to October 2015 are reviewed. The main guidelines provided by Kitchenham et al. [111] are covered in succeeding sections of this chapter.

3.4 Review Methodology

This section explains the procedure of conducting the systematic review adopted in this study as per the guidelines given by Kitchenham et al. [111]. As depicted in figure 3.1, the review methodology adopted in this study is divided into three stages: planning, conducting and reporting.

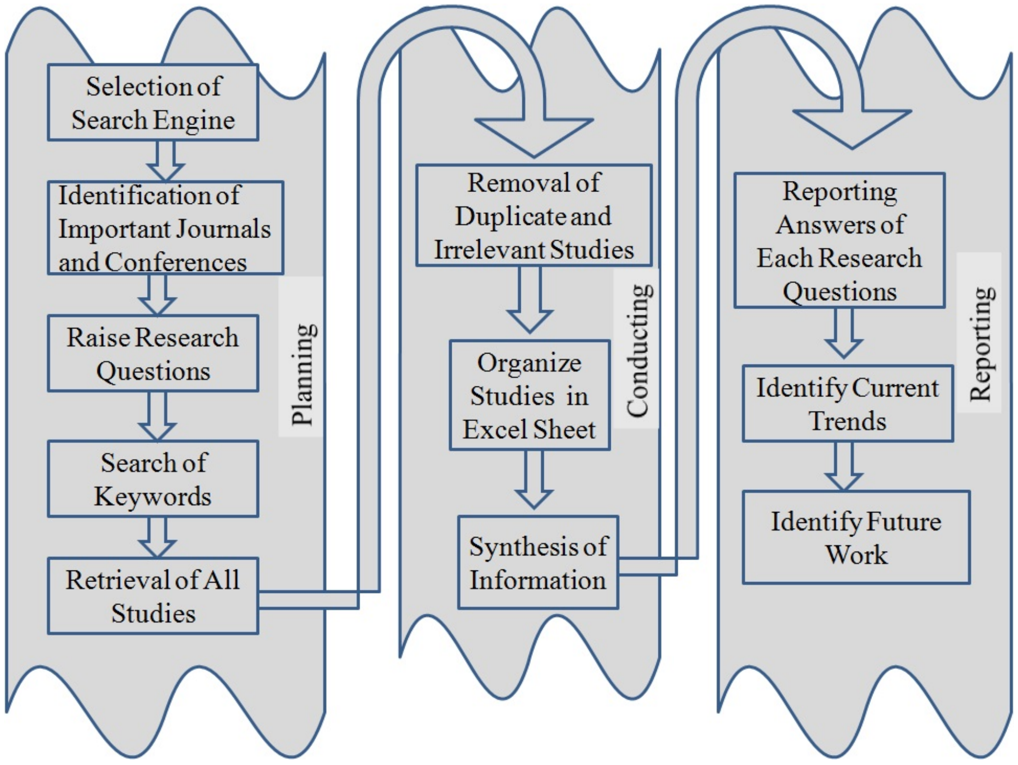
Selection of Search Engine
Identification of Important Journals and Conferences
Raise Research Questions
Search of Keywords
Retrieval of All Studies
Planning
Removal of Duplicate and Irrelevant Studies
Organize Studies in Excel Sheet
Synthesis of Information
Conducting
Reporting Answers of Each Research Questions
Identify Current Trends
Identify Future Work
Reporting

Figure 3.1: Systematic Review Process

During the planning stage, search databases were identified and after the preliminary investigations, RQs were formulated. Policies regarding inclusion and exclusion of the studies were prepared and all the relevant papers were extracted. During the second stage, all duplicate and irrelevant studies were removed and shortlisted papers were organized. The synthesis was also carried during this stage on the basis of the information provided in each of the shortlisted study. In the last stage, answers to all RQs were reported and current trends in each of the sub-field were identified. An endeavor was also been made to highlight the constraints present in each of the paper which would lead us to the future directions of research.

3.5 Planning for Review

3.5.1 Selection of Search Databases

We began our search with the university website resource and EBSCO discovery services provided by the university information resource centre of Guru Gobind Singh Indra Prastha University (GGSIP). The search process was accomplished in two stages. Firstly search string such as "Software Maintainability" was used to get thousands of results. In this process Google scholar, Scopus, Science Direct, Springer, ACM Digital Library, IEEE Xplore, Wiley, Web of Science and Compendex were identified. In the second stage, publications in the important journals and conferences were identified. We restricted our search from the period from Jan 1991 to Oct 2015 only.

3.5.2 Identification of Important Journals and Conferences

Before picking up good journals or conferences, their quality was determined through several perspectives to make the contents trustworthy. The basis for short listing of important journals and conferences in the said field in this review includes a number of citations, circulation of journals, status of reviewers, impact factor and above all the quality of the editorial board. Journals shortlisted for review in the field of software maintenance along with their details are summarized in Table 3.1.

Table 3.1: List of Important Journals in the Field of Software Maintenance

S.No.	Name of the Journal	Publisher	Impact Factor
1	ACM Transactions on Software Engineering and Methodology	ACM, New York	3.958
2	IEEE Transactions on Software Engineering	IEEE Inc	3.569
3	Computer Science - Research and Development	Springer	2.128
4	International Journal of Computer Science and Engineering Research and Development	PRJ Publication	1.9022
5	Empirical Software Engineering	Springer	1.854
6	Automated Software Engineering	Springer	1.4
7	Asian Journal of Information Technology	Medwell Publishing	1.2679
8	Journal of Systems and Software	Elsevier	1.117
9	Software Quality Journal	Springer	0.974
10	Journal of Software Maintenance and Evolution Research and Practice	John Wiley & Sons Ltd	0.844
11	International Journal of Software Engineering and Its Applications	SERSC	0.7621
12	International Journal of Software Engineering and Knowledge Engineering	World Scientific	0.447
13	International Journal of Software Engineering Software Eng.	Competence Center	0.219
14	ACM SIGSOFT Software Engineering Notes	ACM	NA

Major conferences of international repute that address issues in the field of software maintenance were also identified such as International Conference on Software Engineering (ICSE), International Conference on Software Maintenance (ICSM), Computer Software and Application Conference (COMPSAC), International Symposium on Empirical Software Engineering and Measurements (ESEM), International Conference on Program Comprehension (ICPC), Object-Oriented Programming, Systems, Languages & Applications (OOPSLA), European Conference on Object-Oriented Programming (ECOOP), World Conference on Reverse Engineering (WCRE) and European Conference on Software Maintenance and Reengineering (ECSMR). The research studies included in the proceedings of these conferences were also taken into consideration.

3.5.3 Formulation of Research Questions

Practitioners are consistently haunted by the fact that maintenance consumes the lion's share of the total project cost. After the research of almost four decades, we are in a position that we can predict the maintenance behavior of the software using mathematical prediction models in the early phases of SDLC on the basis of its design metrics. After the first round of investigations, next step was to raise certain RQ to identify the trends and scope of future research. Careful selection of RQ is very important as it helps us throughout our study from getting lost or being deviated off-track while navigating through vast information and identifying the related information from the shortlisted papers. Table 3.2 summarizes the list of important RQs set in the current review.

Table 3.2: List of Research Questions

S.No.	**Research Questions (RQ)**	**Aim**
RQ 1	Which techniques have been used for the Software maintainability predictions?	To identify prevalent statistical, machine learning or evolutionary techniques.
RQ 2	Which metrics is found useful and most cited?	To identify most influential metrics among various proposed metric suite.
RQ 3	What are the various tools prevalent in the industry for metrics collection?	How to calculate values of design metrics to measure various design characteristics of any given software.
RQ 4	What kinds of datasets are being used for the software maintainability prediction?	To be able to identify prevalent datasets and the need of new datasets.
RQ 5	What kinds of prediction accuracy measures are used?	To be able to identify the prevalent prediction accuracy measures used to judge the performances of prediction models.
RQ 6	Whether the performance of machine learning or evolutionary techniques is better than the Statistical techniques?	To be able to compare the performance of the statistical, machine learning, and ET.
RQ 7	Is there any effect of refactoring on the software maintainability?	To be able to identify various refactoring methods that could improve the code quality.
RQ 8	How can we measure the maintainability?	Although subjective in nature, its answer would help us in identifying methods to measure maintainability.
RQ 9	What are the advantages of the software maintainability prediction?	Identify the advantages of software maintainability prediction at early phases of SDLC.

3.5.4 Search of Keywords

While conducting this systematic review, our intention was to compile all the papers on the subject under study and present a holistic view towards the end. Final search string formulated as under:

(Software Maintainability AND Software Maintenance) OR (Prediction OR Probability) AND (Classification OR Regression OR Machine Learning OR Artificial Neural Network OR Tree Net OR Multiple Regression OR Decision Tree OR Support Vector Machine OR Evolutionary Technique OR Ensemble Lerner) OR (Software Maintenance AND Refactoring) OR (Accessing Software Maintenance) OR (Software Metrics AND Software Maintenance)

The papers which include keywords such as Software Maintenance, Software Maintainability, Software Design Metrics and Maintainability Index were identified during the initial search. After going through the contents, we went into further details and in the next phase other keywords such as Changeability, Modifiability, Refactoring and Prediction Modeling were also used to refine our search process. Since our aim was to narrow down on the 'Software Maintainability Prediction, articles working on Change Proneness, Fault Proneness, Error Prone and Defect Prediction were deliberately avoided.

3.5.5 Retrieval of Studies

All important studies were retrieved through respective digital libraries. During this process, reference section of each selected study was further explored to find relevant research publications/ reports which were further retrieved and organized for review.

3.6 Conducting Review

During the conduct of the systematic review, all papers were studied thoroughly and adjudged against the RQs as compiled in Table 3.2. Constraints and limitations present in each of the shortlisted study were also identified and compiled which gives directions for future research. Detail synthesis of information took place during this stage only. We set certain criteria based on RQs and continue accessing each paper based on ibid criteria.

3.6.1 Removal of Duplicate and Irrelevant Studies

All the shortlisted studies were arranged in chronological order so that the duplicate studies could be immediately identified and removed. Further contents of the articles were thoroughly scanned and irrelevant studies were removed as discussed among the authors of this study. Inclusion and exclusion criteria were drawn directly from the RQs raised in Table 3.2. The review criterion was set on the basis to find some reasonable relevant contents in the context of software maintainability. Only those papers were shortlisted which predict software maintainability by proposing some models, measure maintainability using design metrics, discuss some empirical investigations or perform some activity which had a direct or indirect impact on maintainability such as refactoring, documentation etc along with their consequences on maintainability. Any paper diverting towards change proneness or error proneness was simply dropped. Criterion was kept neither too narrow which may result to an over exclusion threat nor too broad as it may include poor or irrelevant studies.

Inclusion criteria:

- Empirical studies using the machine learning techniques.
- Empirical studies comparing the performance of machine learning techniques and statistical techniques.
- Empirical studies proposing some hybrid techniques by combining machine learning techniques with some non machine learning techniques.

Exclusion criteria:

- Studies on software maintainability without empirical analysis of results.
- Empirical studies in which dependent variable other than ‘Change’ was used.

- Studies using the machine learning techniques in any other context.
- Review studies.
- Replicated extended paper of the conference into the journal by same author, however, utmost care was taken to identify if the results are different, in which case both the studies were considered for the review.

Initially, 179 studies in total were retrieved using various search engines on the basis of the keywords identified for the search in section. By applying the above-mentioned inclusion-exclusion criteria, 108 studies were shortlisted. In the next phase, we filtered out the irrelevant studies from the huge pool of available information by checking against the quality assessment criteria as presented in Table 3.3 to weigh the relevance of each study

If the paper is qualified as per the quality criteria, it is given one mark, 0.5 marks for partly qualified and 0 for not qualified. The final score was calculated for each of the studies after adding the score obtained from each of the quality assessment measures. Hence, a study could have a maximum score of 12 and a minimum score of 0. All those studies which scored less than 5 were again dropped from the review process. Many brainstorming sessions were conducted and twelve studies [1, 21, 23, 31, 41, 90, 108, 154, 160, 190, 197, 201] were further dropped and finally 96 primary studies were only shortlisted for review on software maintainability.

3.6.2 Data Extraction

“Place for everything and everything would be in place” was taken as a guiding principle to organize all shortlisted articles for easy and quick access and retrieval before setting up the stage for conducting such an extensive review. We prepared database containing various attributes such as author’s name, article name, date of publication, the source of publications (journal/ conference/ position paper/symposium/ white paper), keywords, article’s abstract and remarks. A separate column was also maintained using ‘hyperlink’ for creating a link

Table 3.3: Quality Assessment Measures

S.No.	Quality Criteria (Q)	Score		
		Yes	Partly	No
Q1	Whether the aims of the research study was clearly stated?			
Q2	Whether the independent variables were clearly defined?			
Q3	Whether the data collection procedures were clearly defined?			
Q4	Whether any tool was used to collect the variables? If yes, is it explained?			
Q5	Whether the use of prediction techniques was clearly defined and justified?			
Q6	Whether threats to validity in the empirical study were clearly specified?			
Q7	Whether the adopted research methodology is repeatable?			
Q8	Whether the study is referring to a specific type of maintenance or it is picking maintainability as a whole problem?			
Q9	Whether comparison between performances of various techniques was conducted?			
Q10	Whether the proper and relevant literature survey was conducted?			
Q11	Does the study have consistent and adequate citation over the years?			
Q12	Whether prediction accuracy measures were clearly defined and used to measure the outcome in the study?			

to the corresponding files stored in a separate folder for easy access. No file was stored without creating a record in the database. We found this model to be very comfortable. A detailed synthesis of information present in each of the study was performed to find the answers of all RQ's. Table 3.4 presents lists of shortlisted articles with respective author and reference number. It also assigns an identifier to each of the shortlisted studies which are further referred in rest of this chapter. We examined all the shortlisted studies from numerous perspectives and tried to identify the relationship between these studies. If the collection of studies states similar or comparable viewpoints, it helped us in providing the evidence before reaching any generalized conclusions.

Table 3.4: List of Selected Studies in the Field of Software Maintenance

StudyNo	Author(s)	Ref	StudyNo	Author	Ref
S1	Aggarwal et al.	[5]	S49	Lucia et al.	[133]
S2	Aggarwal et al.	[4]	S50	Malhotra & Chug	[137]
S3	Aggarwal et al.	[6]	S51	Malhotra & Chug	[141]
S4	Arisholm & Sjoberg	[8]	S52	Malhotra & Chug	[140]
S5	Baker et al.	[14]	S53	Malhotra & Chug	[139]
S6	Balogh et al.	[12]	S54	Malhotra et al.	[146]
S7	Bandi et al.	[13]	S55	Misra	[162]
S8	Banker et al.	[14]	S56	Mishra & Sharma	[161]
S9	Baqais et al.	[16]	S57	Muthanna et al.	[164]
S10	Basgalupp et al.	[17]	S58	Niessink & Vliet	[166]
S11	Briand et al.	[34]	S59	Oman & Hagemeister	[168]
S12	Bhattacharya & Neamtiu	[24]	S60	Oman & Hagemeister	[169]
S13	Broy et al.	[37]	S61	Ping	[172]
S14	Chen & Lum	[42]	S62	Pizka & Deisenbock	[173]
S15	Coleman et al.	[49]	S63	Polo et al.	[174]
S16	Dagpinar & Jhanke	[53]	S64	Poole & Huisman	[175]
S17	Dahiya et al.	[54]	S65	Prasanth et al.	[176]
S18	Daly et al.	[55]	S66	Prechelt et al.	[177]
S19	Deissenboeck et al.	[57]	S67	Rajaraman & Lyu	[179]
S20	Deligiannis et al.	[58]	S68	Dubey et al.	[61]
S21	Elish & Elish	[62]	S69	Ranmil et al.	[180]
S22	Ferneley	[67]	S70	Ramil & Smith	[181]

Continued on next page

Table 3.4 – continued from previous page

Study No.	Author(s)	Ref	Study No.	Author(s)	Ref
S23	Fioravanti & Nesi	[68]	S71	Riaz et al.	[183]
S24	Burki & Harald	[39]	S72	Riaz et al.	[184]
S25	Genero et al.	[76]	S73	Schneberger	[197]
S26	Grady	[80]	S74	Schneidewind	[198]
S27	Garcia et al.	[74]	S75	Sheldon et al.	[200]
S28	Hanenberg et al.	[84]	S76	Shibata et al.	[119]
S29	Harrison et al.	[85]	S77	Stark et al.	[209]
S30	Hatton	[86]	S78	Sneed	[203]
S31	Hayes & Zhao	[87]	S79	Sneed and Meray	[204]
S32	Hegedus	[88]	S80	Soni & Khaliq	[205]
S33	Hirota et al.	[91]	S81	Stavrinoudis et al.	[210]
S34	Jeet et al.	[98]	S82	Sun & Wang	[213]
S35	Jin & Liu	[100]	S83	Thongmak & Muenchaisri	[217]
S36	Jorgensen	[101]	S84	Thwin & Quah	[219]
S37	Kabaili et al.	[102]	S85	Upadhyay et al.	[220]
S38	Kataoka et al.	[104]	S86	Velmourougan et al.	[221]
S39	Kaur and Kaur	[105]	S87	Vivanco & Pizzi	[222]
S40	Kaur & Singh	[108]	S88	Welker et al.	[224]
S41	Kaur et al.	[106]	S89	Wen-Hua	[225]
S42	Kumar & Dhanda	[121]	S90	Xing & Stroulia	[228]
S43	Kumar	[120]	S91	Yamashita & Moonen	[230]
S44	Kemerer & Slaughter	[109]	S92	Ye et al.	[234]
S45	Koten & Gray	[118]	S93	Ying et al.	[235]

Continued on next page

Table 3.4 – continued from previous page

Study No.	Author(s)	Ref	Study No.	Author(s)	Ref
S46	Li & Henry	[127]	S94	Zhang et al.	[236]
S47	Lim et al.	[130]	S95	Zhou & Leung	[237]
S48	Lin & Wu	[131]	S96	Zhou & Xu	[238]

Visualization techniques were also used as it presents a considerably larger amount of data in a compressed form using a picture which is always worth more than a million words. In the present study, it helped us in quickly absorbing, interpreting, enhancing the clarity by proving aesthetic appeal to the compiled data. We have used various visualization techniques such as line graph, pie chart, bar chart etc to categorize various methods, models, performance measures, metrics, year of publication and use of tools in the present study.

3.6.3 Distribution of Papers According to Source of Publication

All the shortlisted studies were divided into three parts as per the source in which they are published i.e. published in journals, published in conferences and others which include book chapters, technical reports, white papers, study material, symposium etc. In figure 3.2 distribution of the papers as per the three categories is depicted.

It was found that 49% papers were published in various journals of international repute, 38% papers were published in international conference whereas 13% sources were miscellaneous. Trend in journals and conference is almost similar for the publication where as very few papers were published as while paper (only one), technical report (only two) and .

3.6.4 Distribution of Papers According to the Year of Publication

Maintainability was first introduced by Belady and Lehman [19] in the year 1976 when the cost of producing the software were quite high with fragile output. They raised need of a design methodology that expresses the understanding and intentions of the designer unambiguously and completely. During this period, researchers concentrated more on good

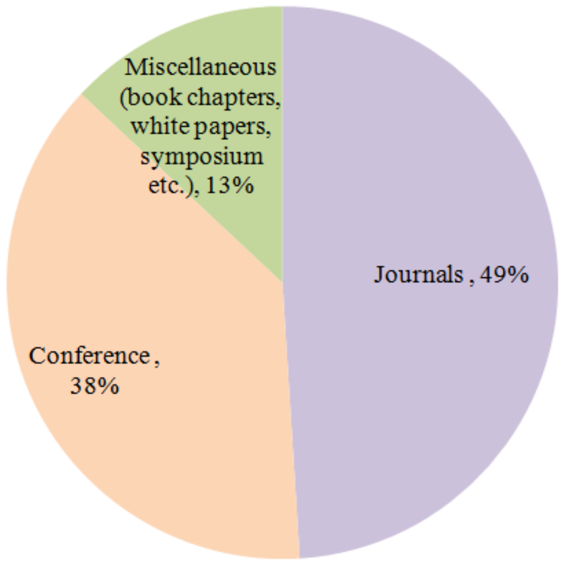

Figure 3.2: Distribution of Studies according to the Source of Publication

programming rather than designing to make the software maintainable because at that time the development of large-scle program was unpredictable. Problems in maintenance process were identified by Lientz and Swanson [129], Martin and McClure [150], Nosek and Palvia [167] etc. and many metrics were suggested [83, 168] to measure procedural languages. Rombach [186] suggested that all those metrics which were useful for procedural language cannot be applied blindly on OO languages.

Practitioners started giving importance to design rather than code and proposed various metrics to measure different design aspects of OO paradigm such as coupling, cohesion, polymorphism, inheritance etc. Further, with the help of empirical investigations, strong correlation between software design metrics and subsequent maintainability were identified. Many metric suites were proposed during this period like Chidamber and Kemerer [43], Li [126], Chen and Lum [42], Lee [123], Li and Henry [127], Martin [151], Fernando and Rogerio [66], Lorenz and Kidd [132], Tang et al. [216], Abreu and Carapuca [2].

Critical analysis of such metric suite has also been done by Grady [80], Chucher and Martin [45], Mayer and Hall [155], Hitz [92]. Many empirical investigations were also carried out to verify and validate proposed metric suite such as Dagpinar and Jahnke [53], Elish and Elish [62], Kaur et al. [106], Koten and Gray [118] and Zhou and Leung [237] in terms of their effect on maintainability. From the year 2000 onward, researchers agreed that we can measure maintainability by measuring the number of changes during operation. Briand et al. [34] suggested that the design metrics can be used for quality prediction at quite early stages of SDLC. The distribution of years for all the shortlisted studies from the year 1991 to 2015 is presented in figure 3.3.

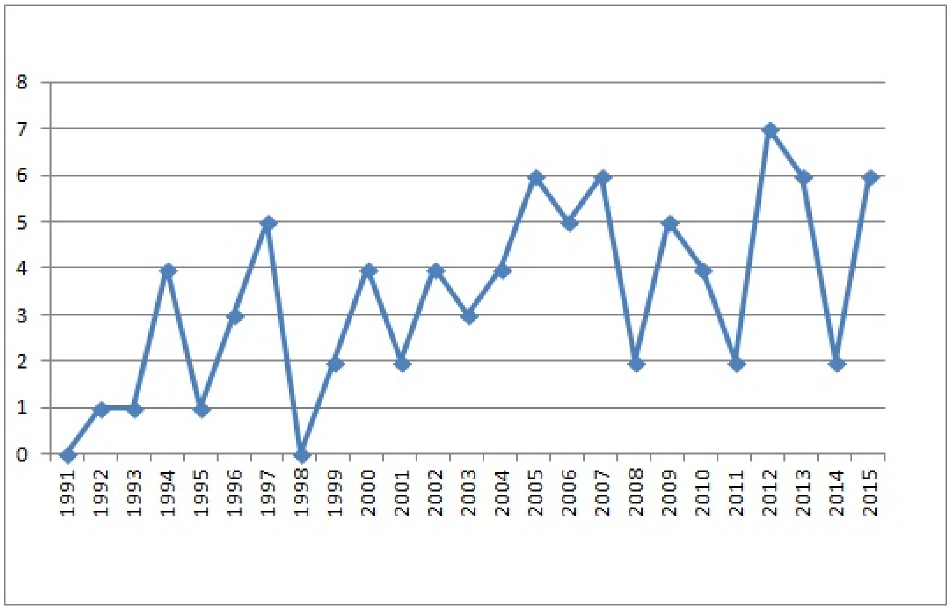

Figure 3.3: Year Wise Distribution of Studies

It is clearly evident that the research on software maintainability has been quite consistent over the years. When we analyzed the contents we found that from the year 1991 to 2004, few important studies build statistical models to predict maintainability using design metrics as they argued that since the value of change would be available only during operations and it's too late by then; we should be able to predict maintainability with the help of design metrics at the earlier stages of SDLC. From the year 2005 onward, a shift has been observed towards the use of machine learning techniques in prediction modeling. Hybrid models were also applied by many researchers during this period in prediction modeling process.

Recently, interest in nature inspired algorithms called as evolutionary methods has also been seen due to their obvious advantages as elaborated further in succeeding sub-section.

3.7 Reporting of Review

We have analyzed all shortlisted studies in the last two and half decade collected from journals, conferences, symposiums etc in the field of software maintainability with an aim to find answers of all the RQs raised in section 3.5.3 above. The following sub-sections present summarized facts and answer to all RQs.

3.7.1 RQ1: Techniques used for Software Maintainability Prediction

Researchers are experimenting new prediction models every day to predict the software maintainability more precisely.

To establish the relationship between software design metrics as the independent variable and maintainability as the dependent variable, various techniques have been practiced in last two and half decade which can be broadly classified in five categories i.e. Statistical techniques, Machine Learning techniques, Expert Judgment and Feed backs, Nature Inspired Techniques and Hybrid techniques. In Table 3.5 all the studies are categorized as per the methods used in them. When we chronologically arranged the choice of models applied for software maintainability prediction, it was quite evident that during the initial period statistical methods such as Linear Regression (LR), Multivariate Binary Logistic Regression (MBLR), Stepwise Logistic Regression (SLR), HMM etc were used. These methods were not only highly mathematical in nature but also unable to handle noise present in the data.

In the later phases from the year 2000 onward, more robust models based on machine learning techniques such as ANN [6, 219, 237] were applied. It works as an excellent alternative to logistic regression since it offers a number of advantages such as less formal statistical training is required, all possible interaction between independent and dependent variables can be identified and complex non-linear relationship can be judged using ANN. In figure 3.4 we have visualize the use of various learning techniques in prediction modeling.

Table 3.5: Types of Modeling/ Methods used to Judge Maintainability

Type of Techniques	**Prominent Prediction Techniques**	**Respective Study Identifier**
Statistical Techniques	Binary Logistic Regression, Multivariate Binary Logistic Regression, SLR, HMM, Multiplicative Adaptive Spline Regression, Random Forest, Naive Bays Classifier, Multilayer Perceptrons, Bagging, Boosting, Projection Pursuit Regression, SVM	S7, S8, S12, S16, S22, S23, S27, S31, S35, S36, S38, S40, S41, S46, S57, S59, S63, S66, S84, S88, S89, S92, S95, S96 .
Machine Learning Techniques	Neural Network Based Models, ANN, Fuzzy Inference System (FIS), Adaptive Neuro FIS (ANFIS), Fuzzy algorithms, Fuzzy Repertory Table (FRT), BBN, Tree Nets, Decision Trees (DT), Data Clustering (DC), Self-Organizing Map (SOM), Generative Topographic Map (GTM), Case-Based Reasoning (CBR), Association Rules (AR) etc.	S1, S2, S3, S13, S21, S34, S35, S39, S41, S43, S45, S49, S51, S53, S68, S72, S84, S92
Nature Inspired Techniques	Evolutionary techniques, Genetic Programming, ACO, PSO CA, Simulated Annealing, Hill Climbing	S5, S6, S10, S17, S50, S82, S87
Expert Judgment Feedbacks	Questionnaires, Surveys, Opinions etc	S12, S18, S20, S22, S25, S29, S47, S55, S65, S66, S73, S81
Hybrid Techniques	Genetic Algorithm with Neural Network (GANN), Neural Network Evolutionary Programming (NNEP), Evolutionary with Fuzzy Network, (GFS-GSP), Genetic-Based Fuzzy Rule Base Construction and Membership Functions Tuning (GFS-RB), Evolutionary with Neural Network, S9 (Evolutionary + Neural), S10 (Evolutionary + Decision Tree), S17 (Fuzzy + Genetic), S25 (UML + Expert Judgment), S47 (Hybrid network with Parallel Computing), S82 (Regression with Neural network), S92 (Multiple Classifier)	S9, S10, S17, S25, S47, S82, S92

In this method, available data from history is often divided into three sets: Learning Set, Validating Set, and Testing Set. Learning Set is the sequence in which the dependent variable is shown to the network during the learning phase and weights are assigned to the independent variables depending upon the values of the independent variable. The network

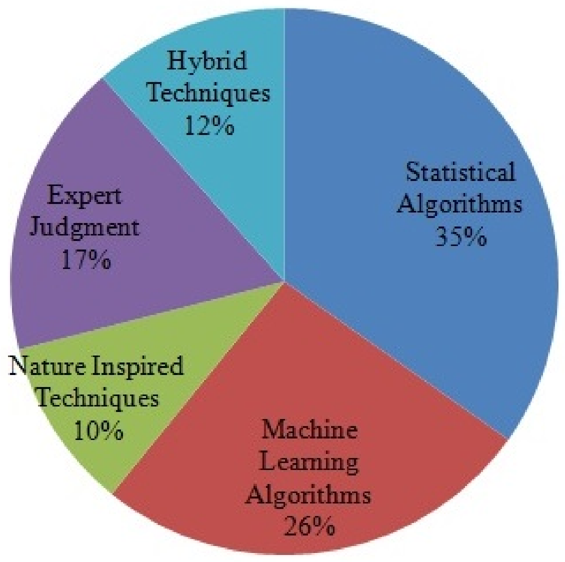

Figure 3.4: Distribution of the use of Learning Techniques in Literature

continuously adapts itself to achieve the particular value of the dependent variable and during this process, values of the assigned weights are changed. The difference between the required output and actual output is measured using validating set to identify whether the learning can be finished. Lastly, Testing Set is used to test whether the network is capable of predicting for the unforeseen data or not. Various variants of ANN such as FFNN, BPN, Kohonen Self-Organizing Network (KSON), Radial Basis Function Network (RBF), PNN, GRNN, Adaptive Neuro-Fuzzy Inference System (ANFIS) were also explored for their predictive capabilities as discussed in chapter 2.

From the year, 2012 onwards, hybrid methods have proven to be even better in prediction, for example, Multiple Classifier (MC) is used by Ye et al. [234] and GMDH is used by Malhotra and Chug [137]. Although nature inspired techniques are used in prediction modeling in various fields of software engineering other than maintenance such as ACO was applied by Azar and Vybihal [9] in software quality prediction, Particle Swarm Optimization was applied by Saed et al. [190] in software performance prediction and Genetic model was applied by Burgess and Leey [46] for software effort estimation. However, the applications of these models for software maintainability prediction were reported to be very less wherein only four studies could be found [12, 17, 213, 222]. Balogh et al. [12] did their study on prediction of development effort, Basgalupp et al. [17] did their study on prediction of maintenance effort, Sun and Wang [213] performed an empirical study on prediction of preventive maintenance and Vivanco and Pizzi [222] identified important software metrics using genetic techniques which can be used in prediction model. Well accepted, established, recognized and generalized prediction method is still awaited by the software industry.

3.7.2 RQ2: Metric Suite used in Software Maintainability Prediction

The relationship between software design metrics and corresponding maintainability has been proposed and validated by many researchers. With the help of many empirical studies, it has been established that the quality of the software design, as well as code, is very important to enhance software maintainability. We observed that during the period from 1969 to 1990, traditional metrics like function point, Halstead Software Science [83] and McCabe's CC [156] were used to judge the quality of procedural languages. Quantitative measure to calculate Maintainability Index (MI) proposed by Oman and Hagemeister [168] is given in equation (3.1).

$$MI = 171 - 5.2 * ln(HV) * 0.23 * CC - 16.2 * ln(LOC) + 50sin(\sqrt{2.4 * COM}) \quad (3.1)$$

Where HV is Halstead Volume metric [83], CC is Cyclomatic Complexity metric [156], LOC is counted as a lines of code, and COM is a percentage of comment lines. With the invention of OO paradigm, traditional metrics mentioned above was no longer effective since other characteristics such as inheritance, coupling, cohesion and polymorphism present in the code take the charge. Subsequently the necessity was highlighted by Rombach [186] and new metric suite to measure the design characteristics of OO software was proposed by Chidamber and Kemerer [43] famously known as Chidamber and Kemerer metric suite.

Some revisions into the Chidamber and Kemerer metric suite were made by Li [126] and

Table 3.6: Metric Suite Proposed in Empirical Studies

Studies	Ref	Referred in Following Studies	Total
Halstead	[83]	S23, S59, S60, S81	5
McCabe	[156]	S23, S59, S60, S81	5
Chidamber and Kemerer	[43]	S3, S11, S16, S21, S23, S35, S39, S40, S45, S50, S51, S52, S53, S65, S68, S84, S95	18
Li	[126]	S3, S11, S16, S21, S23, S35, S39, S40, S45, S50, S51, S52, S53, S55, S56, S65, S68, S84, S95	20
Fernando and Rogerio	[66]	S55	2
Lorentz and Kidd	[132]	S55	2
Tang et al.	[216]	S84	2
Abreu and Carapuca	[2]	S55	2
Aggarwal et al.	[5]	S1, S17, S48	4
Chen and Lum	[42]	S14	2
Fioravanti and Nesi	[68]	S23	2
Li and Henry	[127]	S3, S11, S16, S21, S23, S35, S39, S40, S45, S50, S51, S52, S53, S65, S68, S84, S95	18
Lin and Wu	[131]	S48	2
Prasanth et al.	[176]	S64, S65	3
Sheldon et al.	[200]	S75	2
Stavrinoudis et al.	[210]	S81	2

two more metrics were added. Although Chidamber and Kemerer metric suite was criticized by many researchers [53, 62, 106, 118, 137] for number of reasons such as Lack of Cohesion (LCOM) is not true representation of cohesiveness, however, proposed metric suite became quite popular and further referred by many researchers in their empirical studies from the year 1990 to date as compiled in 3.6. During the same period, many researchers proposed fresh metric suites while others revised or modified the existing metric suites. We collected all proposed metric suites (resulted in 41 studies) and filtered highly cited studies in the field of software maintenance, resulting in 16 studies as summarized in Table 3.6.

We further compiled the data and presented it in visualization form in figure 3.5. It is quite evident from the figure, that the metric suites proposed by Chidamber and Kemerer [43] and Li and Henry [127] are the most commonly used metric suites in empirical validations.

We also observed that many researchers empirically evaluated the effect of only one

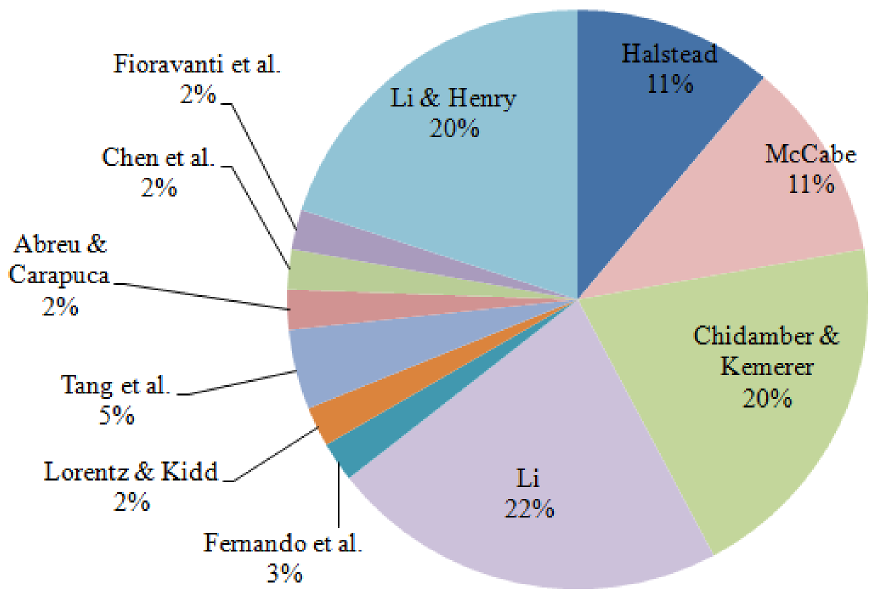

Figure 3.5: Distribution of the use of Metric Suite in Literature

particular design metric on maintainability, for example, inheritance was evaluated by Daly et al. [55], Harrison et al. [85], Prechelt et al. [177] and Sheldon et al. [200], coupling by Rajaraman and Lyu [179], UML diagram by Genero et al. [76], cohesion by Kabaili et al. [102], code metrics by Polo et al. [174]. Inconsistencies in metric naming convention were also seen, for instance somewhere two different names (DIT (Depth of Inheritance) and DIH (Depth of Inheritance) represent same concept as both represents inheritance to measure the longest distance from root node to leaf node whereas at some other places, the same name is used to represent two different concepts (DC is used to represent Descendant Class as well as for measuring Direct Cohesion).

3.7.3 RQ3: Various Tools Used to Calculate Values of the Design Metrics?

In order to find the answer for the RQ2, we observed that a large number of metrics have been suggested in the literature to measure the design characteristics of given software. Numerous free, as well as proprietary tools, have also been developed to collect the values of these design metrics from the given source code so that specific characteristics of a software

Table 3.7: List of Important Tools to Measure Design Metrics

S.No.	Tool Name	Free/ Proprietary	Developed by	Remarks
1.	'C and C++ Code Counter' (CCCC)	Open Source	Tim Littlefair	It analyzes C++ and Java and generates reports for LOC, Chidamber and Kemerer metric suite, and Henry & Kafura metric suite.
2.	Dependency Finder	Open Source	Jean Tessier	This application comes as a command-line tool for analyzing compiled Java code and creating dependency graphs. It is also used for computing OO software metrics to give an empirical quality assessment of given code.
3.	Eclipse Metrics Plug-in 1.3.6	Open Source	Frank Sauer	It's a plug-in for Eclipse platform which provides a calculation of metrics and dependency analyzer.
4.	Eclipse Metrics Plug-in 3.4	Open Source	Lance Walton	It's a plug-in for Eclipse platform which provides a calculation of metrics and dependency analyzer.
5.	Vizz Analyzer	Open Source	Rudiger Lincke	It's a framework designed to support maintenance and re-engineering.
6.	Understand	Proprietary	Proprietary	It is a reverse engineering, code exploration and metrics tool for Java source code. It is a static analysis tool for maintaining measuring and analyzing critical or large code bases.
7.	Analyst4J	Proprietary	Proprietary	It comes as Eclipse plug-in and offers an environment to visualize code quality with the help of metrics and charts. Apart from helping in estimating efforts, it also helps in identifying problem areas and respective refactoring method.
8.	OO Meter	Software Metrics Research Group (SMRG)	Alghamdi et al.	It can be used to quantitatively measure a number of qualities attribute which includes requirement specifications and design models. It supports OO metrics such as coupling, cohesion and code metrics such as LOC.
9.	CKJM	Open Source	Diomidis Spinellis	Chidamber and Kemerer Java Metrics is an open source command-line tool which calculates the metrics by processing the byte code.

code could be measured. In Table 3.7, we have summarized few prevalent tools along with their details.

3.7.4 RQ4: Kind of dataset Used for Empirical Validations

Many researchers have conducted empirical studies in part to prove that the values of design metrics significantly affect maintainability. All these studies are either based on small projects, proprietary software datasets', open source software, dataset published by NASA, PROMISE repository or taken from students' projects as compiled in Table 3.8.

Table 3.8: Kind of dataset Used for Empirical Validations

S.No.	**datasets**	**Referred in Studies**	**Total**
1.	Li and Henry	S3, S16, S21, S39, S40, S45, S51, S84	8
2.	Open Source	S9, S12, S55, S66	4
3.	Proprietary software	S10, S14, S15, S20, S22, S36, S48, S52, S64, S65	10
4.	Students Projects	S2, S25, S29, S48	4

Further, when we visualized the use of dataset in figure 3.6, we found that Li and Henry [127] dataset is quite popular. To empirically validate and evaluate the effect of each metrics on software maintainability, even though the availability of data is still a concern, research studies have taken real life data whereas many studies have used the dataset proposed by Li and Henry [127] from two commercial software's UIMS and QUES.

Maintenance Efforts are generally calculated by counting the number of lines added, deleted or modified during operations. The source code of old and new versions was collected and analyzed against modifications made in every class. Values of OO software design metric suite were calculated and combined with corresponding changes made into that class so as to generate datasets which were further used while implementing prediction model.

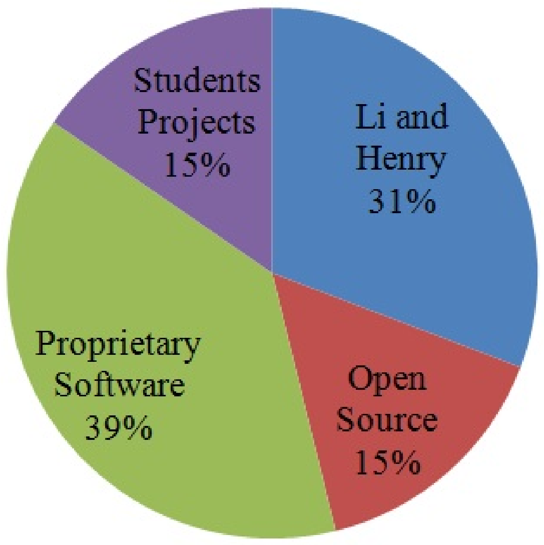

Figure 3.6: Distribution of the use of dataset in Literature

3.7.5 RQ5: Accuracy Measures to Judge the Performance of Prediction Models

For the prediction of maintainability, whatever tool, methods or datasets were suggested in the literature, it is observed that the predicted values of the dependent variable on test data is not very close to actual values. Hence, a number of statistical measures have also been proposed by Conte et al. [50], Kitchenham et al. [110], and Fentom and Bieman [65] to measure the prediction accuracies.

All these studies have presented some measures to ensure that the accurate prediction is not due to sheer coincidence, but it exist in reality by proposing some formulas with their corresponding interpretations. Widespread parameters proposed to measure the significance of prediction model are Absolute Relative Error (ARE), MARE, StdDevARE (Standard deviation of ARE), MRE, MMRE, MaxMRE, Pred (q), R-Square, P-values and Root Mean Square Error (RMSE) as summarized in Table 3.9.

Much attention has been paid for the development models capable of more accurate and precise predictions while adhering to few fundamental characteristics like the models must

Table 3.9: List of Commonly Used Prediction Accuracy Measures

Name	Definition	Referred in Studies	Total
MRE	Measure of the discrepancy between actual values and predicted value	S35, S45, S51, S52, S53, S84, S95	7
MARE	Normalized measure of the discrepancy between actual values and predicted value	S35, S39, S51, S52, S53	5
MMRE	Average relative discrepancy.	S3, S21, S45, S51, S52, S53, S84, S95	8
RMSE	Root Mean Square Error	S3	1
Pred	What proportion of the predicted values have MRE less than or equal to specified value	S21, S45, S51, S52, S53, S95	6
R-Square	Measure of how well the variation in the output is explained by the targets	S2, S21, S35	3
P-values	Used for testing the hypothesis of no correlation	S3, S21, S35	3
Pearson Coefficient of Correlation	Standard Deviation of two series is compared	S35	1

be independent of the language and technology, simple to calculate, straightforward for interpretations, and easy to understand. We analyze the papers and identified the kind of accuracy measure used in them.

As compiled in figure 3.7, we found that MRE and MMRE are quite prevalent for measuring the prediction accuracies and used by many researchers i.e. S3, S9, S10, S12, S14, S15, S16, S20, S21, S22, S39, S40, S45, S51, S52, S64, S65, S66, and S84 to adjudge the performance of their prediction model.

3.7.6 RQ6: Performance Comparison of Statistical, Machine Learning and Evolutionary Techniques

We analyzed all kinds of empirical studies carried for software maintainability prediction and compared the prediction accuracies achieved by all the modeling techniques. Due to the

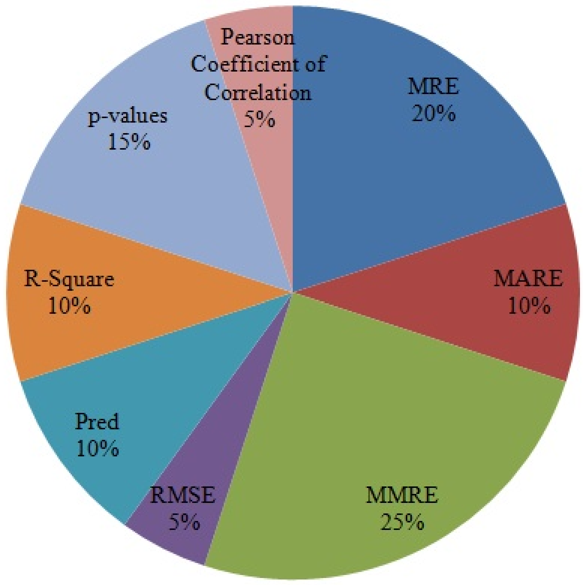

Figure 3.7: Distribution of the use of Accuracy Measures in Literature

diversity in the performance measures considered in different studies, we could not get the clear picture as different studies have considered different measures. MRE and MMRE were used by most of the studies whereas R-value and P-value were used only for once. Machine learning techniques based prediction models were found to be better then statistical techniques based prediction models. Six such studies were found in which MMRE is used as prediction accuracy measure to judge the performances based on ANN as compiled in Malhotra and Chug [137]. In this regard, its values are achieved as 0.59 using ANN [106], 0.403 using BPN [118], 0.265 using ANN [6], 0.23 using PNN [137], 0.765 using GRNN [219] and 0.242 using ANFIS [106]. Overall in each of these six studies, irrespective of the kind of dataset used while making a prediction model, it is empirically proved that machine learning based prediction techniques perform much better than statistical techniques. Use of recently developed nature inspired techniques in maintainability prediction is also found to be very limited. Only one study by Malhotra and Chug [137] has applied GA for software maintainability predictions although these techniques are successfully applied in related areas of maintainability like the prediction of development effort by Balogh et al. [12], prediction of maintenance effort by Basgalupp et al. [17], prediction of preventive maintenance by Sun and Wang [213], identifying software metrics by Vivanco and Pizzi [222].

The machine learning techniques are being successfully applied for making prediction model in many domains other than software engineering such as finance, geography, environmental studies, medicine, engineering, geology and physics. Many machine learning techniques are also explored such as ANN, Bayesian classification, SVM, fuzzy algorithms for predicting quality attributes using OO metrics. However, most of the models present in literature are using statistical methods and very few studies are using machine learning techniques. Utility of many machine learning techniques need to be explored and compare as they give different results. More data based empirical studies that are capable of being verified by observation or experiments are needed. Recently, evolutionary techniques are also explored by many researchers [12, 17, 213, 222] and found to be superior to machine learning techniques due to various aspects listed as under:

- Evolutionary techniques are generally more robust in nature because there are no restrictions on the definition of the objective function.
- Use of evolutionary techniques removes the possibility of biased results.
- Search for an optimized solution is performed in a parallel manner.
- Since the results are only influenced by objective function as well as fitness function, there is no such requirement of auxiliary knowledge.
- They can handle a large amount of noise present in the data as the transition rules are probabilistic in nature not the deterministic in nature.
- They are more capable of working in large and discontinuous search space and able to achieve global optima instead of local ones.
- Evolutionary techniques can provide a number of potential solutions to a given problem and final choice always lies with the user.

3.7.7 RQ7: Effects of Refactoring on Software Maintainability

Refactoring is maintenance process in which the design of an OO software code is improved using various methods without affecting its behavior [69, 77, 158, 163, 170, 212, 226, 228]. It is an effort to improve the quality of the software either by improving the design or by improving readability and understandability while preserving the correctness of the program. Many refactoring methods have been suggested in the literature and each has a particular purpose and corresponding effect. Refactoring is a process in which the internal structure of the OO software system is improved and complexity of the code is reduced however the external behavior of the system remains the same. The source code becomes simpler and easier to maintain as the changes made into the code are very systematic in nature. Few well-known methods of refactoring are dead code elimination, clone code removal, extract method, lazy classes, pull up method, push down method, hide methods, renaming etc. Each method has its own effect on software quality attributes such as extensibility, modularity, reusability, complexity, maintainability and efficiency. It is important and essential to analyze the effects of refactoring on these quality attributes. Many studies have been undertaken where the effect of refactoring has been analyzed on software maintainability [69, 77, 158, 163, 170, 212, 226, 228]. When we analyze the findings of these empirical investigations, it is found that even though refactoring is a very tedious process and might introduce errors if not implemented with utmost care; it is still advisable to refactor the code frequently in order to enhance the maintainability of software. Project managers must take utmost care in identifying the opportunities of refactoring in large code while maintaining a perfect balance between reengineering and over engineering.

3.7.8 RQ8: Various Methods to Measure Maintainability

Many ways have been proposed to access the maintainability as compiled by Berns [22] and Burki & Harald [39] suggested many ways to save the maintenance cost and, in turn, overall project costs. There is a consensus among researchers in this field that there should be some quantified value to measure software maintainability either at the process level, architecture level or at the code level. A range of software maintenance parameters such as Mean Time to Repair (MTTR), Mean Corrective Maintenance Time (MCMT), Mean Preventive Maintenance Time (MPMT), and Maximum Corrective Maintenance Time (MaCMT) are available. As defined in ISO 9126 quality model [95], maintainability consists of external quality attribute i.e. Analyzability, Changeability, Stability, and Testability. Ramil and Smith [181] suggested measuring it in terms of the time taken when the failure was reported to the time taken in repairing it. Jorgensen [101] proposed measuring it in terms of changeability i.e. time taken to implement the changes. Many researchers [53, 62, 106, 118, 137, 219] measured maintainability by measuring the 'Change' i.e. number of lines of source code added, modified or deleted during operations.

3.7.9 RQ9: Advantage of Software Maintainability Prediction

The idea of predicting the maintainability has some inborn problems due to the fact that it is very subjective in nature. Maintainability predictions help us to reduce system's repair time thereby reducing the downtime and increasing system availability as everything can be planned in advance. Certain advantages of maintainability prediction are as follows:

- Managers would be able to compare the productivity and costs among different projects.
- Managers would be able to do more effective planning of the use of valuable resources.
- Managers can take an important decision regarding staff allocation.
- Identify the maintenance process efficiency as it helps in keeping the maintenance cost under control.
- The threshold values of various metrics which drastically affect maintainability of software can be checked and kept under control so as to achieve least maintenance cost.

- Developers can identify the determinants of software quality and hence they can improve the design.
- Practitioners would be able to improve the quality of systems and thus optimize maintenance costs.

3.8 Current Trends and Future Opportunities

Whenever an error occurs in any software, a certain amount of time is needed to correctly identify, isolate and remove the fault. The longer it takes to recover from the occurrence of an error, the higher will be the costs associated with software maintenance. From as early as 1969 to date, the field of software maintenance has evolved over a period of time. Each year many research publications are added to the already available vast amount of knowledge. We have arranged all the qualified studies chronologically and observed the trend. After conducting the assessment of the results obtained in each of the shortlisted studies, we have evaluated them and identified constraints present in order to identify future directions in this field. Based on the results of primary studies, few emerging sub-fields in the field of software maintenance are highlighted as under:

3.8.1 Addition of Dynamic Metrics Along With Static Metrics

In the previous section while finding the answer RQ2, we have discussed various kinds of metric suites proposed, evaluated and validated empirically by many researchers. As evident from the Table 3.6, Chidamber and Kemerer metric suite and Li and Henry metric suite are the most popular one and used by many researchers' in their respective studies. We have observed that unfortunately both the metric suites are actually static in nature. Since some lines of source code might not execute during execution of the software depending upon input supplied and other conditions, hence relating these static design measures to maintainability may not be correct. As Bieman and Ott [26] have measured function cohesion in his study, it would be of great interest to evaluate and analyze the dynamic dependencies between various software artifacts [10, 229]. One of the promising fields is the measurement of design properties of the software using dynamic metrics instead of static metrics such as dynamic LCOM, dynamic RFC etc.

3.8.2 Equal Importance to External and Internal Quality Attributes

Even though Chidamber and Kemerer metric suite [43] is quite popular among researchers, one limitation observed by us is that it takes into account only the internal design metrics while ignoring the importance of external quality attributes such as familiarity with the code, expertise level of programmer, development skills etc. which are semantic in nature. Actually internal design metric suite is as per the specifications of ISO 9126 software quality model, therefore, studying the effect of external quality attributes on maintainability is another promising field which needs to be investigated.

3.8.3 New Metrics for Data Intensive Applications

In earlier times, the data which is stored at the back end might have been accessed a couple of times a week, however with the increase in the use of mobile and mobile based applications, now it is accessed multiple times per hour. As the software systems heavily use databases; hence we observed that Chidamber and Kemerer metric suite would not be adequate as it does not capture the database handling aspects of the applications. Another promising sub-field for research is to explore the significant set of metrics under the new circumstances wherein the applications are highly data intensive along with their respective empirical validations. One such empirical study has been carried by authors Malhotra and Chug [137] in past. In this study, equal attention to the database accesses was given and the new metric suite was empirically proposed and verified to be superior. Analysis of a lot of prevalent metrics to measure system's maintainability, and exploit the possible combination of these metrics into an index for the system's maintainability is still a challenge, yet to be solved.

3.8.4 Use of Hybrid Techniques with More Emphasis on Nature Inspired Techniques

Synthesis of results suggests that initially statistical prediction modeling techniques were in uses which were later on overpowered by machine learning prediction modeling and claimed by the researchers that they better than statistical techniques as they can capture the quality as well as quantity present in the data available for training. In order to achieve more and more accuracy in the prediction, various versions of NN (Neural Network) was used. In this review we have identified the maximum use of NN and seven such studies were found. Regression was used five times whereas Fuzzy Model is used in two studies. We also found here a gap in existing research as many modeling techniques are yet not explored for their prediction capabilities. Ensemble learners (eg bagging, boosting) and instance-based machine learning (eg K- Star) are few methods which have yet not been applied. It is also advised to explore the use of evolutionary techniques for maintainability prediction and combining them with other fields such as data mining, expert systems, GA, artificial intelligence, nature-inspired techniques etc.

3.8.5 Prediction Models for Aspect Oriented Systems

Creating the prediction model(s) that can realistically predict maintainability of applications other than OO system is also yet to be explored. Only two studies were found on relational database-driven software application by Riaz et al. [184] and Schneberger [197]. One study was also found on aspect-oriented systems by Thongmak and Muenchaisri [217]. Hence there is huge demand for creating a generalized prediction model for new systems such as aspect oriented systems, relational database driven systems, component based systems etc.

3.8.6 Use of the Agile Methods and their Effect on Maintainability

The agile software processes [18, 47, 52, 69, 70, 152, 199] such as XP, clean code, scrum, design patterns, Crystal are based on four important pillars: Communication, Simplicity, Feedback, and Courage. Recently a trend is also observed where researchers are taking help of this methodology in keeping maintenance cost constant over time. Although the dataset taken in both the studies Knippers [113] and Poole & Huisman [175] was comparatively small and medium sized, this methodology was proved to be the best for software maintenance work because the human factor is considered as the main component. More empirical investigations on large or very large systems with the help of industry-institute partnership are required to further explore the use of agile methodologies on software maintenance.

3.8.7 Effect of Modern Development Techniques such as Component Based Development on Maintainability

Modern development techniques claim to make the code maintainable such as component-based software development, product line development, model-based development, and design patterns; however, additional research exploration is required to verify such claims. One such study conducted by Mari and Eila [149] concluded that there are different dimensions of software maintainability and each dimension need different treatment. In their study, maintainability is discussed at three levels i.e. abstraction level, architecture level and component level.

3.8.8 Academia-Industry Partnership needs to be Expanded

While finding the answer of RQ4, we found that in the process of empirical investigations, only 16% worked on students projects, 16% on open source, 40% on proprietary software and remaining 28% worked on the data made available in research studies. Out of 40% studies which used proprietary software, 82% used small datasets mostly in academia. Therefore, in order to improve the industrial relevance as well as the validity of the research, it is highly desirable that large sized industrial software must be used in the datasets, their access to the researchers should be provided, more and more industry-academia collaboration in research should be made, and whenever possible the dataset should be made public for carrying out the future research.

3.8.9 More Studies with datasets from Open Source Code Available in Abundance

In continuation with the above point, lots of open source code is available due to the obvious advantage of the internet. Study of open source code, their characteristics and working on the datasets obtained from open source code and further relating it to the maintainability is another promising field while making the prediction models and conducting the empirical studies. Very few studies are found on this field except Zhou and Xu [238] and Malhotra and Chug [142].

3.8.10 Judge Maintainability using Other Quality Measures

In addition to the prediction accuracy measures as described earlier in Table 3.9 to adjudge the quality of the prediction models, other performance measures should also be used to review the prediction quality such as generalization capability, interpretability etc. One of the future directions of research could be carried out in this area wherein overall evaluation of the prediction models can be achieved.

3.8.11 Investigate the Effects of Refactoring on Software Maintainability

Investigating the effects of refactoring on software maintainability is also a very promising field. Many empirical studies have been conducted to prove that local code restructuring process makes the software code more cohesive, less coupled and becomes easier to read and maintain. It is anticipated by the researcher fraternity working in the field of software maintainability that such refactoring operation or software transformations would certainly improve the maintainability of software, however, it is so far undecided that which quality factors are improved by applying shortlisted refactoring methods in specific order. Empirical studies are already being undertaken by many researchers [69, 77, 158, 163, 170, 212, 226, 228] to solve this mystery, still more empirical studies are required to be conducted.

3.8.12 Effect of other Activities such As Risk Analysis, Effort Requirements on Software Maintainability

Most of the empirical research has focused on specific aspects such as programmer productivity and error count which are measured mainly for the short term. It would be an interesting study if undertaken to measure the amount of hours required for maintaining a program developed using agile software development methods when compared to the program developed using a traditional plan-driven approach over a long term, by setting up experiments to specifically test the impact of software development methods on its maintainability. Two such study were conducted by Aggarwal et al. [5] and Prasanth et al. [176] in which effect of risk analysis on maintainability is studied.

3.8.13 Empirical Studies to Identify the Optimum Point when Dropping the Software is more Viable than Developing the New One

For every product whether it is a hardware or software, over a period of time they certainly reach a stage where maintaining them becomes a more costly affair than developing the new one. The researcher fraternity still needs to find the equilibrium point between maintaining the existing software versus scrapping the existing one and developing the new one.

3.8.14 Solve the Mystery involved in Maintainability

Undoubtedly, accessing the maintainability is a bit subjective in nature. The understanding of the intricacies involved in maintainability would certainly assist researchers in the future work. In order to make software architecture and components more maintainable, mysteries of maintainability have to be solved. More studies are required on this field as only one study was found by Broy et al. [37].

3.9 Discussion

In the course of carrying out systematic review of the subject, we performed an exclusive survey and systematic review of the studies published in the field of software maintainability since 1991. Over a period, different researchers have adopted a diverse range of software engineering paradigms and studied its consequences on maintenance cost. New models and innovative techniques have been introduced so that software maintenance prediction could be estimated more accurately. In the current chapter, an effort has been made to review all these models, variables, programming practices etc and identify various important aspects which could greatly affect the maintenance effort.

An extensive search was performed using nine digital libraries after identifying the primary studies in the said field. Overall 179 papers were shortlisted in the initial search out of which only 96 studies were found to be suitable and rest were discarded. These studies were further examined with respect to twelve quality assessment criteria questions and compiled to explore and achieve new insights. Meaningful presentations of the vast collection of the collected data were made using tables and graphs. An attempt has also been made to provide recommendations, constructive guidelines, an overall overview as well as the opportunities and challenges to carry out the future research in the field of software maintainability for researchers and practitioners. This chapter is helpful to the beginners as they can study the concepts of the related area and use the results of this study to identify the complete list of relevant papers in the field. In the end, the survey conducted in this chapter is considered as significant for its contributions as well as timely support to the research community.

Chapter 4

Software Maintainability Prediction using Machine Learning Techniques

4.1 Introduction

Many empirical studies have been conducted in the past to establish the relationship between OO metrics and maintainability using prediction models. The empirical evidence observed in these studies are very powerful support for testing a given hypothesis. However, the relationships between OO metrics and its maintainability is very complex and non linear, hence, conventional statistical techniques based prediction models are not often enough because they are purely based on quantity. Instead, use of machine learning techniques to establish the relationship between metrics and maintainability is much better approach because these techniques are capable of capturing the quantity as well as quality present in the data.

When we checked the accuracy of the models constructed in this study by comparing the predicted results with actual values on 30% of the data which we kept aside initially exclusively for testing, we found the minimum value of MMRE is as low as 0.210 with QUES dataset by applying GMDH technique. Similarly, the value of MMRE is as low as

0.326 with genetic algorithm. It means that machine learning techniques are capable of providing us the accuracy of almost 79% which is quite competitive. Hence, we can say that it can be used as sound alternative for the prediction of maintainability. Many other studies have also empirically examined the strong link between OO software metrics and maintainability such as [53, 62, 106, 118, 137, 237]. Our results are in tandem with the results found in all these studies. Hence, we can say that in general, Chidamber and Kemerer [43] metrics can be used successfully as the predictors of maintenance effort.

Thus, the primary goal of this chapter is to provide empirical evidences using machine learning techniques for investigating and validating OO metrics that capture different aspects of OO design such as coupling, cohesion, inheritance, information hiding and polymorphism and further using them for predicting maintainability. In this regard, the prediction model is constructed using three machine learning techniques i.e. GMDH, GA and PNN with Gaussian activation function. Maintainability is measured by measuring the number of 'change', defined as the number of lines of code which were added, deleted or modified during a three year maintenance period. In order to study and evaluate its performance, two commercial datasets UIMS and QUES written in classical Ada are used. Both, UIMS and QUES are based on OO paradigm and consist of 39 classes and 71 classes respectively. The study is divided into following parts:

1. The descriptive statistics for each of the OO metric is calculated for each class. It helped us to identify low variance metrics which are not useful because of their incapability to differentiate classes.

2. The FSS is used to capture important independent variables for the measurement of given UIMS and QUES system.

3. The relationship between OO design metrics and 'change' for each class is analyzed to empirically determine whether the independent metrics are capable enough to predict the dependent variable.

4. Finally, a prediction model is developed and tested using the three machine learning techniques namely GMDH, GA and PNN.

5. After conducting empirical study, performance of these three proposed machine learning techniques is compared with prevailing models taken from the literature such as GRNN Model, ANN Model, BBN, RT (Regression Tree) Model, Backward Elimination Model, Stepwise Selection Model, Multiple Adaptive Regression Splines (MARS) Model, Tree Nets Model, Generalized Regression Model, Adaptive Neuro Fuzzy Inference System (ANFIS) Model, SVM Model and MLR Model.

6. The comparisons were made using prevalent accuracy measures such as MRE, MMRE, and MARE which indicates that GMDH based prediction model has high accuracy in predicting maintainability.

The chapter is organized as follows: Section 4.2 states research background, the definition of the OO metrics, hypothesis to be tested in this chapter and the data sources. In section 4.3, the results of the chapter are evaluated and compared with outcome present in the literature followed by the validation of hypothesis is presented. Finally, discussion of the chapter is presented in section 4.4.

The results of this chapter have been reported in [137].

4.2 Research Background

In this section, we describe the independent and dependent variables, the hypothesis to be tested, methods used and the results evaluation.

4.2.1 Independent and Dependent Variables

To measure the various features of OO paradigm such as inheritance, cohesion, coupling, memory allocation etc different metrics are carefully selected. We have studied various metrics available in literature and selected only those software metrics that have a strong relationship with software maintainability and used them while constructing our model for prediction of OO software maintainability. In total ten variables were selected as independent variables comprising of five variables from Chidambar and Kemerer Metric Suite [43] namely WMC, DIT, NOC, RFC, and LCOM, five variables are taken from Li and Henry [127] namely MPC, DAC, NOM, SIZE1 and SIZE2. The detail descriptions of these metrics can be found in chapter 2.

The dependent variable is maintenance effort measure by counting the number of lines in the code that were changed during last three year maintenance period per class. A line change could be an addition, deletion or modification.

4.2.2 Hypotheses

Following hypothesis were tested to compare the performance of statistical and machine learning techniques used in this chapter:

4.2.2.1 H1 Hypothesis

- Null Hypothesis: The GMDH model does not outperform the prediction models based on statistical learning methods.

- Alternate Hypothesis: The GMDH model outperforms the prediction models based on statistical learning methods.

4.2.2.2 H2 Hypothesis

- Null Hypothesis: The PNN model does not outperform the prediction models based on statistical learning methods.

- Alternate Hypothesis: The PNN model outperforms the models predicted using statistical learning methods.

4.2.2.3 H3 Hypothesis

- Null Hypothesis: The GA model does not outperform the prediction models based on statistical learning methods.

- Alternate Hypothesis: The GA models outperforms the prediction models based on statistical learning methods.

4.2.3 Group Method of Data Handling

The GMDH model has a forward multi-layer neural network structure. Each layer consists of one or more units wherein two input arcs and one output arc are attached with every unit as shown in equation (4.1), where each unit corresponds to the Ivakhnenko polynomial form.

$$Z = a + bX + cY + dX^2 + eXY + fY^2 \tag{4.1}$$

Where variables x and y are input variables and Z is output variable and a, b, c...f are the parameters.

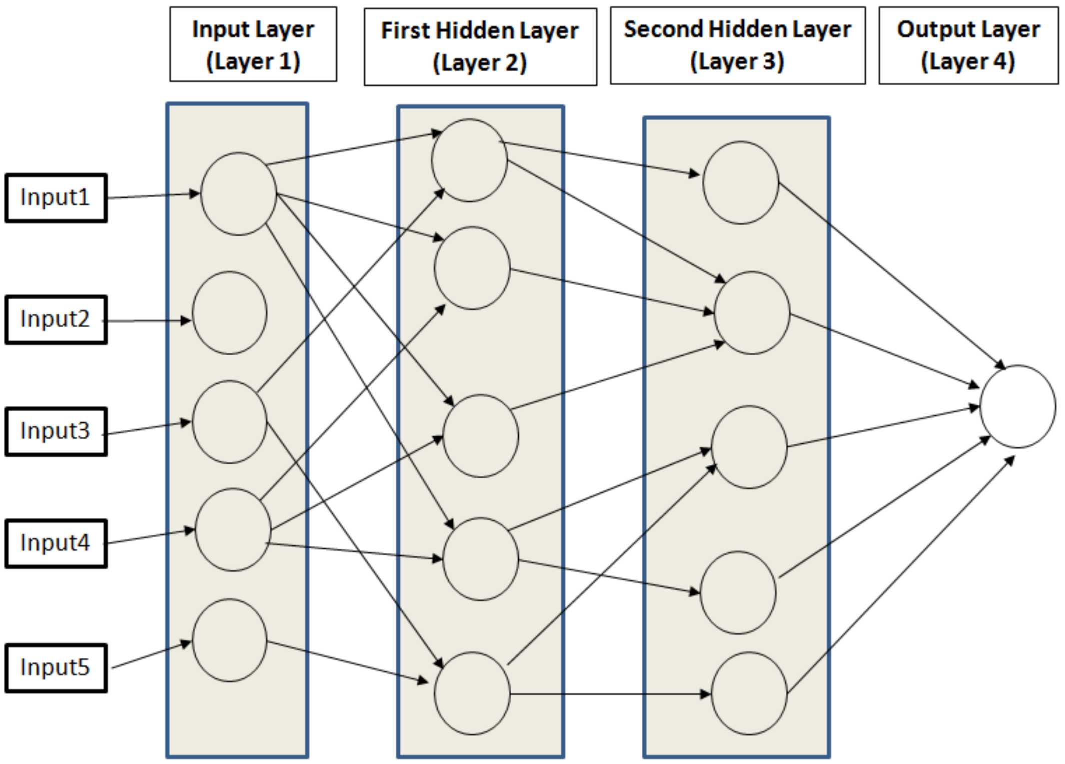

Figure 4.1: Architecture of Group Method of Data Handling Technique

4.2.3.1 GMDH Learning Technique:

As shown in figure 4.1, basic technique of GMDH learning technique is based on self-organization method and it fundamentally consists of the following steps:

1. Given a learning data sample including a dependent variable Y and independent variables X_1, X_2, ... , X_m ; split the sample into a training set and a checking set.

2. Feed the input data of m input variables and generate combination (m, 2) units from every two variable pairs at the first layer.

3. Estimate the weights of all parameters in the formula and apply it on training dataset in the next step using step wise regression method.

4. Compute mean square error between the actual and predicted value of each unit.

5. Sort out the unit by a mean square error in decreasing order and eliminate bad units.

6. Set the prediction of units in the first Layer to new input variables for the next layer, and build up a multilayer structure.

7. When the mean square error become larger than that of the previous layer, stop adding layers and choose the minimum mean square error unit in the highest layer as the final model output.

Input Layer

$$Xi(1) = a_{\mathrm{i}}(1) + b_{\mathrm{i}}(1)X_{\mathrm{m}}(0) + c_{\mathrm{i}}(1)X_{\mathrm{n}}(0) + d_{\mathrm{i}}(1)X_{\mathrm{m}}(0)X_{\mathrm{n}}(0) \tag{4.2}$$

Hidden Layer

$$X_{\mathrm{i}}(k) = a_{\mathrm{i}}(k) + b_{\mathrm{i}}(G)X_{\mathrm{m}}(k-1) + c_{\mathrm{i}}(k)X_{\mathrm{n}}(k-1) + d_{\mathrm{i}}(k)X_{\mathrm{m}}(k-1)X_{\mathrm{n}}(k-1) \tag{4.3}$$

Output Layer

$$X_{\mathrm{i}}(K) = a(G) + b(G)X_1(G-1) + c(G)X_2(G-1) + d(G)X_1(G-1)X_2(G-1) \quad (4.4)$$

4.2.3.2 Advantages of Group Method of Data Handling

It is found that with real problems where noise is present, this model is found to be more accurate. GMDH finds the relationships present in the data and accordingly selects effective input variables automatically. It also automatically determines the parameters, number of layers and number of neuron in each layer present in the structure. GMDH model is found to be most accurate and unbiased models because after each iterations sorting of all variants is performed before switching to the next layer. The only disadvantage with GMDH is that when the numbers of inputs are very large, i.e. more than seven input variables, we have to separate it into several sub-networks with six or fewer input variables since the rule extraction process becomes too complex with many input variables.

4.2.4 Parameters Setup for Prediction Techniques

The main idea behind the first proposed method GMDH is that it tries to build a function called a polynomial model which behave in such a way that the predicted value of the output to be as close as possible to the actual value of the output. Next proposed model is GA which is based on the principles of Darwin's evolution theory. Over many generations, the 'fittest' individuals tend to dominate the population. In predictions based problems, GA tries to discover an optimal solution by simulating the evolution theory. Third model proposed in this chapter is PNN based on neural network technology which mimics the human brain's own problem solving process. As the human beings use their knowledge from earlier experiences to solve new problems or face situations, the neural network also considers earlier solved examples to create a scheme of 'neurons' which makes new choices, classifications, and predictions.

4.2.4.1 Parameters for Group Method of Data Handling

In this section, the parameters of GMDH model are presented for UIMS and QUES datasets. GMDH technique is deployed using Neuroshell2 [81] tool to predict the maintainability of software. We set the parameters as shown in Table 4.1 while applying the proposed models on the selected dataset.

Table 4.1: Parameters Setup for Group Method of Data Handling

S.No.	Parameter	Value
1	Scale Function	[0-1]
2	GMDH Type	Advanced
3	Optimization	Full
4	Maximum Variable	X1, X2, X3
5	Selection Criteria	Regular
6	Missing Value	Considered as Error Condition

Following values of various parameters are received after we finished the process of machine learning using GMDH technique for the given data and comparing the actual values with that of predicted values. Best Formula obtained after applying GMDH machine learning technique is summarized in equation (4.5).

$$Y = 0.34 + 1.8 * {X_2}^2 + 0.68 * {X_{11}}^2 + 17 * X_7 * X_8 - 3.2 * X_2 * X_{10} - 0.76 * X_7 * X_8 * X_{11} \quad (4.5)$$

Where X_1, X_2,..... X_{11} are the parameters estimated by GMDH in terms of OO metrics and their values are given as described in Table 4.2.

4.2.4.2 Parameters for Probabilistic Neural Networks Technique

The PNN are known for their ability to train quickly on sparse datasets. PNN separates data into a specified number of output categories. The network produces activation in the output layer corresponding to the probability density function estimate for that category. The highest output represents the most probable category. In this chapter, the operations of PNN are organized into a multilayered feed forward network with four layers i.e. Input layer,

Table 4.2: Weights Assigned to Each Independent Variable

S.No.	Independent Variable	Assigned Weight
1.	X_1	2*Class-1
2.	X_2	2* DIT/4 -1
3.	X_3	2*NOC -1
4.	X_4	2* (MPC-2)/40 -1
5.	X_5	2* (RFC-17)/139 -1
6.	X_6	2* (LCOM -3)/30-1
7.	X_7	2*DAC/25-1
8.	X_8	2* (WMC-1)/82-1
9.	X_9	2* (NOM-4)/53-1
10.	X_{10}	2* (Size2-4)/78-1
11.	X_{11}	2* (Size1-115)/894-1
12.	Y	Min (Max (change-6)/2.11))

Hidden layer, Pattern layer/Summation layer and an Output layer. As shown in Table 4.3, in the first input layer eleven neurons are presented respectively for each independent variable. The number of neurons in the hidden layer defaults to the number of patterns in the training set because the hidden layer consists of one neuron for each pattern in the training set. Since there are 39 classes for UIMS system, 27 sets are used for training which is approximately 70% share of the total available dataset. Similarly, for QUES system there are 71 classes for UIMS system, 49 sets are used for training. Hidden Layer not only stores the values of the each predictor variables but also stores each neuron along with its target value. In the next pattern layer, one pattern neuron is presented for each category of the output variable. Here remaining 12 datasets for UIMS and 22 datasets for QUES are used for comparing the values. The last layer is output layer where a weighted votes for each target category is compared and accordingly corrections are made.

The smoothing factor that is defined during the design stage is default but we change it in the training sessions in order to make predictions more accurate and precise. We inspected smoothing factor for each link and the same smoothing factor is applied to all links. We have experimented with different smoothing factors to discover which works best for our problem. The success of PNN networks is dependent upon the smoothing factor. There are

Table 4.3: Parameters Setup for Probabilistic Neural Networks

S.No.	Parameter	Value
1	Input Layer	11 in both UIMS system and QUES system
2	Hidden Layer	27 sets for UIMS system and 49 sets for QUES system
3	Pattern Layer	12 sets for UIMS system and 22 sets for QUES system
4	Output Layer	49 sets for UIMS system and 71 sets for QUES system
5	Smoothing Factor	Iterative Calibration method for UIMS system and Genetic Adaptive for QUES system
6	Outliers	Ignored and Considered as Error Condition
7	Probability at Predicted Target	Bayes optimal classification approach

three ways for calibration of PNN networks. The first method is *Iterative Calibration method* which works in two Parts. The first part trains the network with the data in the training set whereas the second part uses a whole range of smoothing factors, trying to hone in on one that works best for the network created in the first part. The second method is *Genetic Adaptive Method* which uses a GA to find appropriate individual smoothing factors for each input as well as an overall smoothing factor. The input smoothing factor is an adjustment used to modify the overall smoothing factor to provide a new value for each input. In the third method known as *'None'* In this calibration technique simply trains the network and we do not find an overall smoothing factor. The value for the smoothing factor is default chosen and applied. The user will have to manually adjust the smoothing factor by entering a new one in the edit box while using this module. Even though PNN are slower and require more memory space, there are several advantages of PNN such as they are much faster, more accurate, and relatively insensitive to outliers, use Bayes optimal classification approach and generate target probability more accurately. Unlike BPN, which require feedback of errors and subsequent adjustment of weights and many presentations of training patterns, training a PNN network is very fast because it requires that each pattern be presented to the network only once during training. During the training session we can see the number of learning events completed during training which is also called as “epoch”. Training can be done in real time as it is almost instantaneous. When data is sparse, training is superior to other network types.

4.2.4.3 Parameters for Genetic Algorithm

The five parameters are needed to be initialized for GA i.e. Generations, Population, Mutation Rate, Mutation percentage and Crossover percentage as summarized in table 4.4 and table 4.5 for UIMS and QUES dataset respectively. Since, time complexity increases drastically if we increase the number of generations; we set the generations at an optimal value of 23 generations for UIMS system and 14 generations for QUES system. Mutation rate also has to be less than 0.5 because more than this value can destroy the solutions. Mutation percentage and crossover percentage also has to be judiciously decided because of the tradeoff between efficiency and optimal value. In empirical study presented in this chapter, since the dataset is comparatively small, the average values of all the parameters are taken.

Table 4.4: Parameters Setup for Genetic Algorithms on UIMS system

S.No.	**Parameter**	**Value**
1	Number of Generations	23 generations for UIMS system
2	Number of population	49 sets for UIMS system
3	Mutation Rate	0.5
4	Mutation percentage on population	50%
5	Crossover percentage on population	50%

Table 4.5: Parameters Setup for Genetic Algorithms on QUES system

S.No.	**Parameter**	**Value**
1	Number of Generations	14 generations for QUES system
2	Number of population	71 sets for QUES system
3	Mutation Rate	0.5
4	Mutation percentage on population	50%
5	Crossover percentage on population	50%

4.2.5 Empirical Data Collection

In this chapter, we used two most popular OO maintainability datasets which are also published by Li and Henry [127] i.e. UIMS and QUES datasets. These datasets were chosen mainly because they have been recently used by many researchers to evaluate the performance of their proposed model in predicting OO software maintainability [53, 62, 106, 118, 237]. Since the main aim of this chapter is to empirically evaluate the predictive capability of machine learning models, the same dataset is selected so that we can compare our results against this published work. The UIMS dataset contains class-level metrics data collected from 39 classes whereas the QUES dataset contains the same metrics collected from 71 classes. Both systems were implemented in Ada and their datasets consist of eleven class-level metrics.

4.3 Results Analysis

In this section, descriptive statistics are collected and analyzed. Experiment setup and values of various parameters initialized for each of the machine learning technique. Further, we have also presented the weights assigned to each of the independent attribute by the GMDH technique. Summarized results for all the three methods are also presented which are used for training and at last results are compared with existing studies followed by their analysis.

4.3.1 Descriptive Analysis

The UIMS and QUES datasets are used in this chapter wherein UIMS dataset contains 39 classes and QUES dataset contains 71 classes. The Descriptive statistics are given in Table 4.6 and Table 4.7 respectively followed by the interpretation.

From the descriptive statistics we noticed some observations as follows:

- In order to measure the inheritance, DIT metrics is used. The median and mean values are minimum in both the system, so we draw the conclusion that the use of inheritance

Table 4.6: Descriptive Statistics of UIMS dataset

S.No.	Metric	Minimum	Maximum	Mean	Std Dev
1.	WMC	0	69	11.38	15.90
2.	DIT	0	4	2.15	0.90
3.	NOC	0	8	0.95	2.01
4.	RFC	2	101	23.21	20.19
5.	LCOM	1	31	7.49	6.11
6.	MPC	1	12	4.33	3.41
7.	DAC	0	21	2.41	4.00
8.	NOM	1	40	11.38	10.21
9.	Size1	4	439	106.44	114.65
10.	Size2	1	61	13.47	13.47
11.	Change	2	253	42.46	61.18

Table 4.7: Descriptive Statistics of QUES dataset

S.No.	Metric	Minimum	Maximum	Mean	Std Dev
1.	WMC	1	83	14.96	17.06
2.	DIT	0	4	1.92	0.53
3.	NOC	0	0	08	0
4.	RFC	17	156	54.44	32.62
5.	LCOM	3	33	9.18	7.31
6.	MPC	2	42	17.75	8.33
7.	DAC	0	25	3.44	3.91
8.	NOM	4	57	13.41	12.00
9.	Size1	115	1009	275.58	171.60
10.	Size2	4	82	18.03	15.21
11.	Change	6	217	64.23	43.13

in both systems is limited.

- The values for median and mean for CHANGE (dependent variable) in the UIMS dataset is lesser than those in the QUES, which means UIMS seems to be more maintainable.

- We removed NOC from the QUES dataset because it is observed that all data points for NOC are zeros in the QUES dataset.

- We had observed that the coupling between classes in QUES is higher than those in the UIMS because the medians and means values for RFC and MPC in the QUES dataset were larger than UIMS dataset.

- Values of mean and median of LCOM were almost same in both systems that mean both have almost similar cohesion.

- The similar medians and means for NOM and SIZE2 in both datasets suggest that both systems had similar class sizes at the design level, however there is a significant difference in SIZE1.

4.3.2 Evaluation of Results

Studies examining the link between OO software metrics and maintainability have found that in general these metrics can be used as predictors of maintenance effort [53, 62, 106, 118, 137, 237]. Although a number of maintainability prediction models have been developed in last two decades, they have low prediction accuracies according to the criteria suggested by Conte et al. [50]. Therefore, it is necessary to explore new techniques, which are not only easy in use, but also provide high prediction accuracy for the purpose of maintainability prediction.

In the empirical study presented in this chapter, we have sought to build OO software maintainability prediction model using three machine learning techniques i.e. GMDH, GA and PNN using Gaussian Activation function. Although ANN has been used previously in literature [6, 106, 219] but for the first time the Probabilistic Neural Network (PNN) along with gaussian activation function has been applied. The GMDH and GA are also proposed for the first time for prediction of the software maintainability. In this chapter, to draw most realistic comparison we have also analyzed the same dataset which was originally proposed by Li and Henry and earlier applied by various researchers to predict maintainability. For analyzing the results of three proposed machine learning techniques in this chapter, MRE, MMRE and Max MRE values are considered. Table 4.8 and Table 4.9 summarize their values for UIMS and QUES datasets respectively.

Table 4.8: Prediction Accuracy Measures for Various Techniques on UIMS dataset

S.No.	Model Name	Max MRE	MMRE	Pred (0.25)	Pred (0.75)
1	GMDH Model	0.883	0.432	0.72	0.821
2	Genetic Model	0.787	0.326	0.67	0.782
3	PNN Model	0.898	0.397	0.7	0.894

Table 4.9: Prediction Accuracy Measures for Various Techniques on QUES dataset

S.No.	Model Name	Max MRE	MMRE	Pred (0.25)	Pred (0.75)
1	GMDH Model	0.983	0.210	0.69	0.944
2	Genetic Model	0.794	0.220	0.66	0.972
3	PNN Model	0.923	0.230	0.68	0.944

Results of the preliminary analyses are presented in Table 4.8 and Table 4.9 for UIMS and QUES dataset respectively. Minimum value of MMRE received is 0.210 (GMDH) for QUES dataset and 0.326 (GA) for UIMS dataset. These values shows high confidence that GMDH as well as GA can be used as a sound alternative for the prediction of maintainability. The criteria for prediction given by Conte et al. [50] states that prediction model is considered accurate if the value of pred(0.25) is greater than pred(0.75) which clearly proposed models in this chapter satisfies. In the literature, it is also suggested that prediction accuracy of software maintenance effort prediction models is often low and thus, it is very difficult to satisfy the criteria [62]. It can also be noticed from Tables that the GMDH model has achieved improved Pred(0.25) and Pred (0.75) over the other models in QUES and UIMS datasets, and its results are quite closer to the criteria set in literature.

4.3.3 Comparison with Existing Studies from Literature

We also have compared the values of prediction accuracy measures of certain selected parameters with the studies conducted in the last decade. In Table 4.10 and 4.11 we presented the summarized performance measures of proposed models and the models whose results are available in the literature on UIMS dataset and QUES dataset respectively.

From the Table 4.10 and 4.11, it can be observed that out of the eighteen techniques,

Table 4.10: Results of Various Prediction Accuracy Measures on UIMS dataset

S.No	Model Name	Max MRE	MMRE	MARE	Pred (0.25)	Pred (0.30)	Pred (0.75)	R-Square	p-value
1.	GMDH	0.883	0.432	-	0.72	0.77	0.821	-	-
2.	Genetic Model	0.787	0.326	-	0.67	0.72	0.782	-	-
3.	PNN	0.898	0.397	-	0.7	0.75	0.894	-	-
4.	GRNN [219]	-	-	-	-	-	-	-	-
5.	ANN [4]	-	0.265	-	-	-	-	0.582	0.004
6.	Bayesian Belief Model [118]	7.039	0.972	-	0.446	0.469	-	-	-
7.	Regression Tree [118]	9.056	1.538	-	0.200	0.208 -	-	-	-
8.	Backward Elimination [118]	11.890	2.586	-	0.215	0.223	-	-	-
9.	Stepwise Selection [118]	12.631	2.473	-	0.177	0.215	-	-	-
10.	MARS [237]	14.06	1.86	-	0.28	0.28	-	-	-
11.	MLR [237]	18.88	2.70	-	0.15	0.21	-	-	-
12.	SVM [237]	9.13	1.68	-	0.31	0.36	-	-	-
13.	ANN [237]	19.63	1.95	-	0.15	0.15	-	-	-
14.	Regression Tree [237]	24.57	4.95	-	0.10	0.10	-	-	-
15.	TreeNets [62]	-	1.57	-	0.31	0.41	-	-	-
16.	Generalized Regression [106]	-	-	0.308	-	-	-	-	-
17.	ANFIS [106]	-	-	0.242	-	-	-	-	-
18.	Linear Regression [127]	-	-	-	-	-	-	0.9096	-

the GMDH model, Genetic model and PNN model gives very competitive results and hence show their worth that they can be used in the process of software maintainability prediction. It is also evident from the Table 4.10 and 4.11, that prediction accuracy of GMDH network model is much better than other models. It is the only model which is close to the criterion laid by Conte et al. [50]. At pred(0.25) its values are 0.69 which means that almost 69% predictions are less than the error of 0.25 prediction accuracy. At pred(0.30) its value is 0.722 which means that almost 72% predictions are less than the error of 0.30 prediction accuracy as compared to other models. Following comparative analysis, it is safe to conclude that GMDH has clearly outperformed than other models. The GMDH models can predict

Table 4.11: Results of Various Prediction Accuracy Measures on QUES dataset

S.No.	Model Name	Max MRE	MMRE	MARE	Pred (0.25)	Pred (0.30)	Pred (0.75)	R-Square	p-value
1.	GMDH	0.983	0.210	-	0.69	0.722	0.944	-	-
2.	Genetic Model	0.794	0.220	-	0.66	0.722	0.972	-	-
3.	PNN	0.923	0.230	-	0.68	0.75	0.944	-	-
4.	GRNN [219]	4.295	0.765	-	-	-	-	-	-
5.	ANN [4]	-	0.265	-	-	-	-	0.582	0.004
6.	Bayesian Belief Model [118]	1.592	0.452	-	0.391	0.430	-	-	-
7.	Regression Tree [118]	2.104	0.493	-	0.352	0.383 -	-	-	-
8.	Backward Elimination [118]	1.418	0.403	-	0.396	0.461	-	-	-
9.	Stepwise Selection [118]	1.471	0.392	-	0.422	0.500	-	-	-
10.	MARS [237]	1.91	0.32	-	0.48	0.59	-	-	-
11.	MLR [237]	2.03	0.42	-	0.37	0.41	-	-	-
12.	SVM [237]	2.07	0.43	-	0.34	0.46	-	-	-
13.	ANN [237]	3.07	0.59	-	0.37	0.45	-	-	-
14.	Regression Tree [237]	4.82	0.58	-	0.41	0.45	-	-	-
15.	TreeNets [62]	-	0.42	-	0.58	0.65	-	-	-
16.	Generalized Regression [106]	-	-	0.308	-	-	-	-	-
17.	ANFIS [106]	-	-	0.242	-	-	-	-	-
18.	Linear Regression [127]	-	-	-	-	-	-	0.8737	-

maintainability of the OO software systems with least MMRE when compared with others models such as GRNN, ANN, BBN, MARS, TreeNets and SVM for QUES dataset. Hence it is clear inference that GMDH is the most accurate and best model for the predictions of software maintainability.

The results of the validation of GRNN model studies by Thwin and Quah [219] for quality prediction is not available for UIMS dataset as it was conducted only on QUES dataset. The SVM model was proposed recently by Jin and Liu [100] for predicting maintainability using OO metrics, however it is not comparable to the current empirical study because of the fact that their study was merely conducted on the code which was written for ‘Temper proof

HTML web page' in C++ whereas our study is conducted on commercially available QUES dataset written in ADA with much higher scope. Not only the sizes of the software differ to a large degree but also both systems varied in to great extent with respect to their paradigm and complexity. Secondly, Max MRE and Pred(q) were not provided despite being de facto prediction standards. MARE in their model recorded as 0.218. When it is compared with the current study, MMRE has been recorded better at 0.210 with GMDH model that clearly confirms higher competence even in a complex environment.

4.3.4 Validation of Hypotheses

In this section, we validate our hypothesis stated in section 4.2.2.

4.3.4.1 H1 Hypothesis

We make the first assumption that the GMDH model do not outperform the models predicted using statistical learning methods. In order to validate this hypothesis, we compared the MMRE values of all models and represented them graphically. As shown in figure 4.2, MRE values for GMDH model is found to be minimum with both the system i.e. UIMS dataset as well as for QUES dataset.

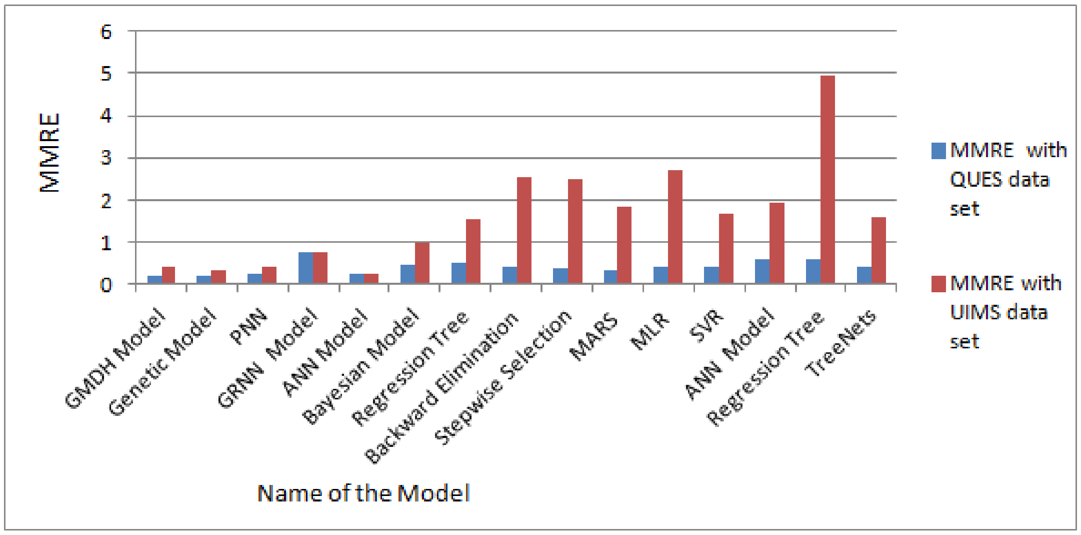

Figure 4.2: Comparison of Various Models with Reference to their MMRE values

It is observed that the value of minimum MMRE with GMDH model is 0.21 which means 79% accuracy is achieved. In order to verify whether these results are significant and not coincidental, Wilcoxon significance test is also conducted. The level of significance is revised from 0.05 to (0.05/14) due to the Benfeorrani adjustment discussed in Section 2.13.3 from equation 2.18. The value of α is divided by fifteen because there are fifteen techniques whose MMRE values are available for UIMS and QUES dataset as shown in Table4.10 and 4.11. Wilcoxon tests are conducted to compare GMDH technique with other fifteen techniques because it is ranked most accurate in UIMS and QUES dataset. Table 4.12 shows the results of Wilcoxon test for UIMS and QUES dataset. Since, the value of revised significance level α is 0.05/15=0.003, hence, the difference value at Z significance is compared with 0.003.

Table 4.12: Results of Wilcoxon Test for MMRE Measure (paired with GMDH Model

S.No.	**Model Name**	**Z Significance**
1.	GMDH - Genetic Model	2.972 (0.001)
2.	GMDH - PNN	3.472 (0.025)
3.	GMDH - GRNN	2.481 (0.002)
4.	GMDH - ANN	2.258 (0.008)
5.	GMDH - BBN	2.48 (0.003)
6.	GMDH - Regression Tree	3.152 (0.002)
7.	GMDH - Backward Elimination	4.466 (0.000)
8.	GMDH - Stepwise Selection	3.553 (0.038)
9.	GMDH - MARS	1.834 (0.001)
10.	GMDH - MLR	1.891 (0.002)
11.	GMDH - SVM	1.232 (0.025)
12.	GMDH - ANN	3.502 (0.047)
13.	GMDH - Regression Tree	3.672 (0.002)
14.	GMDH - TreeNets	4.909 (0.026)

Wilcoxon test for UIMS and QUES dataset is compiled in Table 4.12 with all techniques are paired with GMDH technique as it is ranked most accurate technique in terms of MMRE values. However, the results of pair wise Wilcoxon Test for MMRE indicate that GMDH is not significantly better from all other fourteen techniques. Instead, GMDH is significantly superior only to eight out of fourteen techniques. Hence, we conclude that the GMDH models outperform the models predicted using statistical learning methods.

4.3.4.2 H2 Hypothesis

The second hypothesis assumes that GA does not outperform the models based on statistical method. From the figure 4.2 it is evident that GA model is also very competitive in terms of MMRE values. Its values are 0.33 with UIMS dataset and 0.22 with QUES dataset which means almost 67% and 78% accuracy is achieved. As per the criterion set by Conte et al. [50] the results are very competitive. Wilcoxon test is also conducted and the results are summarized in Table 4.13 in order to investigate if the results are not coincidental.

Table 4.13: Wilcoxon Test for MMRE Measure (Paired with Genetic Algorithm Model)

S.No.	**Model Name**	**Z Significance**
1.	GA - Genetic Model	2.972 (0.001)
2.	GA - PNN	3.472 (0.025)
3.	GA - GRNN	2.481 (0.002)
4.	GA - ANN	2.258 (0.008)
5.	GA - BBN	2.48 (0.003)
6.	GA - Regression Tree	3.152 (0.002)
7.	GA - Backward Elimination	4.466 (0.000)
8.	GA - Stepwise Selection	3.553 (0.038)
9.	GA - MARS	1.834 (0.001)
10.	GA - MLR	1.891 (0.002)
11.	GA - SVM	1.232 (0.025)
12.	GA - ANN	3.502 (0.047)
13.	GA - Regression Tree	3.672 (0.002)
14.	GA - TreeNets	4.909 (0.026)

Wilcoxon test for UIMS and QUES dataset is compiled in Table 4.13 with all techniques are paired with GA. The value of revised significance level α is 0.05/14=0.003. So the difference value at Z significance is compared with 0.003. However, the results of pair wise Wilcoxon Test for MMRE indicate that GA is not significantly better from all other fourteen techniques. Instead, it is significantly superior only to four out of fourteen techniques. Hence, we conclude that the GA model does not outperform the models predicted using statistical learning methods.

4.3.4.3 H3 Hypothesis

The third hypothesis assumes that PNN model does not outperform the models based on statistical method. From the figure 4.2, it is evident that PNN model is also very competitive in terms of MMRE values. Its values are 0.397 with UIMS dataset and 0.230 with QUES dataset which means almost 60% and 77% accuracy is achieved. As per the criterion set by Conte et al. [50] the results are very competitive. Wilcoxon test is also conducted to confirm that the results are not coincidental and results are summarized in Table 4.14.

Table 4.14: Wilcoxon Test for MMRE Measure (Paired with PNN Model)

S.No.	**Model Name**	**Z Significance**
1.	PNN - Genetic Model	2.972 (0.001)
2.	PNN - PNN	3.472 (0.025)
3.	PNN - GRNN	2.481 (0.002)
4.	PNN - ANN	2.258 (0.008)
5.	PNN - BBN	2.48 (0.003)
6.	PNN - Regression Tree	3.152 (0.002)
7.	PNN - Backward Elimination	4.466 (0.000)
8.	PNN - Stepwise Selection	3.553 (0.038)
9.	PNN - MARS	1.834 (0.001)
10.	PNN - MLR	1.891 (0.002)
11.	PNN - SVM	1.232 (0.025)
12.	PNN - ANN	3.502 (0.047)
13.	PNN - Regression Tree	3.672 (0.002)
14.	PNN - TreeNets	4.909 (0.026)

Wilcoxon test for UIMS and QUES dataset is compiled in Table 4.14 with all techniques are paired with PNN technique. Value of revised significance level α is 0.05/14=0.003. So the difference value at Z significance is compared with 0.003. However, the results of pair wise Wilcoxon Test for MMRE indicate that PNN is not significantly better from all other fourteen techniques. Instead, it is significantly superior only to four out of fourteen techniques. Hence, we conclude that the PNN model does not outperform the models predicted using statistical learning methods.

4.4 Discussion

Three different machine learning techniques GMDH technique, GA technique and PNN technique are used for the purpose of prediction of software maintainability. Even though many studies reported the wide application of GMDH model and GA model in diverse fields for the purpose of prediction of high order input output relationship which is complex, non-linear and unstructured but for the first time they are used for predicting software maintainability. The results of this work are summarized as follows:

1. Minimum value of MMRE is received as 0.210 with QUES dataset when GMDH technique is applied which means 79% accuracy is achieved. Hence, we can say that it can be used as sound alternative for the prediction of maintainability.

2. For UIMS dataset, minimum value of MMRE is 0.326 with GA, hence, it also satisfies the criteria laid out by Conte et al. [50].

3. When the results are analyzed using Pred(0.25), its values are 0.69 which means that almost 69% predictions are less than the error of 0.25 prediction accuracy with GMDH technique.

4. Pred(0.30) is also used to judge the performances of machine learning techniques and we found that its best value is 0.722 with GMDH technique which means that almost 72% predictions are less than the error of 0.30 prediction accuracy as compared to other models. Hence, it is safe to conclude that GMDH outperformed than other models.

5. In order to verify whether these results are significant and not coincidental, Wilcoxon significance test is also conducted and performed with fifteen machine learning techniques whose values were available in literature. The level of significance is revised from 0.05 to (0.05/14= 0.003) due to the Benfeorrani adjustment.

6. Results show GMDH is significantly superior only to eight techniques out of fourteen techniques whose MMRE values were available.

7. The results of pair wise Wilcoxon Test for MMRE indicate that GA is not significantly better from all other fourteen techniques. Instead, it is significantly superior only to four out of fourteen techniques

8. The results of pairwise Wilcoxon Test for MMRE indicate that PNN is not significantly better from all other fourteen techniques. Instead, it is significantly superior only to four out of fourteen techniques.

Chapter 5

A Metric Suite for Predicting Software Maintainability in Data Intensive Applications

5.1 Introduction

Many empirical studies have conducted [6, 62, 106, 118, 137, 237] to validate that the prediction of OO software maintainability can be achieved before actual operation of the software using OO design metrics proposed by Chidamber and Kemerer [43]. However, the framework and reference architecture in which the software systems are being currently developed have changed dramatically in recent times due to the emergence of data warehouse and data mining field. All six metrics defined by Chidamber and Kemerer [43] measures the object oriented properties such as inheritance, coupling and cohesion. None of the metric measures the amount of data handling by an application. In the prevailing scenario, certain deficiencies were discovered when Chidamber and Kemerer metric suite is evaluated for the prediction of maintainability of data intensive software systems. Firstly, we take into the considerations, the major observation given by Li and Henry [127] that Chidamber and Kemerer metric suite does not measure the structural complexity of the code. Hence, in this study apart from using Chidamber and Kemerer metric suite, we have also added MI metric [168] and CC metric to measure the structural complexity. Secondly, we observed that as there is surge in the use of database technology nowadays, it's very important to give equal attention to the understandability of database. We proposed two new metrics in this regard to overcome this aspect. First metric is Number of Data Base Connections (NODBC) which counts the connections made by the application each time for query processing. Second metric measures the understandability of the databases by comparing the ratio of the documentation to the number of fields in the schema and it is defined as Schema Complexity to Comment Ratio (SCCR).

In this chapter, we propose a new metric suite and redefine the relationship between design metrics with maintainability for data intensive applications. The proposed metric suite is evaluated, analyzed and empirically validated using five proprietary software systems.

The study is divided into following parts:

1. The data was collected from five proprietary software systems developed in Microsoft Visual Studio using C# language and based on OO methodologies with heavy use of databases for processing of each query. They are operational in real time scenario since last three years in the industry.

2. Two new metrics were proposed and validated for the data intensive applications.

3. The descriptive statistics for each of the OO metric is calculated for each class. It helped us to identify low variance metrics which are not useful because of their incapability to differentiate classes.

4. The uni-variate and multivariate linear regression are used to capture the relationship between newly proposed metric with dependent variable on the basis of five real-life datasets.

5. The relationship between OO design metrics and ‘change’ for each class is analyzed to empirically determine whether the independent metrics are capable enough to predict the dependent variable.

6. Four different versions of ANN i.e. BPN, Kohonen Network, FFNN and GRNN are used for making the prediction model along with GMDH for proprietary systems.

7. The performance of GMDH was compared with prevailing models using prevalent accuracy measures such as MRE, MMRE and MARE.

The chapter is organized as follows: Section 5.2 states research background, the definition of the OO metrics, data sources, hypothesis for testing and the parameter setup for crating machine learning based prediction model. In section 5.3, the results of the study are evaluated and compared with existing studies and the hypotheses are validated. Discussion of the study is presented in section 5.4.

The results of this chapter have been reported in [136, 138, 140, 141].

5.2 Research Method

In this section we summarize the independent and dependent variables, new proposed metrics, details of the empirical data and hypothesis.

5.2.1 Independent and Dependent Variables

Various metrics have been proposed in the literature which has significant impact on software maintainability. The main purpose of this empirical study is twofold, firstly to review the role of Chidamber and Kemerer metric suite for the prediction of software maintainability and secondly to propose a new suite of metrics with the induction of two new metrics which have a larger impact on maintainability in highly data intensive applications.

To measure the features of OO paradigm, Chidamber and Kemerer metric suite [43] is found to be a significant indicator of maintainability predictions in a large number of studies

[6, 62, 106, 118, 137, 237]. We rely on the outcome of these studies and use Chidamber and Kemerer metric suite to capture the OO characteristics. Six variables were selected as independent variables from Chidambar and Kemerer Metric Suite [43] namely WMC, DIT, NOC, CBO, RFC and LCOM.

5.2.2 Proposed Metrics

There were two deficiencies found in Chidambar and Kemerer Metric Suite [43]. The first observation was the same as noted by Li and Henry [127] that it does not take into account the structural complexity of the software. To overcome this deficiency we added two metrics as summarized in Table 5.1. First is MI proposed by Oman and Hagemeister [168] and second is CC proposed by McCabe [156]. Both the metrics are defined in chapter 2.

Table 5.1: Metrics to Capture the Structural Complexity

S.No.	Metric Name	Description
1.	Cyclomatic Complexity (CC)	It is used to measure the complexity of the program in terms of linearly independent paths in the given source code. If the number of independent paths cross certain thresh-hold limit, that means system/ class is more complex; hence, it need more testing and maintenance efforts.
2.	Maintainability Index (MI)	It is calculated as a factored formula dependent upon LOC, CC and Halstead volume as discussed in equation (3.1) in chapter 3, which in turn dependent upon number of operators and operands.

Table 5.2: Proposed New Metrics for Data Intensive Applications

S.No.	Metric Name	Description
1.	Scheme Complexity to Comment Ratio (SCCR)	It calculate the ratio of number of comments lines to number of field in the schema of data base.
2.	Number of Data Base Connections (NODBC)	Number of Data Base Connection is a measure to count number of times database connections were made.

The second and main deficiency found in the metric suite is on account of the amount of database handling. To overcome this deficiency, two new metrics were proposed and validated for the applications which heavily use databases. The proposed metrics are Number of Data Base Connections (NODBC) made each time for query processing and the Schema Complexity to Comment Ratio (SCCR) to measure the understandability of the databases because it's important to give equal attention to the database accesses with the enhancement in data base usage nowadays. With the increase in the use of mobile and mobile based applications, data that once might have been accessed a couple of times a week now might be accessed multiple times per hour. As the software systems heavily use data bases; hence we observed that Chidamber and Kemerer metric suite would not be adequate as it does not capture the database handling aspects of the applications. We proposed two more metrics as summarized in table 5.2, to remove these deficiencies and we claim that two proposed metrics carries more impact on software maintainability in database intensive applications. NODBC is measured by counting the number of times database connections were made using the function call 'Open( )'. To count the SCCR, ratio of the numbers of the field in the schema to the number of comment lines was considered. We are of the strong opinion that understandability of the schema of the database is equally important in determining the maintaining any application.

Overall a set of ten metrics were considered as independent variables in this study including six from Chidamber and Kemerer metric suite and four metrics defined in table 5.1 and 5.2 i.e. CC, MI, NODBC and SCCR.

5.2.3 Dependent Variable

The dependent variable was the number of the changes in the lines of source code to measure the amount of maintenance efforts required by each class. Two versions of each of the software systems were taken and analyzed to count the changes made in the new version with respect to the older version as described in chapter 2.

5.2.4 Hypotheses

Following hypothesis were tested to investigate the performance of new proposed metric suite in this study:

5.2.4.1 H1 Hypothesis

- Null Hypothesis: The new proposed metric suite does not outperform the prediction models based on Chidamber and Kemerer metric suite.

- Alternate Hypothesis: The new proposed metric suite outperforms the prediction models based on Chidamber and Kemerer metric suite.

5.2.4.2 H2 Hypothesis

- Null Hypothesis: The GMDH model does not outperform the prediction models based on ANN based techniques such as BPN, Kohonen Network, FFNN and GRNN.

- Alternate Hypothesis: The GMDH model outperforms the prediction models based on ANN based techniques such as BPN, Kohonen Network, FFNN and GRNN.

5.2.5 Parameters Setup for Prediction Techniques

In this section, we explain the various machine learning methods used for making the prediction models as well as to ascertain the relationship of design metrics with maintainability. In our previous studies [6, 136, 137, 141], we found that ANN is very powerful in classifying and recognizing the data patterns, so they are well suited for prediction problems as in such cases although the required knowledge is difficult to specify but enough data for observations are available to learn. Hence in this study, four different versions of ANN models i.e. BPN, Kohonen Network, FFNN, and GRNN along with one more machine learning model GMDH as discussed in chapter 4 are used. The brief description of parameters setup of all these models is discussed as below.

5.2.5.1 Back Propagation Network

During the training process parameters as shown in Table 5.3 are used in the prediction model. In total, the numbers of input units are 7 and hidden units are 15. If the difference in the values of actual and predicted is found then the weights of the network are readjusted to reduce this error and the process is repeated until the desired output is produced.

Table 5.3: Parameters Setup for Back Propagation Network

S.No.	**Parameter**	**Corresponding Value**
1.	Architecture	Back Propagation Network
2.	Layers	3
3.	Input Units	7
4.	Hidden Units	15
5.	Output Units	1
6.	Training Transfer Function	Tansig
7.	Technique Used	Back Propagation
8.	Training Function	BR

5.2.5.2 Kohonen Network

Although, Kohonen Network is invented to provide a way of representing multidimensional data in much lower dimensional spaces, a network is created that learn the information such that any topological relationships within the training set are maintained without supervision. Kohonen feature maps are used for classifications purpose which are nothing but the extension of learning vector quantification. During the training process parameters as shown in Table 5.4 are used while creating the prediction model based on Kohonen Network. When an input pattern is available for learning in Kohonen network, the closest neuron in the competition layer is determined and called as winner neuron. It becomes the focal point of the weight changes. Another attribute 'Number of Iteration' parameter defines how many iterations the technique will execute. Neighborhood means the initial size of the neighborhood of best matching which decreases with every iteration while running the algorithm every time. If this parameter is set to 0, then the neighborhood will contain only the winner neuron. If it is set to 1, then the network will contain neurons which are directly connected with the winner neuron. Two shapes are available for neighborhood i.e. square and hexagonal and in this study later one is selected. Alpha determines how much the neighborhood of the best matching unit get affected and it is set to 0.5. The value of K which is also called cooling determines how fast the fill size of the neighborhood and the alpha parameter decrease and it is set to 0.2.

Table 5.4: Parameters Setup for Kohonen Network

S.No.	Parameter	Corresponding Value
1.	Architecture	Kohonen Network
2.	Number of Iterations	200
3.	Neighborhood	3
4.	Shape of Neighborhood	Hexagonal
5.	Alpha (to determine the size of the neighborhood affected)	0.5
6.	Value of K (Cooling) to determine speed of the fill size of neighborhood	0.2
7.	Algorithm	Kohonen Propagation

5.2.5.3 Feed Forward Neural Network

FFNN are the first and simplest type of ANN and as the name implies they do not have any cycle or loops in the network. Information moves in forward direction only from the input unit to the output unit. In the current study, as shown in Table 5.5, the number of hidden neuron selected as 10 for the sample data collected from these five real life applications. The approach we follow is that the weights of hidden layer are chosen randomly and the output layer is trained by single layer learning rule using the pseudo-learning technique.

Table 5.5: Parameters Setup for Feed Forward Neural Network

S.No.	Parameter	Corresponding Value
1.	Architecture	Feed Forward Neural Network
2.	Number of Hidden Layers	10
3.	Learning technique	Sigmoid Non Linear
4.	Weights	Random
5.	Output layer Learning	Pseudo-inverse Techniques

5.2.5.4 General Regression Neural Networks

Since it performs well even in the case of noisy and sparse data and the over-fitting problem does not arise as neither do they set the training parameters during the commencement of learning process, nor they define the momentum. Once the network finished the training process, the only smoothing factor is applied to determine how tightly the network matches its prediction.

Table 5.6: Parameters Setup for General Regression Neural Network

S.No.	Parameter	Corresponding Value
1.	Architecture	General Regression Neural Network
2.	Density Function	Uni-variate Probability Estimator
3.	Gradient of Regression Surface	Numerical Approximation
4.	Kernel Function	Symmetric
5.	Smoothness Function	Standard Deviation

5.2.5.5 Group Method of Data Handling

GMDH technique was deployed using Neuroshell2 [81] tool to predict the maintainability of software. We set the parameters as shown in Table 5.7 while applying the proposed models on the selected dataset.

Table 5.7: Parameters Setup for Group Method of Data Handling

S.No.	Parameter	Corresponding Value
1	Scale Function	[0-1]
2	GMDH Type	Simple
3	Optimization	Maximum
4	Maximum Network Layer	20
5	Maximum Polynomial Order	12
6	Missing Value	Data Point Not Considered
7	Convergence to Learn	1.000e-04
8	Network Layer Connections	Previous Layer and Original Input Variables

5.2.6 Empirical Data Collection

In this chapter, five proprietary systems are used for the validation of new metric suite namely FLM system, EASY system, SMS system, IM system and ABP system as described in chapter 2. They consist of 233, 292, 129, 96 and 114 classes respectively. Descriptive statistics such as Max, Min, Mean, and Median (Med) and Standard Deviation (Std Dev) were calculated for FLM system, EASY system, SMS system, IM system and ABP system and presented in Table 5.8 to 5.12 respectively.

Table 5.8: Descriptive Statistics of FLM System

S.No.	Metric	Max	Min	Mean	Median	Std Dev
1.	WMC	16	1	6.276	5	4.97
2.	DIT	7	1	4.379	5	1.32
3.	NOC	7	0	3.1	3	1.67
4.	CBO	50	3	26.14	30	13.85
5.	RFC	67	12	25.16	18	7.89
6.	LCOM	0	0	0	0	0
7.	SCCR	5	2	3.276	3	2.97
8.	NODBC	12	0	2.483	0	3.53
9.	MI	91	40	61.14	56	18.04
10.	CC	29	1	19.31	16	13.76
11.	Change	95	5	41.98	67	45.67

Table 5.9: Descriptive Statistics of EASY System

S.No.	Metric	Max	Min	Mean	Median	Std Dev
1.	WMC	23	1	10.5	9.5	8.57
2.	DIT	5	1	3.6	4	2.50
3.	NOC	8	0	4.23	3	2.91
4.	CBO	54	0	33.73	38.5	21.58
5.	RFC	78	21	37.73	27	4.89
6.	LCOM	0	0	0	0	0
7.	SCCR	7	3	4.57	5	5.57
8.	NODBC	7	0	2.79	0.5	3.43
9.	MI	94	43	64.14	56.5	17.91
10.	CC	22	1	20.6	19	14.26
11.	Change	87	9	52.52	63	43.23

From the tables Table 5.8 to 5.12 following observations are made:

Table 5.10: Descriptive Statistics of SMS System

S.No.	Metric	Max	Min	Mean	Median	Std Dev
1.	WMC	29	2	16.63	17.5	9.17
2.	DIT	6	1	3.25	4	2.12
3.	NOC	11	0	4.85	4	2.67
4.	CBO	59	3	45.38	52.5	18.66
5.	RFC	83	19	37.09	31	5.87
6.	LCOM	0	0	0	0	0
7.	SCCR	6	2	4.625	16.5	9.17
8.	NODBC	6	0	3.89	3	2.50
9.	MI	81	49	55.25	52	10.56
10.	CC	27	1	21.50	19.5	19.61
11.	Change	79	13	67.89	47	32.43

Table 5.11: Descriptive Statistics of IMS System

S.No.	Metric	Max	Min	Mean	Median	Std Dev
1.	WMC	12	0	3.147	3	2.57
2.	DIT	5	4	4.029	4	0.17
3.	NOC	7	0	2.81	3	1.91
4.	CBO	30	2	13	13.5	8.09
5.	RFC	43	18	21.09	27	5.07
6.	LCOM	0	0	0	0	0
7.	SCCR	12	0	3.147	3	2.57
8.	NODBC	5	0	2.118	1	3.85
9.	MI	100	48	71.79	67	17.84
10.	CC	13	2	10.79	7	12.78
11.	Change	213	18	79.87	103	67.93

- The size of a class measured in terms of lines of source code ranges from 23-7890.
- Max value of LCOM for FLM, EASY, SMS, IMS and ABP are 0, 0, 0, 3 and 6 respectively which represents that classes are quite cohesive in first three applications.
- Values of DIT for FLM, EASY, SMS, IMS and ABP are 7, 5, 6, 5 and 6 which represents that inheritance is properly exploited in all systems.
- SCCR is medium in FLM, EASY and SMS and High in IMS and ABP which means schema are better documented for IMS and ABP system.

Table 5.12: Descriptive Statistics of ABP System

S.No.	Metric	Max	Min	Mean	Median	Std Dev
1.	WMC	11	1	2.483	2	1.84
2.	DIT	6	3	4.017	4	0.13
3.	NOC	9	0	5.25	5	1.09
4.	CBO	29	4	14.93	17	8.56
5.	RFC	49	21	26.83	31	9.89
6.	LCOM	6	0	0.155	0	0
7.	SCCR	11	1	2.483	2	1.84
8.	NODBC	8	0	4.931	1	1.04
9.	MI	100	40	69.5	61	21.03
10.	CC	14	2	10.33	8.5	8.88
11.	Change	189	19	91.23	78	45.63

- A value of NODBC is more than 8 in FLM and ABP systems and less than 7 in EASY, SMS and IMS systems which means first two systems are more data intensive than the later three systems.

5.2.7 Prediction Accuracy Measures

After obtaining the results, we analyzed their performances using various prediction accuracy measures given by Conte et al. [50], Fentom and Bieman [65] and Kitchenham et al. [110]. Most commonly measures are used to adjudge the prediction accuracy such as MRE, MMRE and prediction accuracy at 25% and 30%. Their detail definition and formula of each measure are given in chapter 2.

5.3 Results and Analysis

In this section, we have summarized the descriptive stastistics of ten independent and one dependent variable. Total 864 classes were collected from all five proprietary systems and combined with respective changes made in each class. Data analysis was performed using correlation coefficient to verify the findings. Univariate and Multivariate analysis were performed to find the significance of each metric proposed individually and cumulatively have also been explained.

5.3.1 Univariate Linear Regression

Univariate Analysis using linear regression was performed to find the individual effect of NODBC and SCCR on change and the results are presented in Table 5.13. Four columns represent estimated coefficient, standard error, the t-ratio and p-value. The value of Sig (p-value) represents the amount of significance of these metrics on change. As evident from the outcome, both variables received the p-value as 0.000 which means they are significantly correlated with change.

Table 5.13: Univariate Analysis between NODBC, SCCR and 'Change'

S.No.	**Metrics**	**Unstandardized Coefficients (B)**	**Unstandardized Coefficients (Standard Error)**	**Standardized Coefficients (Beta)**	**t-value**	**Significance (p-value)**
1.	NODBC	0.228	0.047	-0.587	4.868	0.000
2.	SCCR	0.245	0.046	0.626	5.382	0.000

5.3.2 Multivariate Linear Regression

MLR was also performed using stepwise linear regression model in order to identify the most significant metrics for each system. MLR is the most commonly used technique for fitting a linear equation on observed data. There are three methods used for identifying and picking the subset of important metrics from the set of independent variables i.e. forward selection, backward selection and stepwise selection. In this study, the stepwise selection method is used as it guarantees to provide optimum and a most significant subset of independent variables. At each step either the certain variables are added or deleted to identify the final most optimized regression model. Unstandardized Coefficient, Std Error, t-ratio and p-value (sig) to three decimal places are presented in Table 5.14.

Results show that two proposed metrics were found to be significantly correlated with a dependent variable for all systems as almost all p-value are less than .050. Unstandardized Coefficients represents the value when the dependent and independent (predictor) variables

Table 5.14: Multivariate Analysis between NODBC, SCCR and Dependent variable 'Change'

S.No.	Metrics	Unstandardized Coefficients (B)	Unstandardized Coefficients (Standard Error)	Standardized Coefficients (Beta)	t-value	Significance (p-value)
1.	WMC	0.860	0.211	0.055	-0.150	0.002
2.	DIT	0.707	9.876	0.013	0.013	0.074
3.	NOC	0.758	0.463	0.019	-0.055	0.001
4.	CBO	0.912	0.798	0.258	0.049	0.049
5.	RFC	0.345	0.605	0.069	-0.021	0.042
6.	LCOM	0.707	2.463	0.150	0.531	0.062
7.	SCCR	0.301	4.501	0.663	0.659	0.011
8.	NODBC	-0.032	1.268	-0.757	0.661	0.003
9.	MI	0.817	3.412	0.681	0.119	0.010
10.	CC	0.476	3.406	0.146	0.249	0.004

were all transformed to standard scores before running the regression and used to compare the relative strength of the various predictors. NODBC has the largest coefficient and one standard deviation increase in NODBC leads to a 0.915 decrease in change for IMS system. Variable SCCR is also found to be quite competitive as one standard deviation increase in SCCR, in turn, leads to 0.858 standard deviation increase in change for SMS system. Apart from two reported metrics WMC and MI were also found to be a most significant predictor of change.

5.3.3 Correlation Analysis

Correlation Analysis provides important information about the interdependence between two variables. We calculated the Pearson's correlation coefficient represented as 'r' to measures the linear relationship between independent variables versus change and presented in Table 5.15.

The value of 'r' represents the amount of correlation exists between the two variables and lies between +1 to -1. Values in the range of ± 0.5 to ± 1 represent high correlation; ± 0.3 to ± 0.5 represents medium correlation whereas less than ± 0.3 represents very low correlation.

Table 5.15: Pearson Correlation Coefficient at 0.01 Level of Significance (Two-Tailed)

S.No.	**Metrics**	**FLM Change**	**EASY Change**	**SMS Change**	**IMS Change**	**ABP Change**
1.	WMC	0.73	0.66	0.54	0.61	0.59
2.	DIT	0.38	0.42	0.36	0.42	0.44
3.	NOC	0.29	0.48	0.33	0.41	0.45
4.	CBO	0.46	0.61	0.49	0.58	0.51
5.	RFC	0.64	0.49	0.50	0.51	0.47
6.	LCOM	0.48	0.42	0.41	0.68	0.71
7.	SCCR	0.54	0.55	0.66	0.69	0.73
8.	NODBC	0.74	0.65	0.58	0.79	0.81
9.	MI	0.61	0.49	0.36	0.47	0.58
10.	CC	0.59	0.62	0.39	0.41	0.55

It is inferred that NODBC metric, as well as SCCR metric is significantly related to change metric for all the systems. The value of ‘r’ for the newly proposed metric is quite competitive as compared to other metrics. For IMS and ABP systems, more than 75% correlation was observed whereas for FLM, EASY and SMS systems it was in the range of 58-75% which is quite significant. SCCR is also found to be significantly correlated with change metric for all systems. When compared with other metrics it was found that although DIT is comparatively less correlated with the change, however MI and CC are reasonably well correlated. Among the Chidamber and Kemerer metric suite, WMC is found to be most significantly related as for all systems, the value of ‘r’ is found to be more than 54% for all systems. RFC is significantly correlated with change for FLM, SMS and IMS systems. CBO found to be significantly correlated with change in EASY and IMS systems.

5.3.4 Maintainability Prediction

Two types of prediction models were constructed for each system. Model-1 is constructed using metric suite presented by Chidamber and Kemerer and Model-2 is constructed by adding four more metrics MI, CC, NODBC and SCCR to the existing Chidamber and Kemerer metric suite resulting in the set of 10 metrics in all. BPNN, Kohonen Network, FFNN and GRNN were employed for software maintainability prediction by dividing the data into three parts i.e. 70% for training and 30% for testing as it is the commonly accepted proportion used by many practitioners. Three prediction accuracy measures proposed by Kitchenham et al. [110] are used to compare the performance of Model-1 and Model-2 as discussed in chapter 2 i.e. Max MRE, MMRE and Prediction accuracy at 25%. Results are presented in Tables 5.16, 5.17, 5.18 and 5.19 when BPNN, Kohonen Network, FFNN and GRNN were used respectively for machine learning. For each software system, values of accuracy measures are shown when all four machine learning techniques are applied with metric suite Model-1 and Model-2. For example first three rows belong to the results received using FLM system.

Table 5.16: Comparison of Chidamber and Kemerer Metric Suite and Proposed Metric Suite with Back Propagation Neural Network Machine Learning Technique

Software System	**Prediction Accuracy Measures**	**Proposed Metric Suite (Model-2)**	**Chidamber and Kemerer metric suite (Model-1)**
FLM System	Max MRE	1.207	1.987
	MMRE	0.47	0.49
	Pred(0.25)	0.69	0.57
EASY System	Max MRE	1.656	1.246
	MMRE	0.46	0.51
	Pred(0.25)	0.63	0.59
SMS System	Max MRE	1.223	1.302
	MMRE	0.40	0.44
	Pred(0.25)	0.69	0.52
IMS System	Max MRE	1.431	1.521
	MMRE	0.35	0.40
	Pred(0.25)	0.71	0.59
ABP System	Max MRE	1.339	1.292
	MMRE	0.29	0.37
	Pred(0.25)	0.67	0.58

5.3.5 Validation of Hypotheses

In this section, we validate our hypothesis stated in section 5.2.4.

Table 5.17: Comparison of Chidamber and Kemerer Metric Suite and Proposed Metric Suite with Kohonen Network Machine Learning Technique

Software System	Prediction Accuracy Measures	Proposed metric suite (Model-2)	Chidamber and Kemerer metric suite (Model-1)
FLM System	Max MRE	1.090	1.876
	MMRE	0.42	0.45
	Pred(0.25)	0.78	0.68
EASY System	Max MRE	0.985	0.998
	MMRE	0.36	0.46
	Pred(0.25)	0.74	0.69
SMS System	Max MRE	0.973	1.112
	MMRE	0.41	0.43
	Pred(0.25)	0.77	0.63
IMS System	Max MRE	1.109	2.332
	MMRE	0.32	0.43
	Pred(0.25)	0.70	0.68
ABP System	Max MRE	0.332	1.898
	MMRE	0.32	0.43
	Pred(0.25)	0.72	0.62

Table 5.18: Comparison of Chidamber and Kemerer Metric Suite and Proposed Metric Suite with Feed Forward Neural Network Machine Learning Technique

Software System	Prediction Accuracy Measures	Proposed metric suite (Model-2)	Chidamber and Kemerer metric suite (Model-1)
FLM System	Max MRE	0.989	1.332
	MMRE	0.39	0.41
	Pred(0.25)	0.71	0.66
EASY System	Max MRE	1.023	1.090
	MMRE	0.37	0.43
	Pred(0.25)	0.77	0.70
SMS System	Max MRE	1.109	2.786
	MMRE	0.38	0.46
	Pred(0.25)	0.69	0.52
IMS System	Max MRE	1.223	1.803
	MMRE	0.30	0.39
	Pred(0.25)	0.63	0.59
ABP System	Max MRE	1.667	2.092
	MMRE	0.37	0.48
	Pred(0.25)	0.66	0.67

Table 5.19: Comparison of Chidamber and Kemerer Metric Suite and Proposed Metric Suite with General Regression Neural Networks Machine Learning Technique

Software System	**Prediction Accuracy Measures**	**Proposed metric suite (Model-2)**	**Chidamber and Kemerer metric suite (Model-1)**
FLM System	Max MRE	1.112	1.782
	MMRE	0.41	0.48
	Pred(0.25)	0.76	0.68
EASY System	Max MRE	1.329	1.478
	MMRE	0.42	0.49
	Pred(0.25)	0.69	0.68
SMS System	Max MRE	1.762	1.986
	MMRE	0.39	0.42
	Pred(0.25)	0.71	0.60
IMS System	Max MRE	1.632	1.886
	MMRE	0.29	0.38
	Pred(0.25)	0.69	0.58
ABP System	Max MRE	1.456	1.672
	MMRE	0.40	0.58
	Pred(0.25)	0.74	0.71

5.3.5.1 H1 Hypothesis

From the results, it is quite evident that overall improvement in the prediction accuracy is observed with new proposed metric suite for all systems. To further analyze the results we further sorted the systems in ascending order on the values of NODBC and SCCR. We observed that more improvement in prediction accuracy was achieved for those systems which have high values of NODBC and SCCR. ABP system has maximum SCCR and NODBC as compared to other systems. For ABP system, maximum improvement in prediction accuracy is observed i.e. 23% in the for MMRE whereas other systems such as FLM, EASY, SMS and IMS observed 7%, 14%, 11%, and 19% improvement in MMRE respectively. MaxMRE was improved by 39%, 1%, 21% 28% and 29% for FLM, EASY, SMS, IMS and ABP System respectively.

Lowest improvement for Easy systems was noticed which also has lowest SCCR as well as NODBC among all systems. Prediction accuracies achieved by all models were also compared and observed that the performance of machine learning models is better than MLR in general.

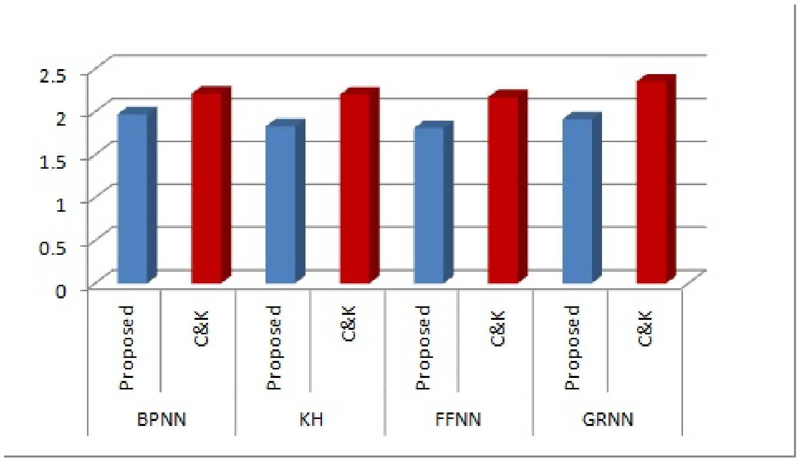

Figure 5.1: Comparison of Model 1 (M-1, Chidamber and Kemerer metric suite) and Model 2 (M-2, Proposed metric suite) with Reference to their MMRE values

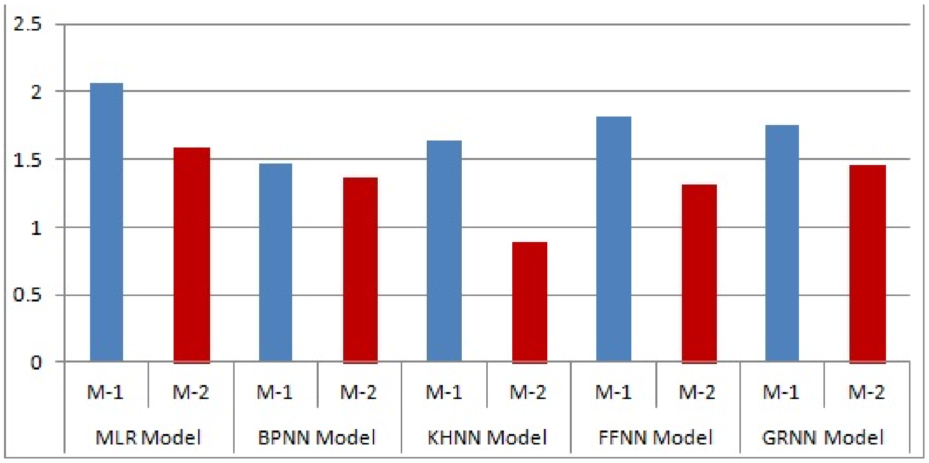

Figure 5.2: Comparison of Model 1 (M-1, Chidamber and Kemerer metric suite) and Model 2 (M-2, Proposed metric suite) with Reference to Max MRE values

When we compared the MMRE values for Model-2, it is found to be 0.94, 0.82, 0.66, 0.79 and 0.86 for MLR, BPNN, Kohonen Network, FFNN and GRNN respectively. That means Kohonen Network performance is best among all machine learning models.

Graphs were also plotted to observe the improvement in prediction accuracies from Model-1 to Model-2 and presented in figure 5.1, figure 5.2 and figure 5.3 for MMRE,

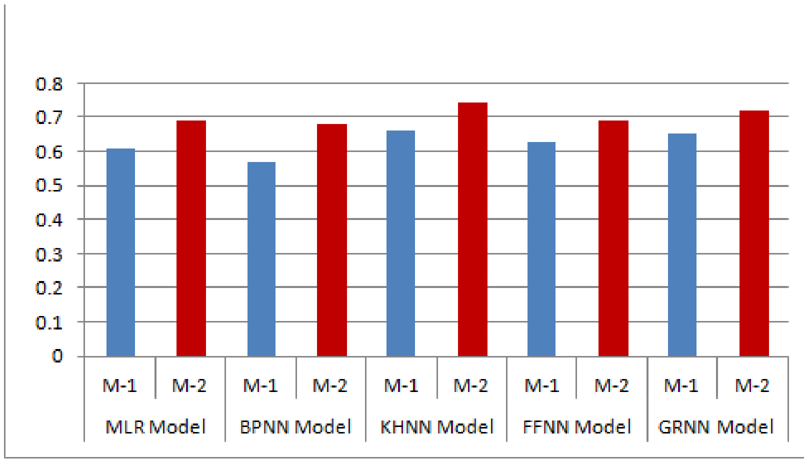

Figure 5.3: Comparison of Model 1 (M-1, Chidamber and Kemerer metric suite) and Model 2 (M-2, Proposed metric suite) at 25% prediction accuracy

MaxMRE and Pred(0.25) respectively. Enhancement in MaxMRE, MMRE and Pred(0.25) were observed in Model-2 than Model-1 for all prediction models. In figure 5.2, improvement in the value of MaxMRE is observed from Model-1 to Model-2. Figure 5.3 represents the comparison of prediction accuracies achieved at 25%. It is quite visible from the graph that pred(0.25) is improved for all systems. Hence, we accept the alternate hypothesis and reject the null hypothesis and on the basis of these results, we reasonably claim that proposed metric suite is more efficient and concise in predicting maintainability.

5.3.5.2 H2 Hypothesis

In order to observe the performance of GMDH model, we compared the MMRE values on all five datasets viz FLM, EASY, SMS, IMS and ABP for all five prediction models i.e. GMDH, BPN, Kohonen Network, FFNN and GRNN and compiled them in Table 5.20. We observed that the performance of GMDH is found to be best among all machine learning models. Graphs were also plotted to observe the improvement in terms of MMRE prediction accuracy measure. According to the results as visible in figure 5.4, the best performing model on all the datasets was developed using GMDH technique. This model gave an accuracy measure of 65%, 66%, 72%, 73% and 70% on FLM, EASY, SMS, IMS and ABP system respectively.

Table 5.20: Prediction Accuracy Measures for Various Techniques on all datasets

S.No.	Model Name	FLM System	EASY System	SMS System	IMS System	ABP System
1	GMDH Model	0.356	0.342	0.282	0.273	0.308
2	BPN Model	0.47	0.46	0.40	0.35	0.29
3	Kohonen Network Model	0.426	0.368	0.419	0.321	0.327
4	FFNN Model	0.391	0.375	0.384	0.303	0.372
5	GRNN Model	0.412	0.429	0.397	0.298	0.413

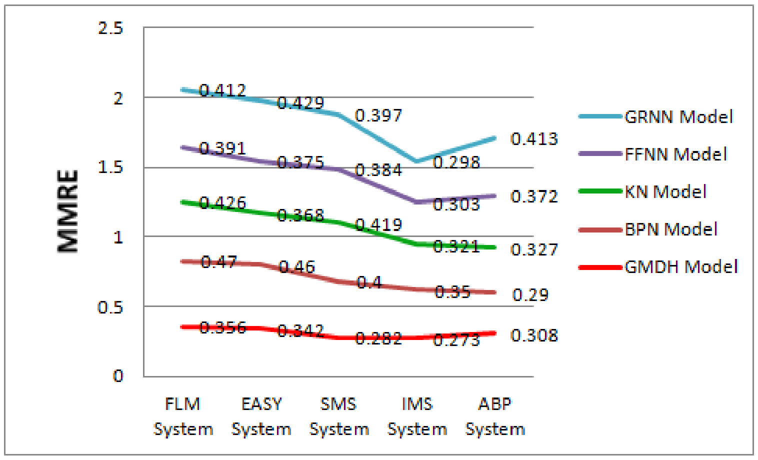

Figure 5.4: Comparison of MMRE Prediction Accuracy Measure for Machine Learning Various Techniques

In the current chapter, we were analyzing the performance of five techniques on five datasets. Although it's visible from the figure 5.4 and Table 5.20 that there is the difference in the performance of the models developed using various techniques, we need to assess whether the difference in performance is significant statistically. In order to verify the performance of GMDH model, we conducted Friedman Statistical test and obtained mean ranks for each of the techniques and compiled in Table 5.21. This classification is based on the MMRE values received by each of the models on all datasets. According to the results, the best performing technique on all datasets was the GMDH technique as it obtained a mean rank of 2.45. The second rank was given to the BPN technique which was closely followed by the Kohonen Network technique. The GRNN technique was designated as the worst technique with a mean rank of 6.83. Hence we reject the null hypothesis, accept the alternate hypothesis and reasonably claim that GMDH technique outperforms other machine learning techniques in maintainability prediction model.

Table 5.21: Mean Rank assigned to Various Machine Learning techniques from Friedman Test Results w.r.t. MMRE Values

S.No.	Technique	Mean Rank
1.	GMDH	2.45
2.	BPN	3.53
3.	Kohonen Network	3.83
4.	FFNN	4.53
5.	GRNN	4.68

5.4 Discussion

We found that the new proposed metric suite is significantly related to the dependent variable. It is also observed that maintainability predictions for the applications which heavily use databases were more precise and accurate using the new metric suite. Univariate as well as multivariate analysis further confirmed the results and proved the significance of proposed metric suite. Software practitioners can considerably take decisions whether the developed application is maintainable or not, which would save the time and money for the organizations responsible for developing and deploying the customized software's for the customers to gain their better satisfaction in the industry.

The goal of the empirical study conducted in this chapter is to empirically examine the effectiveness of new proposed metric suite for predicting software maintainability for data intensive applications as it's important to give equal attention to the database accesses with the increase in data as well as the number of times data get accessed. We employed GMDH, BPNN, FFNN, Kohonen Network, and GRNN techniques for making software maintainability prediction model. Observing five proprietary software namely FLM system, EASY system, SMS system, IM system and ABP system over a period of three years, we analyzed the performance of proposed metric suite using prediction accuracy measures such as MRE, MMRE and pred(0.25). The results of this work are summarized as follows:

1. Two metrics Maintainability Index and Cyclomatic Complexity are added to measure the structural complexity of the software.

2. Two metrics Number of Database Connections (NODBC) and Schema to comment ratio (SCCR) are added to measure the database handling aspect of the software.

3. Univariate analysis of the proposed metrics NODBC and SCCR indicates that they are very much correlated to change as p-value with between metrics and change is received as 0.000, hence, both of them can be used along with OO metrics for the early detection of change.

4. Multivariate analysis using stepwise linear regression identified NODBC and SCCR as a good indicator of software maintainability in data intensive applications. It was found that maximum two values of correlation coefficient are 0.915 between NODBC and change for IMS system and 0.858 between SCCR and change for SMS system. That means NODBC and SCCR are highly correlated with change.

5. The predicted results indicate that proposed metric suite is significant indicator of software maintainability as improvements in all five datasets were observed by 23%, 7%, 14%, 11%, and 19% for ABP system, FLM, EASY, SMS and IMS system respectively

6. When four metrics are added to Chidamber and Kemerer metric suite and prediction accuracy measure MaxMRE is used, we observed that it is improved by 39%, 1%, 21% 28% and 29% for FLM, EASY, SMS, IMS and ABP System respectively.

7. We observed that the performance of GMDH is best among all machine learning techniques as accuracy of 65%, 66%, 72%, 73% and 70% is achieved on FLM, EASY, SMS, IMS and ABP system respectively.

8. We conducted Friedman Statistical test and obtained mean ranks for each of the technique based on MMRE values. Results indicated the best performing technique on all

datasets was the GMDH technique as it obtained a mean rank of 2.45. We also found the GRNN technique as the worst technique with a mean rank of 6.83 in respect of MMRE values.

9. The results help us in identification of those classes which require big share of maintenance resources.

Chapter 6

Benchmarking Framework for Maintainability Prediction of Open Source Software using Object-Oriented Metrics

6.1 Introduction

Controlling the software maintainability and understandability of any open source software system is extremely challenging because it's written and constantly modified by the developers located all over the world. The open source software development process facilitates the production of low cost software in very less time. Many companies willingly provide funds to the open source development community because they can further use these software in their own work. As these types of software systems are written and modified by different persons, it is essential that they must be easy to understand and maintain. Since the development process of open source software is entirely different from proprietary software, it is also equally important to develop maintainability prediction model for open source software.

This study analyzes the effectiveness of machine learning techniques for the maintainability prediction of open source software systems. In this work, large-scale empirical comparisons of thirteen classifiers over seven open source datasets were conducted followed by extensive statistical tests and post hoc analysis to establish the confidence on the performance of one machine learning technique over another. In this study, maintenance prediction model is developed to assess accurate maintainability of open source system during the early phases of the SDLC. This is achieved with the help of some measurable software design characteristics such as cohesion, coupling, abstraction, complexity and inheritance as suggested by Jorgensen [101] and Lucia et al. [133].

Although various maintainability prediction models using statistical and machine learning techniques have been developed in past [13, 34, 53, 62, 68, 100, 118, 127, 133, 162, 210, 219, 237] but to the best of author's knowledge, studies on observing the maintainability of open source software systems are very limited except one conducted by Zhou and Xu [238]. Even though Ramil et al. [180] has compiled many empirical studies on open source software, however, all the studies focused on intuitively judging the software maintainability instead of creating a mathematical prediction model. Myrtveit et al. [165] have also raised an important issue that more reliable research procedures must be developed before believing on the outcome of any one of the prediction models. As the interest for open source software has been rising across the globe, a powerful customized machine learning technique based prediction model for open source software was seen as the potential scope of research.

In order to address these issues, an effort has been made in this study to create maintainability prediction model for open source dataset using machine learning techniques. In order to get unbiased, accurate and repeatable maintainability prediction model for open source software, the current study attempts to create an empirical framework using thirteen machine learning techniques over various releases of seven open source software. The main contribution of this chapter is summarized as follows:

- Analyze the characteristics of open source software.
- Explore the impact of OO metrics on maintainability in the context of open source software?
- Comparative performance of machine learning techniques is analyzed for maintainability prediction using open source software.
- Identify which machine learning techniques perform significantly better from another in terms of prediction accuracy measures?

We extensively compare the experimental results of thirteen machine learning classifiers over seven open source software systems using statistical test followed by post hoc analysis to scrutinize if there exists a significant difference among the performance of any particular machine learning technique. The thirteen selected classifiers include Linear Regression (LR), M5Rules, DT, SVM, K Star, Bagging, JERN, BPN, Kohonen Network, PNN, GMDH, GRNN, and GGAL. The source code of seven open source software Drumkit, OpenCV, Abdera, Ivy, Log4j, JEdit and JUnit is obtained from http://sourceforge.net and https://apache.org to carry out this widespread investigation.

The chapter is organized as follows: Section 6.2 begin with the hypothesis to be tested in this study followed by the research background, the definition of the OO metrics, sources for empirical data collection and description of machine learning techniques. In section 6.3, the results of the study are evaluated and validation of all hypothesis is performed. Discussion of the study is presented in section 6.4.

The results of this chapter have been reported in [142].

6.2 Research Background

Development of open source software is entirely different from proprietary software system even though both the systems are not polar opposites. In the open source software, term ’openness’ refers to the ability of diverse parties to create a technology that can inter-operate.

Open source software is usually developed by developers spread geographically apart over the world and working cooperatively without the need of license. Several empirical studies have carried out to investigated the relationships between design metrics and maintainability of open source software. [130, 162, 192, 238]. There are many advantages associated with open source software over proprietary software. Since open source software projects is developed by millions of persons, the probability of detecting an error is higher in contrast with proprietary software which have far smaller development staff. Furthermore, open source communities are quick to fix if there is an error. Even the customers are also free to apply their own patches at will. Maintenance of open source software is comparatively easier because there is minimal reliance on a single vendor or group for continued improvements.

Basically, there are two broad approaches in which the maintainability of a software can be measured, firstly through the measurement of external quality factors such as understandability, analyzability, modifiability etc. and secondly through the measurement of internal quality metrics and use them for making software maintainability prediction model. In the first approach, external factors can only be measured by collecting the opinion from the developers who participate in writing the source code of the open source software. Conducting such surveys is not only time-consuming and involves high cost but also produces different opinion due to the subjective nature of external quality factors. The second approach of measuring the internal quality attributes through OO metrics suite has been used in many empirical studies [53, 62, 68, 100, 118, 127, 133, 137, 141, 210, 219, 237]. Almost all of the studies showed the existence of the relationship between OO metric suite and maintainability. In the current empirical investigation, the second approach is used and we have adopted the research methodology presented in chapter 2.

This section presents the selection of dependent and independent variables, states the hypothesis to be tested and subsequently the process of collecting the empirical data for the validation of machine learning techniques. We have also presented the descriptive statistics followed by their interpretations. Various machine learning techniques used in the prediction model are also explained along with their parameter value settings.

6.2.1 Hypotheses

Following hypothesis are tested to compare the performance various machine learning techniques used in this study:

6.2.1.1 H1 Hypothesis

- Null Hypothesis: Impact of OO metrics on software maintainability does not exist in the context of open source software?
- Alternate Hypothesis: Impact of OO metrics on software maintainability exist in the context of open source software?

6.2.1.2 H2 Hypothesis

- Null Hypothesis: There is no significant difference between the performance of machine learning techniques.
- Alternate Hypothesis: There exist significant difference between the performance of machine learning techniques.

6.2.1.3 H3 Hypothesis

- Null Hypothesis: The GMDH and GGAL technique does not outperforms the eleven compared techniques (LR, M5Rules, DT, SVM, K Star, Bagging, JERN, BPN, Kohonen Network, PNN and GRNN) in predicting software software maintainability of the classes.
- Alternate Hypothesis: The GMDH and GGAL technique outperforms the eleven compared techniques (LR, M5Rules, DT, SVM, K Star, Bagging, JERN, BPN, Kohonen Network, PNN and GRNN) in predicting software software maintainability of the classes.

6.2.2 Independent and Dependent Variables

Our goal was to capture the various attributed of OO paradigm such as size, coupling, cohesion, abstraction, complexity and inheritance. In the current study, we have used Chidamber and Kemerer metric suite which is very common and used by various researchers [53, 62, 100, 106, 118, 137, 141, 219, 237]. In previous chapter 5, we realized few shortcomings in Chidamber and Kemerer metric suite such as it does not take into account the structural complexity of the software and any metric on account of the amount of database handling. In order to overcome such shortcomings, in addition to Chidamber and Kemerer metric suite, we have also included the metric suites proposed by Henderson-Seller [89] and Bansiya and Davis [15]. Various OO metrics were carefully selected from the metric suites proposed by researchers [15, 43, 89] to capture all the design attributes such as coupling, cohesion, inheritance, abstraction and complexity of OO paradigm. In total 17 independent variables are selected consisting of 6 from the Chidamber and Kemerer metric suite (WMC, DIT, NOC, RFC, CBO, LCOM), 2 from Martin Metric suite (Ca, Ce), 5 from Bansiya and Devis metric suite (NPM, MOA, MFA, CAM, DAM), 1 from Li and Henry metric suite (Size1 as LOC), 1 from Tang metric suite (IC) and 1 from Henderson-Seller metric suite (LCOM3). The detail definitions of these metrics are provided in chapter 2. The dependent variable in the current study is maintenance effort observing the number of changes made between two consecutive versions and it is counted in terms of a number of lines of source code added, deleted or modified in the newer version with respect to the older version for each class. Same approach is adopted by many researchers [53, 62, 68, 100, 118, 127, 133, 137, 141, 210, 219, 237].

6.2.3 Empirical Data Collection

We explore open source repositories for collecting the empirical data keeping in mind two important characteristics which include that it should follow OO paradigm and it should have a high number of downloads in recent times (last 12 months) as it is a clear indication that there are active users contributing constantly. Seven open source software systems collected in this regard are Drumkit, OpenCV, Abdera, Ivy, Log4J, JEdit and JUnit. Their details such as versions, release date, size, number of classes etc are summarized in chapter 2. After the data collection, pre-processing was performed as described in chapter 2 during which we extract those classes which are common in current as well as the previous version for each software system. Classes either added in latest version or deleted from the older version are simply discarded. Library classes, as well as interface classes were also excluded from the list.

6.2.4 Descriptive Statistics

For the purpose of the qualitative analysis, the descriptive statistics are calculated from the collected data for each of the selected software in the current study, Outliers were removed by taking 95 percentile of each metrics followed by calculations of descriptive statistics. The Mean, Median, Standard Deviation, Minimum and Maximum for all the Chidamber and Kemerer metrics were calculated and presented in Table 6.1 and Table 6.2, Table 6.3, Table 6.4, Table 6.5, Table 6.6, Table 6.7 for Drumkit, OpenCV, Abdera, Ivy, Log4J, JEdit and JUnit respectively. It is very useful in understanding and comparing the characteristics of both OO systems. During this process 11 classes of Drumkit, 14 classes of OpenCV, 37 classes of Abdera, 16 classes of Ivy, 23 classes for Log4J, 18 classes of JEdit and 33 classes for JUnit datasets were removed and following observations were made from the descriptive statistics:

- Size measured in terms of lines of source code i.e. LOC is ranging from 0 to 2558 for all selected software.

- The mean values of DIT and NOC are Drumkit (0.78, 0.20), OpenCV (1.70, 0.38), Abdera (0.91, 0.17), Ivy (0.64, 0.24), Log4J (0.95, 0.20), JEdit (1.21, 0.46) and JUnit (0.73, 0.42) which means inheritance is comparatively less exploited in all the systems. The median of DIT for all software is more than 0 which means that at least more

Table 6.1: Descriptive Statistics for Drumkit System

Name of the Metric	Mean	Median	Std Dev	Max	Min
WMC	8.02	10.00	34.44	213.00	1.00
DIT	0.78	1.00	0.65	4.00	0.00
NOC	0.20	0.00	0.77	5.00	0.00
CBO	5.60	2.50	7.48	34.00	0.00
RFC	27.22	14.00	36.30	214.00	2.00
LCOM	803.75	27.00	2796.44	22578.00	0.00
Ca	1.66	0.00	3.72	24.00	0.00
Ce	4.20	2.00	5.94	34.00	0.00
NPM	15.48	5.00	30.88	212.00	0.00
LCOM3	1.21	1.07	0.44	2.00	0.00
LOC	213.93	99.50	296.03	1721.00	4.00
DAM	0.40	0.16	0.42	1.00	0.00
MOA	1.52	0.00	4.03	37.00	0.00
MFA	0.05	0.00	0.22	1.00	0.00
CAM	0.40	0.37	0.26	1.00	0.00
IC	0.02	0.00	0.14	1.00	0.00
CBM	0.02	0.00	0.14	1.00	0.00
AMC	8.30	5.00	16.48	144.00	0.00

than half of the classes have a parent class. Thus, inheritance is widely used in these selected software.

- Cohesion which is measured through LCOM have high mean values for Drumkit (803.75), OpenCV (111.17), Abdera (273.18), Ivy (308.80), Log4J (130.65), JEdit (244.36) and JUnit (24.88) that means cohesion is high in all systems.

- the value of the metric CBO which is used to measure the interaction between the classes is high for all the software used except Poi which indicates that there is a high interaction between classes.

- WMC is used for comparing the complexity characteristics between two software. It was observed that its value for three software i.e. Abdera (11.48), Ivy (12.90) and JEdit (10.77) is more than 10, that means these three software systems are comparatively more complex.

Table 6.2: Descriptive Statistics for OpenCV System

Name of the Metric	Mean	Median	Std Dev	Max	Min
WMC	9.71	4.00	13.17	87.00	1.00
DIT	1.70	1.00	1.84	6.00	0.00
NOC	0.38	0.00	2.37	23.00	0.00
CBO	10.26	5.00	13.25	68.00	0.00
RFC	28.17	15.00	30.93	226.00	1.00
LCOM	111.17	3.00	399.55	3705.00	0.00
Ca	6.49	2.00	11.61	68.00	0.00
Ce	4.61	2.00	5.82	31.00	0.00
NPM	7.69	3.00	12.10	85.00	0.00
LCOM3	1.11	0.97	0.62	2.00	0.00
LOC	231.65	85.00	346.14	2558.00	1.00
DAM	0.61	0.60	0.05	1.00	0.05
MOA	0.71	0.00	2.67	33.00	0.00
MFA	0.24	0.00	0.41	1.00	0.00
CAM	0.57	0.56	0.27	1.00	0.00
IC	0.17	0.00	0.44	3.00	0.00
CBM	0.32	0.00	1.24	15.00	0.00
AMC	19.40	14.63	22.15	175.17	0.00

6.2.5 Machine Learning Techniques

Overall thirteen machine learning techniques are used in the current study for making prediction models namely LR, M5Rules, DT, SVM, K Star, Bagging, JERN, BPN, Kohonen Network, PNN, GMDH, GRNN, and GGAL. Their details are provided in chapter 2.

6.3 Result Analysis

This section presents the prediction results of various classifiers based on machine learning techniques for maintainability prediction using OO metrics.

6.3.1 Feature Sub Selection

The aim for carrying out the feature sub-selection process is to remove irrelevant and redundant independent variables from the dataset before it can be used further by the seventeen classifiers selected in the current study for training purpose as suggested by Donell [60]. This dimensionality reduction process not only reduces the unnecessary attributes and

Table 6.3: Descriptive Statistics for Abdera System

Name of the Metric	Mean	Median	Std Dev	Max	Min
WMC	11.48	6	20.67	255	0
DIT	0.91	1	0.61	4	0
NOC	0.17	0	0.98	17	0
CBO	1.17	1	1.94	17	0
RFC	12.42	7	20.70	256	0
LCOM	273.18	15	1665.13	32385	0
Ca	0.60	0	1.63	17	0
Ce	0.61	0	0.93	5	0
NPM	9.96	4	20.08	254	0
LCOM3	1.65	2	0.43	2	1.0039
LOC	64.74	27	120.40	1531	0
DAM	0.39	0	0.48	1	0
MOA	0.89	0	14.28	327	0
MFA	0.03	0	0.16	1	0
CAM	0.48	0.4375	0.30	1	0
IC	0.06	0	0.27	3	0
CBM	0.07	0	0.41	5	0
AMC	3.28	5	2.24	5	0

irrelevant noisy data, but it also enhances the execution time, improves the quality of datasets and thereof amplifies the accuracy of the prediction process. The first step in this empirical study was FSS in which irrelevant and unimportant features were removed. Table 6.8 summarizes the relevant metrics found after applying FSS over all the releases of seven datasets selected in the current study.

Out of the seventeen independent variables, we found that LCOM3, LOC, and DIT are the most commonly selected OO metrics in the current study. Efferent coupling, Ce is also found to be significant in Abdera and Log4j Systems. In total 17%, 17%, 41%, 35%, 23%, 41% and 35% reductions were observed for Drumkit, OpenCV, Abdera, Ivy, Lo4j, JEdit and JUnit datasets respectively. On an average 24% saving is observed for all datasets. We also found that the results obtained using reduced set of independent variables after applying FSS were slightly better as compared to the results obtained using all independent variables in prediction models. Similar observations were made by Kohavi and John [115] as well as

Table 6.4: Descriptive Statistics for Ivy System

Name of the Metric	Mean	Median	Std Dev	Max	Min
WMC	12.90	7.00	21.56	243.00	1.00
DIT	0.64	1.00	0.60	4.00	0.00
NOC	0.24	0.00	1.22	17.00	0.00
CBO	1.81	1.00	1.96	17.00	0.00
RFC	13.90	8.00	21.56	244.00	2.00
LCOM	308.80	21.00	2147.88	29403.00	0.00
Ca	0.82	0.00	1.74	17.00	0.00
Ce	1.05	1.00	1.17	9.00	0.00
NPM	10.42	5.00	18.89	215.00	0.00
LCOM3	1.45	1.25	0.42	2.00	1.00
LOC	77.00	42.00	132.46	1461.00	6.00
DAM	0.60	1.00	0.47	1.00	0.00
MOA	0.24	0.00	0.69	7.00	0.00
MFA	0.02	0.00	0.12	1.00	0.00
CAM	0.53	0.48	0.28	1.00	0.06
IC	0.04	0.00	0.21	2.00	0.00
CBM	0.04	0.00	0.21	2.00	0.00
AMC	2.57	2.50	2.39	5.00	0.00

Yang and Honavar [231] that not only the impact of FSS on the accuracy is minimal but they are also capable of capturing all the characteristics irrespective of the size of the extracted subset. Moreover, the time consumed by prediction model on newly reduced dataset using FSS is comparatively lesser than the time consumed on the actual dataset.

6.3.2 Summary of Results for Various Prediction Accuracy Measures

Firstly, we present the results of all 13 machine learning techniques for maintainability prediction models validated using 10-fold cross-validation on seven open source software. The difference between the predicted value and actual value is compared and analyzed using various accuracy measures such as MARE, RMSE, Pred(25%) and Pred(75%) and described in following subsections respectively.

Table 6.5: Descriptive Statistics for Log4J System

Name of the Metric	Mean	Median	Std Dev	Max	Min
WMC	9.29	5	14.16	123	1
DIT	0.95	1	1.09	6	0
NOC	0.20	0	1.20	16	0
CBO	3.06	1	6.65	76	0
RFC	14.42	7	20.48	137	1
LCOM	130.65	6	696.02	7503	0
Ca	1.47	0	5.41	65	0
Ce	1.65	1	2.98	29	0
NPM	7.42	4	12.39	122	0
LCOM3	1.41	1.3	0.52	2	0
LOC	102.63	36	207.78	1864	1
DAM	0.40	0	0.45	1	0
MOA	0.43	0	1.25	14	0
MFA	0.10	0	0.28	1	0
CAM	0.48	0.45	0.22	1	0
IC	0.08	0	0.34	3	0
CBM	0.13	0	0.55	5	0
AMC	6.35	5	15.81	205	0

6.3.2.1 Mean Absolute Relative Error

Predicted value of the dependent variable ‘change’ is compared with the actual value of change for each class and the mean absolute error is calculated as given in equation (2.8) defined in chapter 2. The MARE values of each machine learning technique for all seven datasets in the study are summarized in Table 6.9.

From the table, it is observed that when LR technique was applied on JEdit datasets, it gave an accuracy of 62% (since the error is 0.38, its accuracy is 100-38). Similarly, we found 67% accuracy when GGAL technique applied on Drumkit datasets respectively. Best machine learning technique found to be the GGAL technique as it has achieved 33%, 31%, 29%, 36%, 28%, 34% and 36% error for the Drumkit, OpenCV, Abdera, Ivy, Lo4j, JEdit and JUnit datasets respectively. Second best machine learning technique is found to be the GMDH technique as it has achieved 49%, 37%, 32%, 34%, 32%, 30% and 35% error for the Drumkit, OpenCV, Abdera, Ivy, Lo4j, JEdit and JUnit datasets respectively. The accuracy

Table 6.6: Descriptive Statistics for JEdit System

Name of the Metric	Mean	Median	Std Dev	Max	Min
WMC	10.77	5.00	24.85	351.00	0.00
DIT	1.21	1.00	1.46	7.00	0.00
NOC	0.46	0.00	2.71	38.00	0.00
CBO	11.69	7.00	22.78	396.00	0.00
RFC	34.77	19.00	54.74	570.00	0.00
LCOM	244.36	3.00	2376.47	41713.00	0.00
Ca	6.77	2.00	18.91	327.00	0.00
Ce	6.26	4.00	8.93	116.00	0.00
NPM	6.58	3.00	15.04	228.00	0.00
LCOM3	0.99	0.81	0.69	2.00	0.00
LOC	347.70	130.00	962.51	1253.00	0.00
DAM	0.45	0.28	0.46	1.00	0.00
MOA	1.01	0.00	1.89	13.00	0.00
MFA	0.15	0.00	0.33	1.00	0.00
CAM	0.48	0.45	0.25	1.00	0.00
IC	0.16	0.00	0.55	3.00	0.00
CBM	0.49	0.00	2.34	21.00	0.00
AMC	25.80	18.87	27.07	201.50	0.00
CHANGE	4.96	0.00	22.05	249.00	0.00

of all the machine learning techniques w.r.t. MARE on all seven selected datasets lies between the ranges of 39-77% which is quite encouraging. Thus, it highlights the capability of machine learning technique for effective maintainability predictions of open source software.

6.3.2.2 Root Mean Square Error

For each class, the value of ‘change’ is compared with predicted value of change and the value of RMSE is obtained using equation 2.9 after performing ten runs of ten-fold cross-validation for each machine learning technique on each dataset are tabulated in Table 6.10.

In the table 6.10, each row represents the RMSE value of a particular technique on specific datasets. For example, first row compiles the value of RMSE when LR technique is used with all seven datasets Drumkit, OpenCV, Abdera, Ivy, Lo4j, JEdit and JUnit datasets and generates values as 0.66, 0.56, 0.61, 0.58, 0.49, 0.42 and 0.47 respectively. Similarly, last row compiles the value of RMSE when GGAL technique is used with all seven datasets Drumkit,

Table 6.7: Descriptive Statistics for JUnit System

Name of the Metric	Mean	Median	Std Dev	Max	Min
WMC	5.10	3.00	6.33	50.00	0.00
DIT	0.73	1.00	0.84	4.00	0.00
NOC	0.42	0.00	1.68	16.00	0.00
CBO	5.80	4.00	6.54	53.00	0.00
RFC	12.30	7.00	14.67	80.00	0.00
LCOM	24.88	0.00	105.89	1225.00	0.00
Ca	3.15	1.00	5.86	53.00	0.00
Ce	3.15	2.50	3.37	24.00	0.00
NPM	3.43	2.00	4.83	42.00	0.00
LCOM3	1.02	0.71	0.84	2.00	0.00
LOC	63.67	30.00	94.95	546.00	0.00
DAM	0.40	0.00	0.48	1.00	0.00
MOA	0.54	0.00	0.82	3.00	0.00
MFA	0.06	0.00	0.24	1.00	0.00
CAM	0.50	0.50	0.29	1.00	0.00
IC	0.02	0.00	0.14	1.00	0.00
CBM	0.02	0.00	0.18	2.00	0.00
AMC	8.55	7.29	8.00	53.50	0.00

Table 6.8: Metrics Obtained using Feature Subset Selection Technique

Name of the Software	Selected Relevant OO Attributes
Drumkit	WMC, RFC, DIT, LCOM3
OpenCV	CBO, DIT, LCOM3, LOC
Abdera	Ce, NPM, LOC, LCOM3, DAM, CAM
Ivy	LCOM3, LOC, DAM, MOA, CAM, AMC
Log4j	NPM, Ce, LOC, LCOM3, DIT, MOA, CAM
JEdit	WMC, LOC, DIT, DAM, CAM, AMC
JUnit	RFC, CBO, LCOM, LCOM3, NPM, IC

OpenCV, Abdera, Ivy, Lo4j, JEdit and JUnit datasets and generates values as 0.33, 0.41, 0.49, 0.36, 0.28, 0.37 and 0.32 respectively. Best result was found at 77% accuracy when GMDH technique applied on Drumkit datasets. Best machine learning technique found to be the GMDH technique as it has achieved 23%, 37%, 32%, 34%, 37%, 38% and 42% error for the Drumkit, OpenCV, Abdera, Ivy, Lo4j, JEdit and JUnit datasets respectively. Second best machine learning technique is found to be the GMDH technique as it has achieved 49%, 37%, 32%, 34%, 32%, 30% and 35% error for the Drumkit, OpenCV, Abdera, Ivy, Lo4j,

Table 6.9: Mean Absolute Relative Error Values

dataset	Drumkit	OpenCV	Abdera	Ivy	Log4j	JEdit	JUnit
LR	0.57	0.48	0.41	0.29	0.36	0.38	0.54
M5Rule	0.49	0.52	0.43	0.37	0.38	0.41	0.52
DT	0.58	0.57	0.44	0.39	0.42	0.45	0.49
SVM	0.43	0.64	0.46	0.49	0.46	0.61	0.52
K Star	0.45	0.62	0.51	0.33	0.48	0.42	0.43
Bagging	0.51	0.45	0.43	0.41	0.47	0.52	0.44
JERN	0.53	0.54	0.41	0.59	0.51	0.52	0.44
BPN	0.42	0.58	0.61	0.37	0.35	0.42	0.47
Kohonen Network	0.56	0.49	0.37	0.41	0.43	0.52	0.50
PNN	0.38	0.42	0.38	0.53	0.37	0.41	0.47
GMDH	0.49	0.37	0.32	0.34	0.32	0.30	0.35
GRNN	0.38	0.42	0.45	0.41	0.32	0.49	0.48
GGAL	0.33	0.31	0.29	0.36	0.28	0.34	0.36

Table 6.10: Root Mean Square Error Values

dataset	Drumkit	OpenCV	Abdera	Ivy	Log4j	JEdit	JUnit
LR	0.66	0.56	0.61	0.58	0.49	0.42	0.47
M5Rule	0.72	0.38	0.30	0.37	0.41	0.56	0.63
DT	0.86	0.74	0.41	0.34	0.43	0.47	0.58
SVM	0.64	0.67	0.47	0.42	0.39	0.48	0.54
K Star	0.67	0.77	0.55	0.38	0.39	0.56	0.61
Bagging	0.78	0.83	0.59	0.39	0.47	0.59	0.43
JERN	0.53	0.54	0.41	0.59	0.47	0.39	0.48
BPN	0.42	0.58	0.61	0.33	0.41	0.42	0.51
Kohonen Network	0.56	0.48	0.37	0.40	0.35	0.43	0.52
PNN	0.38	0.42	0.33	0.53	0.43	0.53	0.41
GMDH	0.23	0.37	0.32	0.34	0.37	0.38	0.42
GRNN	0.38	0.42	0.45	0.41	0.32	0.49	0.47
GGAL	0.33	0.41	0.49	0.36	0.28	0.37	0.32

JEdit and JUnit datasets respectively. The accuracy of all the machine learning techniques w.r.t. MARE on all seven selected datasets lies between the ranges of 39-77% which is quite encouraging. Thus, it highlights the capability of machine learning technique for effective maintainability predictions of open source software.

6.3.2.3 Prediction Accuracy at 25% and 75%

Prediction accuracy of each classifier on each dataset is calculated at 25% as well as at 75% and the results are compiled in Tables 6.11 and 6.12.

Table 6.11: Results of Prediction Techniques at 25% Accuracy

S.No.	**dataset**	**Drumkit**	**OpenCV**	**Abdera**	**Ivy**	**Log4j**	**JEdit**	**JUnit**
1.	LR	62	58	56	61	63	59	54
2.	M5Rule	63	49	57	59	65	51	57
3.	DT	70	53	49	57	68	61	63
4.	SVM	65	47	51	52	73	42	49
5.	K Star	68	55	52	58	72	57	61
6.	Bagging	75	51	59	65	69	43	38
7.	JERN	68	69	72	62	74	61	52
8.	BPN	62	57	62	73	77	49	60
9.	Kohonen Network	67	63	53	48	73	58	52
10.	PNN	51	61	59	64	73	59	61
11.	GMDH	69	73	65	78	79	73	68
12.	GRNN	65	59	65	68	71	75	62
13.	GGAL	72	67	73	75	69	70	74

Table 6.12: Results of Prediction Techniques at 75% Accuracy

S.No.	**dataset**	**Drumkit**	**OpenCV**	**Abdera**	**Ivy**	**Log4j**	**JEdit**	**JUnit**
1.	LR	67	71	66	71	73	70	64
2.	M5 Rule	69	69	76	74	75	73	55
3.	DT	78	72	63	87	78	71	69
4.	SVM	76	67	72	82	81	64	66
5.	K Star	88	75	69	78	88	72	69
6.	Bagging	79	69	72	77	89	61	52
7.	JERN	88	73	74	71	81	69	64
8.	BPN	86	77	81	79	87	57	77
9.	Kohonen Network	87	79	83	69	81	65	72
10.	PNN	83	73	79	71	85	66	75
11.	GMDH	77	78	74	88	86	84	79
12.	GRNN	81	83	75	73	84	80	73
13.	GGAL	85	82	72	69	88	83	81

It also helps in determining whether the results are as per the criterion set by Conte et al. [50] and kitchenham et al. [110], that any prediction model is considered accurate if the value of (0.25) is less than Pred(0.75). As per the results presented in Table 6.9, 6.10, 6.11 and 6.12 for the respective values of MARE, RMSE, Pred(25%) and Pred(75%), we found that even though the prediction models for software maintainability usually attain less accuracy for example in many studies such as Bandi et al. [13], Briand et al. [34], Dagpinar and Jhanke [53], Stavrinoudis et al. [210], Elish [62], Fioravati and Nasi [68], Jin and Liu [100], Koten and Gray [118], Li and Henry [127], Lucia et al. [133], Misra [162], Thwin and Quah [219] and Zhou & Lung [237], however in the current study we found their reasonable values.

6.3.3 Validation of Hypotheses

To assess the outcome of machine learning techniques based prediction models, their prediction accuracy was measured through MARE, RMSE, Pred(0.25) and Pred(0.75). Their values were evaluated as per criterion set by previous researchers [50, 110] that any prediction model is considered accurate if its MARE values are less than 0.40 also the value of Pred(0.25) should always be greater than Pred(0.75).

6.3.3.1 H1 Hypothesis

When we analyzed the values of pred(0.25) and pred(0.75) from the Table 6.11 and 6.12, its recorded in the range of 72-78% for Pred(25%) and 66-89% for Pred(75%) which is quite reassuring that machine learning techniques are very effective. With respect to Pred(0.25), GMDH is found to be most accurate with Log4j dataset and JUnit dataset i.e. 79% accuracy. We also observe that GMDH method achieved more than 70% accuracy with four out of seven datasets. Similarly, if Pred(0.75) is taken as accuracy measure, Bagging is found to be most accurate with Log4j dataset i.e. 89% accuracy. GGAL is also found to be the best machine learning technique because more than 80% accuracy is achieved with five out of seven datasets at Pred(0.75). When we closely observe the range of accuracies, GGAL has performed outstandingly. Results are in the range of 67-75% across all datasets which are quite close to the criterion set by Kitchenham et al. [110] and Conte et al. [50].

Further, figure 6.1 and figure 6.2 depicts the MARE and RMSE values obtained by each machine learning technique on all the seven datasets. The figure 6.1 clearly shows that GMDH and GGAL machine learning techniques have performed highest over the all seven datasets. It's also evident from the figure 6.1 that minimum values recorded for MARE on all seven datasets were within the range of 0.28-0.36. Additionally, with respect to RMSE as depicted in figure 6.2, we found that mean values of RMSE range from 0.23-0.63. It is also observed that four machine learning techniques PNN, GMDH, GRNN and GGAL have achieved less than 30% error which is considered to be excellent.

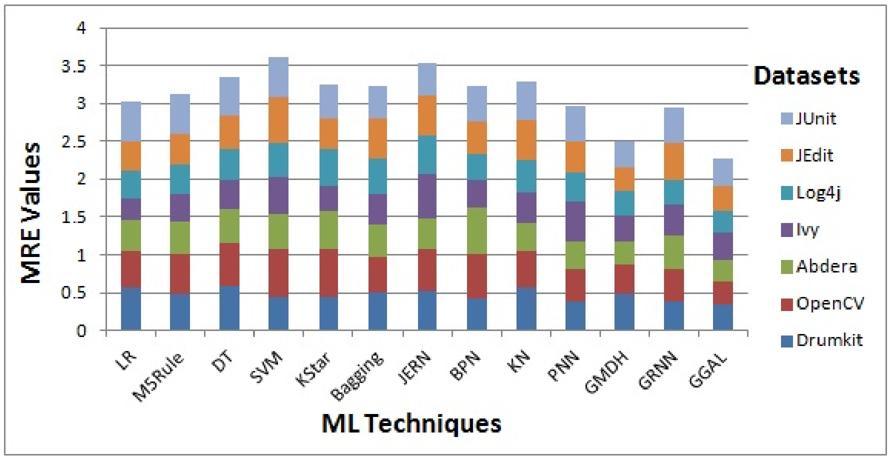

Figure 6.1: Mean Absolute Relative Error (MARE) Values of Machine Learning Technique on Corresponding datasets using 10-Fold Cross-Validation

On judging the overall performance of all machine learning techniques using four measures in the current chapter, it clearly satisfies the criteria laid down by [50, 110]. Hence we reject the NULL hypothesis, accept the alternate hypothesis and conclude that the impact of OO metrics on maintainability indeed exists in the perspective of open source software, thus machine learning techniques can be successfully applied for their maintainability prediction using OO metric suite.

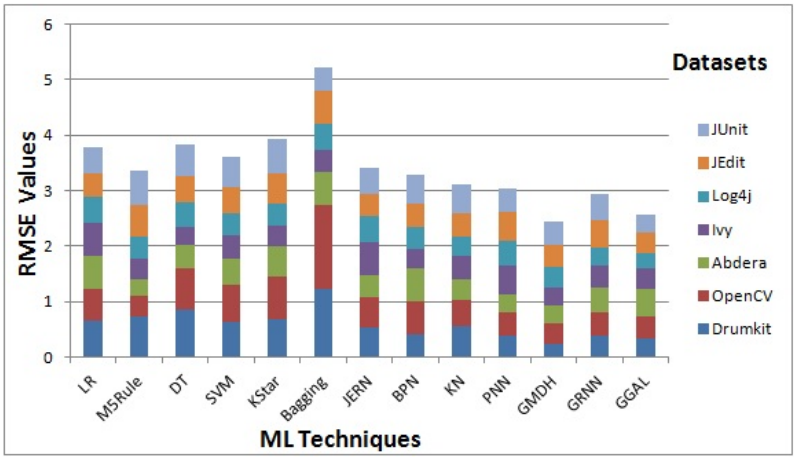

Figure 6.2: Root Mean Square Error (RMSE) Values of Machine Learning Technique on Corresponding datasets using 10-Fold Cross-Validation

6.3.3.2 H2 Hypothesis

We applied extensive statistical tests in order to check whether the performances of proposed machine learning techniques are significantly different or not. As per Demvsar [59], non-parametric tests are safer as they do not assume normal distribution or homogeneity of variance in the data. In the current investigation, Friedman test was used to compare the performance of thirteen machine learning techniques on seven datasets. We calculate the value of critical region at 5% significance level and degree of freedom twelve (for thirteen machine learning techniques). The value of $X_{(tabulated)}$ is obtained from Chi-square table where the degree of freedom is twelve (for thirteen machine learning techniques) at 95% level of significance.

The null hypothesis of the Friedman test states that there is no significant difference between the performance of machine learning techniques. We found that at significant level 0.05, calculated value of Friedman measure i.e. $X_{calculated}$ lies in the critical range for MARE as well as RMSE, hence, the Null hypothesis is rejected and alternative hypothesis is accepted and it is concluded that significant difference exists between the performances of participant machine learning techniques.

Further, in order to rank the performance of each of the machine learning technique, their FIR is calculated using equation 2.15 and compiled in Table 6.13 and Table 6.14, for MARE and RMSE respectively. As discussed earlier, lower the mean rank means better the performance. The outcome of the Friedman test using FIR for ranking as compiled in Table 6.13 with respect to MARE measure indicates that the performance of GGAL technique is the best and GMDH is the second best technique. With respect to RMSE from the 6.14, we observe that GMDH technique is the second best and GGAL as the best technique for the maintainability prediction of open source software on the basis of their mean rank.

Table 6.13: Mean Ranking of Machine Learning Techniques by Friedman Test on Mean Absolute Relative Error Value

S.No.	**Machine Learning Technique**	**Mean Rank**
1.	GGAL	1.79
2.	GMDH	2.71
3.	PNN	3.57
4.	GRNN	4.36
5.	LR	5.36
6.	BPN	6.64
7.	M5 Rule	7.14
8.	K Star	8.57
9.	Bagging	9.1
10.	Kohonen Network	9.43
11.	DT	10.6
12.	JERN	11.28
13.	SVM	11.57

In order to ascertain whether the performance differences which exist between FIR values of various machine learning techniques is statistically significant or not, we proceed towards post hoc analysis in RQ3.

6.3.3.3 H3 Hypothesis

In hypothesis H2, with the help of Friedman Test, we concluded that there exists a significant difference in the performance of machine learning technique; hence we proceed towards post hoc analysis using Nemenyi test to determine whether the difference is actually statistically significant between the performances of machine learning techniques or not.

Table 6.14: Mean Ranking of Machine Learning Techniques by Friedman Test on Root Mean Square Error Values

S.No.	**Machine Learning Technique**	**Mean Rank**
1.	GMDH	1.21
2.	GGAL	2.08
3.	GRNN	3.79
4.	Kohonen Network	4.14
5.	PNN	5.79
6.	BPN	7.28
7.	M5Rule	7.87
8.	JERN	8.32
9.	DT	8.5
10.	SVM	9.21
11.	LR	9.57
12.	K Star	10.36
13.	Bagging	11.58

The value of CD is calculated as 6.8 after putting the values of n as 13 (Number of machine learning techniques) and value of k as 7 (number of datasets) into equation (2.14). Next, we make a pair for each machine learning technique with every other to calculate their rank differences (FIR) and compiled in Table 6.15 and Table 6.16 for MARE and RMSE, respectively. In total, 78 such pairs were formed as we have used 13 machine learning techniques in our study.

Table 6.15: Computation of Pairwise Rank Difference Amongst all Machine Learning Techniques in Terms of Mean Absolute Relative Error

Tech	GGAL	GMDH	PNN	GRNN	LR	BPN	M5Rule	K Star	Bagging	Kohonen Net-work	DT	JERN	SVM
GGAL	–	0.92	1.78	2.57	3.57	4.85	5.35	6.78	7.31	7.64	8.81	9.49	9.78
GMDH		–	0.86	1.65	2.65	3.93	4.43	5.86	6.39	6.72	7.89	8.57	8.86
PNN			–	0.79	1.79	3.07	3.57	5.00	5.53	5.86	7.03	7.71	8.00
GRNN				–	1.00	2.28	2.78	4.21	4.74	5.07	6.24	6.92	7.21
LR					–	1.28	1.78	3.21	3.74	4.07	5.24	5.92	6.21
BPN						–	0.5	1.93	2.46	2.79	3.96	4.64	4.93
M5Rule							–	1.43	1.96	2.29	3.46	4.14	4.43
K Star								–	0.53	0.86	2.03	2.71	3.00
Bagging									–	0.33	1.5	2.18	2.47
Kohonen Network										–	1.17	1.85	2.14
DT											–	0.68	0.97
JERN												–	0.29
SVM													–

In the Table 6.15, we have highlighted those entries which have values greater than CD. It is quite evident that the out of 78 pairs of machine learning techniques, 13 pairs (bold entries) were found to have significant differences among their performances. One pair between GGAL and K Star has attained the difference (6.78) almost touching the CD value (6.8). Hence, 14 pairs out of 78 means, the performance of 17.9% of pairs was found to be significantly different using statistical test and not coincidental.

Table 6.16: Computation of Pair Wise Rank Difference Amongst all Machine Learning Techniques in Terms of Root Mean Square Error

Tech	GMDH	GGAL	GRNN	Kohonen Network	PNN	BPN	M5 Rule	JERN	DT	SVM	LR	K Star	Bagging
GMDH	–	0.87	2.58	2.93	4.58	6.07	6.66	7.11	7.29	8.00	8.36	9.15	10.37
GGAL		–	1.71	2.06	3.71	5.2	5.79	6.24	6.42	7.13	7.49	8.28	9.5
GRNN			–	0.35	2.00	3.49	4.08	4.53	4.71	5.42	5.78	6.57	7.79
Kohonen Network				–	1.65	3.14	3.73	4.18	4.36	5.07	5.43	6.22	7.44
PNN					–	1.69	2.08	2.53	2.71	3.42	3.78	4.57	5.79
BPN						–	0.59	1.04	1.22	1.93	2.29	3.08	4.3
M5Rule							–	0.45	0.63	1.34	1.7	2.49	3.71
JERN								–	0.18	0.89	1.25	2.04	3.26
DT									–	0.71	1.07	1.86	3.08
SVM										–	0.36	1.15	2.37
LR											–	0.79	2.01
K Star												–	1.22
Bagging													–

Results are shown in Table 6.15 also depicts that GGAL performed better than K Star, Bagging, Kohonen Network, DT, JERN and SVM whereas GMDH performed better than DT, JERN and SVM. Hence, on the basis of post hoc analysis of MARE, we conclude that GMDH and GGAL outperformed than other machine learning techniques. The difference between the performances of all other machine learning techniques were not found to be significant.

We performed the same procedure for RMSE and the rank difference of each pair was calculated and compiled in Table 6.16. Highlighted entries in the table indicate that the difference of FIR between that pair of machine learning technique is greater than CD. It is observed that out of 78 pairs of machine learning techniques, 12 pairs (bold entries) were found to have significant differences among their performances. So, with the help of Nemenyi Test conducted on RMSE measure, 12 pairs were found to be significantly different out of 78 pairs which is almost 15.3% of the total pairs.

It is also quite apparent that GMDH-JERN, GMDH-DT, GMDH-SVM, GMDH-LR, GMDH-K Star and the GMDH-Bagging pair were found to be significant as they have a value greater than CD. GGAL was found to be performing significantly superior to SVM, LR, K Star and Bagging. GRNN and Kohonen Network also performed better than bagging technique.

Hence, Hence we reject the NULL hypothesis, accept the alternate hypothesis and conclude that the difference in the performance of GGAL and GMDH were statistically different significantly as well as better than other machine learning techniques and the difference among the performance of all other machine learning techniques is not found to be significant.

6.4 Discussion

There are three broad categories of research papers working in the field of software maintainability of open source software. First category belongs to those papers which are finding if the open source software has better maintainability than closed source software. In the second category, research papers find how the maintainability evolves with every new version of the open source software. In the third category, researchers are investigating the relationships between design metrics and maintainability in the context of open source software. In this chapter which apparently lies in third category, we identified that the design metrics can be very useful for predicting maintainability of open source software. We investigated the relationships between a numbers of OO metrics and maintainability using large open source software systems. We built software maintainability prediction models based on OO metrics for open source software. Since few open source datasets are explored, hence this work can be repeated on other open source datasets in order to confirm or improve our findings.

The objective was to analyze the effectiveness of machine learning techniques for predicting software maintainability and the results are validated using dataset collected from open source software. An extensive empirical comparison of thirteen machine learning techniques on seven datasets obtained from open source code repositories is conducted. Prediction models were developed using seventeen most commonly used OO metrics. We further compared the performance of machine learning techniques using four prediction accuracy measures MARE, RMSE, Pred(0.25) and Pred(0.75). The variations amongst the performance of various machine learning techniques were further evaluated for significance using Friedman Test. Post hoc analysis using Nemenyi Test was also conducted to identify whether there exists the statistical difference of performance between the pair of different machine learning techniques. The main findings of the work are summarized below:

1. Feature sub-selection using GA is used and the relevant metrics are extracted for each dataset. It was found that in total 17%, 17%, 41%, 35%, 23%, 41% and 35% reductions were observed for Drumkit, OpenCV, Abdera, Ivy, Lo4j, JEdit and JUnit datasets respectively. On an average, it could successfully reduce the dimensions by almost 26.6%.

2. Out of the seventeen independent variables, we found that LCOM3, LOC, and DIT are

the most commonly selected OO metrics whereas Efferent coupling and Ce is found to be significant in only Abdera and Log4j Systems. We also found that the results obtained using reduced set of independent variables after applying FSS were better by 8.3% as compared to the results obtained using all independent variables in prediction models.

3. To measure the residual error, MARE and RMSE prediction accuracy was used and we found that GGAL and GMDH techniques perform better than other machine learning techniques.

4. The accuracy of all the machine learning techniques w.r.t. MARE on all seven selected datasets lies between the ranges of 39-77% which is quite encouraging.

5. When we analyzed the values of pred(0.25) and pred(0.75), its recorded in the range of 72-78% for Pred(25%) and 66-89% for Pred(75%) which is quite encouraging.

6. We also observe that GMDH method achieved more than 70% accuracy with four out of seven datasets. Similarly, if Pred(0.75) is taken as accuracy measure, Bagging is found to be most accurate with Log4j dataset i.e. 89% accuracy

7. We found that mean values of RMSE range from 0.23-0.63 and it is also observed that four machine learning techniques PNN, GMDH, GRNN and GGAL have achieved less than 30% error which is considered to be excellent.

8. The outcome of the Friedman test using FIR for ranking with respect to MARE measure indicates that the performance of GGAL technique is the best and GMDH is the second best technique. With respect to RMSE, we observe that GMDH technique is the second best and GGAL as the best technique for the maintainability prediction of open source software on the basis of their mean rank.

9. It is found that out of 78 pairs of machine learning techniques, 14 pairs were found

to have significant different among their performances. It means 17.9% of pairs was found to be significantly different using statistical test and not coincidental.

10. GGAL performed significantly better than K Star, Bagging, Kohonen Network, DT, JERN and SVM machine learning technique and GMDH performed significantly better than DT, JERN and SVM techniques.

11. With the help of post-hoc analysis using Nemenyi Test conducted on RMSE measure, 12 pairs were found to be significantly different out of 78 pairs which is almost 15.3% of the total pairs.

12. It is also quite apparent that GMDH-JERN, GMDH-DT, GMDH-SVM, GMDH-LR, GMDH-K Star and the GMDH-Bagging pair were found to be significant as they have value greater than CD.

13. GGAL was found to be performing significantly superior to SVM,LR, K Star and Bagging. GRNN and Kohonen Network also performed better than bagging technique.

14. The work presented in this chapter confirms that machine learning techniques have overall fare predictive ability as Pred(0.25) values are more than 60% in all cases. The superiority of GGAL and GMDH techniques over other machine learning techniques in the context of maintainability prediction of open source software was further confirmed by the results of Friedman test and post hoc analysis.

Chapter 7

Application of Evolutionary Techniques for Software Maintainability Prediction using Object-Oriented Metrics

7.1 Introduction

This study was undertaken with a view of applying evolutionary techniques in designing prediction model for software maintainability which is a very important software quality attribute. The task of is not as simple as it seems due to the subjective nature of software maintainability. There is the pressing demand for more and more precise software maintainability prediction models so that the resource planning can be optimized well in advance to produce cost effective software systems. The significance of the evolutionary technique has substantially increased in recent time due to their capability of maximizing the quality function. Inspired by the evolutionary process we have conducted an empirical study for exploring the application of the evolutionary technique for software maintainability prediction. Although several traditional methods such as statistical and machine learning were applied in past, we experimented to apply the evolutionary technique for the first time in the current study and compared their performance with traditional ones.

Five open source software projects viz Apache Poi 3.9, Apache Rave 0.21.1, OrDrumBox 0.6.5, HuDoKu 2.0 and JWebUnit 1.2 written in Java languages were used to carry out this empirical investigation and the results were analyzed using prevalent prediction accuracy measures. Although, evolutionary techniques are successfully applied in various another discipline of software engineering [12, 16, 213, 222], however there is a need to evaluate its performance in predicting software maintainability. Few significant metrics are also successfully identified in this study which can be used by software maintainability practitioner in the early phases of software development to predict those classes which needs more maintenance efforts.

The proposed prediction model may also be used as a quality benchmark to assess and compare various software maintainability prediction models. The results showed that the proposed evolutionary techniques can be used to achieve good accuracy while predicting maintainability. In this chapter, we investigate the following issues:

1. How accurately and precisely do the OO metrics predict the maintainability of open source software?
2. Can evolutionary techniques be used for software maintainability prediction?
3. How accurately and precisely do the evolutionary techniques predict maintainability of open source software using OO metrics?
4. Is the performance of evolutionary techniques is better than traditional machine learning techniques and statistical techniques?

The chapter is organized as follows: Section 7.2 depicts the evolutionary techniques, their advantages and classification. Section 7.3 summarizes the research methodology consisting of OO metrics, empirical data collection, setting values of the parameters and hypothesis to be tested in the study. The results of the study are given in section 7.4 followed by the validation of the hypothesis. Finally, discussion of the study is presented in section 7.5.

The results of this chapter have been reported in [139].

7.2 Evolutionary Techniques

The evolutionary techniques are the set of techniques inspired by the metaphor of natural biological evolution such as ant-colony optimization, bees techniques, cuckoo techniques, and particle swarm optimization etc [7, 11, 125, 159]. As shown in figure 7.1, certain operators are applied on potential solutions to produce better and better approximations in these techniques. Each time, at each generation various operators such as selection, recombination, mutation, migration, locality and neighborhood are applied to produce the next generation [125]. Each individual solution of the next generation is calculated against the survival of the fittest and the unqualified solutions are discarded [11]. When this process is repeated again and again, it leads to the evolution of populations consisting of potential solutions which are optimized and better suited to their environment.

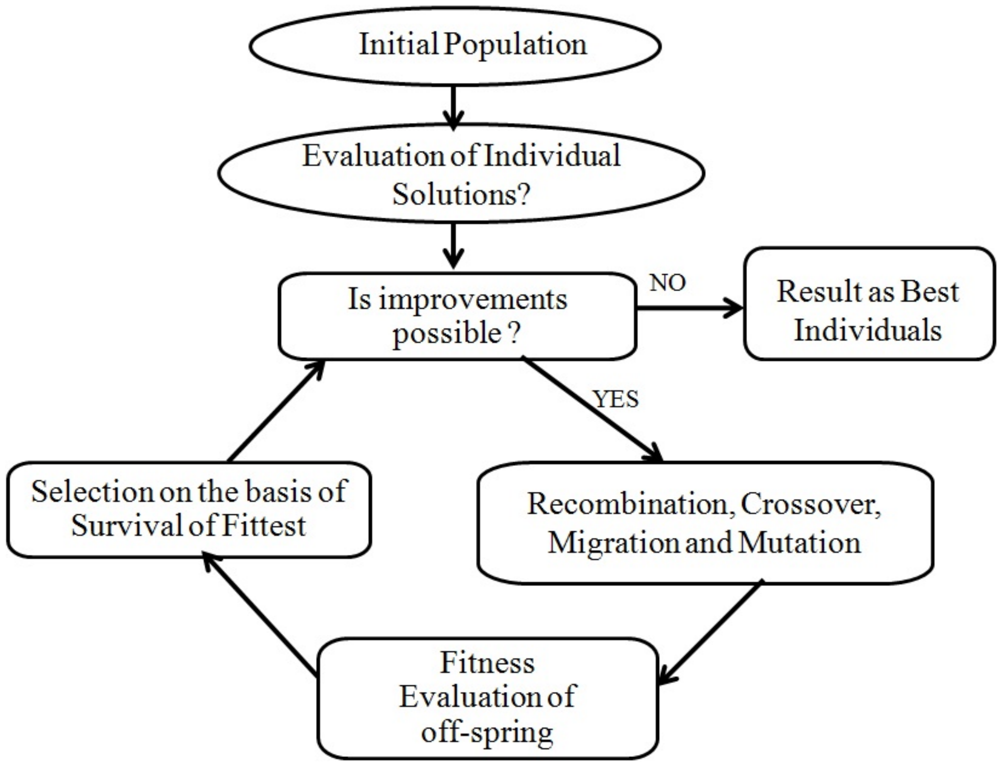
Initial Population
Evaluation of Individual Solutions?
Is improvements possible ?
NO
Result as Best Individuals
YES
Selection on the basis of Survival of Fittest
Recombination, Crossover, Migration and Mutation
Fitness Evaluation of off-spring

Figure 7.1: Architecture of Evolutionary Techniques

When this process is repeated over time, only the better-fit individuals survived; hence the evolutionary techniques are also called as function optimizer. While implementing evolutionary techniques we first create a population with or without fixed size; First time usually, this population is randomly generated. Each individual of this population is then tested against 'fit function'. Reproductive opportunities are given to those individuals who have a better solution to the target problem and they have better chances of survival. Those individual solutions of the populations which are poorer and produce 'weaker' solutions, they have fewer chances of survival. The 'goodness' of a solution is defined in terms of the problem which needs to be solved. While solving any issue using evolutionary techniques, researchers first break the given issue into two problems i.e. the encoding problem and the evaluation problem. Evolutionary technique are found to be superior to traditional methods due to various aspects listed as under:

- Evolutionary techniques are generally more straightforward to apply because there are no restrictions on the definition of the objective function.

- Use of evolutionary technique removes the possibility of biasness and the results are only influenced by objective function and fitness function and as such there is no requirement of auxiliary knowledge.

- Search for an optimized solution is performed in a parallel manner and evolutionary technique provides a number of potential solutions to a given problem with final choice lies with the user.

- They can handle a large amount of noise present in the data as the transition rules are probabilistic in nature and not deterministic in nature.

- They are more capable of working in large and discontinuous search space and able to achieve global optima instead of local ones.

- Evolutionary techniques are generally more straightforward to apply because no restrictions for the definition of the objective function exist.

7.2.1 Classification of Techniques

In this section classification of various machine learning techniques used in this empirical study are presented. In the current study, we have identified a set of 14 techniques divided into three major categories as shown in figure 7.2.

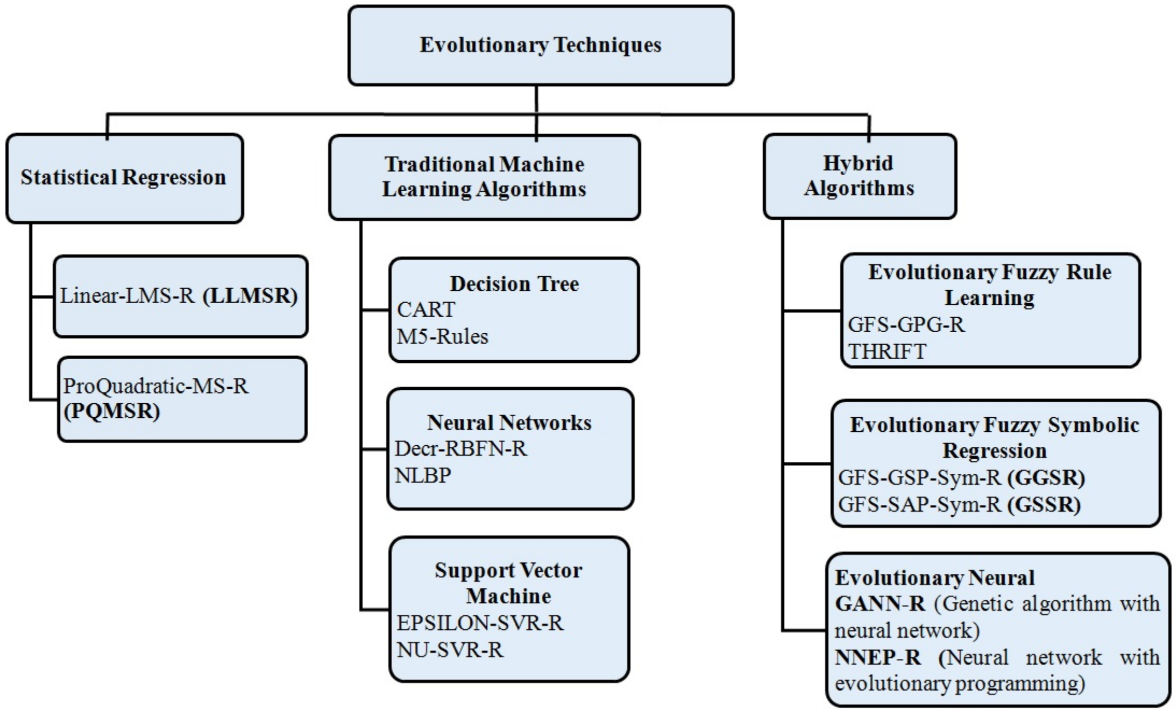

Figure 7.2: Categories of Evolutionary Techniques

A set of 14 techniques including 2 statistical, 6 machine learning, and 6 evolutionary based techniques have been carefully selected to compare their performance on the diverse platform using the accuracy measures proposed by Conte et al. [50], Fentom and Bieman [65], and Kitchenham et al. [110]. The prediction model was developed using the KEEL tool (http://www.keel.es) and we performed some initial experiments to adjudge the effectiveness of the evolutionary technique against the traditional techniques. Each technique is explained in brief as under:

- **Linear-LMS-R**: It is an adaptive technique which follows an iterative procedure that makes successive corrections to the weight vector in the direction of the negative of the gradient vector. Continuous improvement in each iteration eventually leads to the minimum mean square error [189].

- **ProQuardratic-MS-R:** In this technique, based on the quantitative information present in the individual terms, terms are placed in respective groups for classifications [189].

- **CART**: Classification and Regression Trees (CART) are machine learning methods for constructing prediction models obtained by recursively partitioning the data space and fitting a simple prediction model within each partition [33]

- **M5-Rules:** M5 build tree-based models like CART but unlike CART, here the tree can have a multivariate linear model to tackle high dimensionality up to 100 attributes [33].

- **Decr-RBFN-R:** Generalization in terms of interpolation between known points is created for such networks. Thus RBFN-R model represents the non-linear relationship with guaranteed learning rules [36].

- **Non Linear Boosting Projections:** In NLBP approach, instead of random space, constructive non linear projections are created using neural networks and further combined with the philosophy of boosting to handle noise present in the data [74].

- **EPSILON-SVR-R:** In this method, first order approximation of the objective function is used to achieve faster convergence during the working set selection for training SVM [64].

- **NU-SVR-R:** Instead of first order information, this method uses the second order information to achieve faster convergence for working set selection while training the SVM [64].

- **GFS-GPG-R:** It combines genetic programming and genetic techniques to solve symbolic regression problems and applied to find an analytic expression in order to relate input variables with output variables [193].

- **THRIFT:** It was given by Philip Thrift [218] and in this technique, the discrete nature of fuzzy strategies are used during the discovery process by genetic techniques .

- **GFS-GAP-Sym-R:** Designed especially for electrical engineering problems, in this technique, a fuzzy arithmetic-based GA-P procedure is applied to the search of an analytic expression that relates input and output variables [193].

- **GFS-SAP-Sym-R :** It is a Symbolic Fuzzy-Valued Data Learning based on Genetic Programming Grammar Operators and Simulated Annealing [193].

- **GANN-R:** It stands for Genetic technique with Neural Network. When neural network is combined with genetic techniques, convergence characteristics degrade significantly as the size of network increases. In this technique graph grammatical encoding is used to encode the chromosome in order to generate more regular connective patterns [159].

- **NNEP-R:** It stands for Neural Network Evolutionary Programming for Classification. In this technique basis function, units of FFNN model is evolved both in terms of weight and structure using evolutionary programming technique which leads to overall performance gain for real world high order functions [153].

Although evolutionary techniques have been used successfully for predictions in many fields such as structure prediction of molecular crystals [73], Prediction of protein contact map [223] etc however to the best of authors knowledge yet not applied in any study for maintainability prediction. In the current study, these excellent bio-inspired techniques have been used for the first time for software maintainability prediction using software design characteristics of five open source software systems.

7.3 Research Methodology

This section is further divided in to sub-sections wherein we begin our empirical study by setting up the goal and hypothesis. The summary of the OO metrics used to measure various characteristics of software is presented in next sub section followed by the details of the empirical data collection process.

7.3.1 Hypotheses

The goal of this study is to evaluate the performance of the evolutionary technique with the traditional statistical and machine learning methods and compare their prediction accuracies. In this section, research hypotheses are presented as follows:

7.3.1.1 H1 Hypothesis

- Null Hypothesis: The relationship between design metrics and subsequent maintainability does not exist.
- Alternate Hypothesis: The relationship between design metrics and subsequent maintainability significantly exist.

7.3.1.2 H2 Hypothesis

- Null Hypothesis: The evolutionary techniques can not be applied for software maintainability prediction.
- Alternate Hypothesis: The evolutionary techniques can be applied for software maintainability prediction.

7.3.1.3 H3 Hypothesis

- Null Hypothesis: Evolutionary techniques do not outperform the models predicted using statistical and machine learning techniques.
- Alternate Hypothesis: Evolutionary techniques significantly outperform the models predicted using statistical and machine learning techniques.

7.3.2 Independent and Dependent Variables

We have investigated the prediction capability of the evolutionary technique on datasets collected from five open sources software system. An attempt has been made to test whether the OO software design metrics measured at development time could be used for the prediction of software maintainability. Various characteristics of open source software metric suites were measured using Chidamber & Kemerer [43] metric suite. To measure the OO features present in the software, we determine the amount of coupling, cohesion and inheritance. Coupling of the class is measured through RFC and CBO, inheritance is measure through DIT and NOC, cohesion is measure through LCOM. Further, size is measured through LOC and complexity is measure through WMC. Detail definitions of independent variables are summarized in chapter 2. The dependent variable is the maintenance effort as defined in chapter 2. In order to determine the degree of maintainability of the software on the basis of its design metrics, the evolutionary techniques were applied.

7.3.3 Empirical Data Collection

In the current study five open source software namely Apache Poi 3.9, Apache Rave 0.21.1, OrDrumBox 0.6.5, HuDoKu 2.0 and JWebUnit 1.2 were analyzed as summarized in Table 7.1.

Table 7.1: Details of the Data Points for Open Source Software Systems

Name of the Software	Version	Release Date	Total Number of Common Classes	No. of Change in Class	Percentage of Changes	LOC of Common Classes	LOC Changed	LOC Delete	LOC Added
Apache	3.9	28 Nov 2013	940	919 changed	97%	2080940	725407	453942	271465
Poi	3.10	01 Feb 2014		21 not changed					
Apache	0.21.1	02 May 2013	672	222 classes changed	33%	14115	6105	2089	4016
Rave	0.22	09 July 2013		440 no change					
OrDrumBox	0.6.5	24 Aug 2006	218	106 changed	48%	1987602	573902	5392	271465
	0.9.8	07 Jan 2012		112 not changed					
HuDoKu	2.0	01 Apr 2010	245	102 classes changed	41%	674502	392143	546682	282931
	2.2	01 Aug 2012		143 no change					
JWebUnit	1.2	06 July 2009	230	172 classes changed	74%	54320	4239	3160	10398
	3.0	08 Oct 2015		58 no change					

7.3.4 Descriptive Statistics

For the purpose of the qualitative analysis, the descriptive statistics are calculated from the collected data for each of the selected software in the current study, Outliers were removed by taking 95 percentile of each metrics followed by calculations of descriptive statistics. The Mean, Median, Standard Deviation, Minimum and Maximum for all the Chidamber and Kemerer metrics were calculated and presented in Table 7.2, Table 7.3, Table 7.4, Table 7.5, Table 7.6 for Apache Poi 3.9, Apache Rave 0.21.1, OrDrumBox 0.6.5, HuDoKu 2.0 and JWebUnit 1.2 respectively. It is very useful in understanding and comparing the characteristics of both OO systems. During this process 21 classes of Poi, 39 classes of Rave, 27 classes for OrDrumBox, 46 classes for HuDoKu and 58 classes for JWebUnit datasets were removed.

Table 7.2: Descriptive Statistics for Apache POI 3.9

Name of the Metric	**Mean**	**Median**	**Std Dev**	**Max**	**Min**
WMC	9.72	72	12.67	142	0
DIT	1.01	3	0.332	4	0
NOC	0.02	1	0.192	3	0
CBO	2.35	17	1.290	27	0
RFC	10.69	69	12.91	143	0
LCOM	29.83	38	42.18	127	0
LOC	155.67	389	73.39	858	27
CHANGE	0.62	72	34.13	470	0

Table 7.3: Descriptive Statistics for Apache Rave 0.21.1

Name of the Metric	**Mean**	**Median**	**Std Dev**	**Max**	**Min**
WMC	11.48	118	14.44	165	6
DIT	0.62	4	0.701	5	0
NOC	0.45	12	5.07	151	1
CBO	9.08	7	21.39	38	0
RFC	27.45	35	33.34	426	0
LCOM	39.87	41	48.53	208	0
LOC	224.51	849	368.32	4455	12
CHANGE	7.56	87	121.27	956	0

Following observations were made from the descriptive statistics:

Table 7.4: Descriptive Statistics for OrDrumBox 0.6.5

Name of the Metric	Mean	Median	Std Dev	Max	Min
WMC	8.68	57	20.09	93	0
DIT	2.13	2	0.832	4	0
NOC	0.04	2	0.168	2	0
CBO	13.72	21	22.08	42	0
RFC	9.52	58	11.39	139	0
LCOM	33.41	38	36.78	281	0
LOC	143.32	278	53.07	780	21
CHANGE	0.59	73	37.52	338	0

Table 7.5: Descriptive Statistics for HuDoKu 2.0

Name of the Metric	Mean	Median	Std Dev	Max	Min
WMC	8.53	61	13.17	205	0
DIT	2.34	4	0.897	5	0
NOC	0.89	2	0.439	6	0
CBO	16.92	41	16.62	18	0
RFC	13.09	49	13.73	256	0
LCOM	19.43	28	37.18	209	0
LOC	159.73	423	88.73	791	53
CHANGE	1.41	83	41.93	408	0

Table 7.6: Descriptive Statistics for JWebUnit 1.2

Name of the Metric	Mean	Median	Std Dev	Max	Min
WMC	7.63	43	8.76	34	0
DIT	0.93	2	0.74	3	0
NOC	0.09	1	0.168	2	0
CBO	11.35	16	2.79	21	0
RFC	9.88	32	9.12	131	0
LCOM	9.17	24	34.09	97	0
LOC	113.57	254	53.55	7658	39
CHANGE	0.51	59	29.63	308	0

- Size measured in terms of lines of source code i.e. LOC is ranging from 0 to 4455 for all selected software.

- The mean values of DIT and NOC are Poi (1.01, 0.02), Rave (0.62, 0.45), OrDrumBox (2.13, 0.04), HuDoKu (2.34, 0.89) and JWebUnit (0.93, 0.09) which means inheritance is comparatively less exploited in all the systems. The median of DIT for all software is more than 0 which means that at least more than half of the classes have a parent class. Thus, inheritance is widely used in these selected software.

- Cohesion which is measured through LCOM have high mean values. Maximum value of LCOM have recorded as 127 (for Poi), 208 (for Rave), 281 (for OrDrumBox), 209 (for HuDoKu) and (2.34, 0.89) and 97 (for JWebUnit). Since the values are either approximately 100 or more, that means cohesion is high in both the software systems.
- The value of coupling is measured through CBO and RFC which is found to be notably less. Its minimum values are 0 for all the datasets. Maximum values are recorded as 27 and 143 (for Poi), 38 and 426 (for Rave), 42 and 139 (for OrDrumBox), 18 and 256 (for HuDoKu), 21 and 131 (JWebUnit) are recorded. Since the value of the CBO metric which is used to measure the interaction between the classes is high for all the software, it indicates that there is a high interaction between classes.
- WMC is used for comparing the complexity characteristics between two software. It was observed that its value for Rave is 11.48, for Poi its value is 9.72, for OrDrumBox its value is 8.68, for HuDoKu its value is 8.53 and for JWebUnit its value is 7.63, that means Rave software is more complex.
- Since the characteristics of all datasets are heterogeneous in nature, hence they cannot be combined and they have to be considered separately while building software maintainability prediction models.

7.3.5 Parameter Setting for Evolutionary Techniques

The evolutionary technique is bio-inspired computer technique and it mimics the natural evolution of living organisms. Parameters shown in Table 7.7 are used for evolution strategy in order to decide how to proceed for the next generation. The evolution continues until either a good enough optimized solution is found or we reached the maximum number of generations allowed.

Table 7.7: Parameters Setup for Experiments

S.No.	Technique	Parameter Value
1.	CART	MaxDepth = 90
2.	M5-Rules	Prunning Factor = 2, Verbosity = 0, Heuristic = Coverage
3.	Decr-RBFN-R	Percent = 0.1, Neurons = 20, alpha = 0.3
4.	NLBP	Hidden-layers =2, Hidden-nodes = 15, Transfer = Htan, Eta = 0.15, Alpha = 0.10, Lambda = 0.0, cycle = 10000, improve = 0.01, verbose = false, Tipify-input = true, ensemble method = BEM, combination = WeightedSum, Network = 10
5.	EPSILON-SVR-R	Kernal = RBF, eps = 0.001, Degree = 3, Gamma = 0.001, coef() = 0.0, nu = 0.5, p=1.0, Shrinking = 0
6.	NU-SVR-R	Kernal = RBF, eps = 0.001, Degree = 1, Gamma = 0.001, coef() = 0.0, nu = 0.1, p=1.0, Shrinking = 0
7.	GFS-GPG-R	Numlabels = 3, numrules = 8, popsize = 30, numisland = 2, steady = 1, numitera = 10000, toursize=4, probmuta = 0.1, amplmuta = 0.1, probmigra = 0.001, proboptimlocal = 0.00, numoptimlocal = 0, idoptimlocal = 0, probmutaga = 0.5, maxtreeheight = 8
8.	THRIFT	Number of labels = 3, population size = 61, number of evaluation = 10000, Crossover Probability 0.6, Mutation Probability 0.1
9.	GFS-GSP-Sym-R	Popsize = 30, numisland = 2, steady = 1, numitera = 10000, toursize = 4, probmuta = 0.01, amplmuta = 0.1, probmigra = 0.001, maxtreeheight = 8
10.	GFS-SAP-Sym-R	Deltafitsap = 0.5, posap = 0.5, p1sap = 0.5, amplmuta = 0.1, nsubsap = 10, proboptimlocal = 0.00, numoptimlocal = 0, idoptimlocal = 0, probcrossga = 0.5, probmutaga = 0.5, maxtreeheight = 8
11.	GANN-R	Hidden-layers = 2, Hidden-nodes = 15, Transfer = Htan, Eta = 0.15, Alpha = 0.10, Lambda = 0.0, Test-data = true, Validation-data = false, BP-Cycle = 10000, Improve = 0.01, Tipify-inputs = true, Verbose = true, Elite = 0.1, individual = 100, W-range = 5.0, connectivity = 0.5, max-generations = 100
12.	NNEP-R	Hidden-nodes = 4, Transfer = Product-Unit, Generations = 1000

7.3.6 Prediction Accuracy Measures

After obtaining the results, we analyzed their performances using various prediction accuracy measures given by Conte et al. [50], Fentom and Bieman [65] and Kitchenham [110]. Most commonly used accuracy measures are applied in this chapter such as MaxMRE, MMRE and prediction accuracy at 25% and 30%. Their detail definition and formula of each measure are given in chapter 2.

7.4 Results and Discussion

In this section, results are presented for all the techniques on five selected software systems along with their interpretations. In the next sub-section results of various prevalent accuracy measures are presented which are used to adjudge the various techniques and in the last sub-section based on the results, validations of the hypothesis are conducted.

7.4.1 Feature Sub Selection

For the purpose of dimensionality reduction, feature sub selection was performed in the beginning as discussed in chapter 2. Table 7.8 summarizes the relevant metrics found after applying FSS for all of the five datasets selected in the current study. In total 21%, 19%, 29%, 13%, 19% reductions were observed for Poi, Rave, OrDrumBox, HuDoKu and JWebUnit datasets respectively. On an average approximately 20% saving is observed for all datasets.

Table 7.8: Metrics Obtained using Feature Sub-Selection using Genetic Algorithm Technique

Software Name	**Selected Relevant OO Attributes**
Poi	WMC, RFC, DIT, LCOM
Rave	CBO, DIT, LCOM
OrDrumBox	WMC, DIT, LOC, LCOM
HuDoKu	LCOM, LOC, RFC, CBO
JWebUnit	LCOM, DIT, CBO

When a minimal optimized subset of attributes is selected using features subset selection method, it enhances the prediction accuracy and reduces the time taken by the model on account of training of the prediction model.

7.4.2 Summary of Results for Various Prediction Accuracy Measures

This section presents the prediction results of various evolutionary techniques and their comparison with traditional statistical and machine learning techniques. We present the results of various accuracy measures achieved by applying software maintainability prediction modeling techniques for Apache Poi, Apache Rave, OrDrumBox, HuDoKu and JWebUnit as compiled in Tables 7.9, Table 7.10, Table 7.11, Table 7.12 and Table 7.13 respectively. The first column represents the category, the second column represents the name of the technique, the third column represents the maximum value of MRE, the fourth column represents MMRE, the fifth column represents prediction accuracy at 25% and sixth column represents accuracy at 30%.

Table 7.9: Results of Various Prediction Techniques on Apache Poi dataset

Category of the Technique	**Name of the Technique**	**Max MRE**	**MMRE**	**Pred (0.25)**	**Pred (0.30)**
Statistical Regression	Linear-LMS-R	12.94	1.511	0.34	0.38
	ProQuardratic-MS-R	16.83	1.003	0.23	0.29
Decision Tree	CART	14.27	1.007	0.42	0.48
	M5-Rules	20.69	0.856	0.52	0.55
Neural Networks	Decr-RBFN	10.96	0.617	0.49	0.57
	NLBP	15.65	0.557	0.37	0.44
SVM	EPSILON-SVR-R	8.28	0.739	0.29	0.36
	NU-SVR-R	6.76	0.619	0.41	0.48
Evolutionary Fuzzy	GFS-GPG-R	3.67	0.246	0.59	0.67
	THRIFT	6.89	0.224	0.64	0.69
Fuzzy Symbolic Regression	GFS-GAP-Sym-R	5.73	0.396	0.52	0.58
	GFS-SAP-Sym-R	6.88	0.422	0.57	0.61
Evolutionary Neural	GANN-R	4.56	0.364	0.46	0.49
	NNEP-R	6.37	0.257	0.51	0.58

7.4.3 Validation of Hypotheses

The purpose of conducting any empirical study is essentially to obtain unbiased results which can be generalized and their interpretation can be stored and applied on future release. Hence, it is very vital that the model should be properly validated on the dataset which is different from the training dataset. As discussed in chapter 2, in this research 10-fold cross-validation is used which divides the data into 10-folds and each time nine parts are used for training whereas one part is used for validation purpose. In order to statistically analyze the

Table 7.10: Results of Various Prediction Techniques on Apache Rave dataset

Category of the Technique	Name of the Technique	Max MRE	MMRE	Pred (0.25)	Pred (0.30)
Statistical Regression	Linear-LMS-R	14.63	1.047	0.42	0.43
	ProQuardratic-MS-R	15.89	0.809	0.51	0.59
Decision Tree	CART	13.73	0.814	0.52	0.6
	M5-Rules	10.98	0.758	0.39	0.45
Neural Networks	Decr-RBFN	11.96	0.791	0.41	0.48
	NLBP	13.92	0.845	0.47	0.52
SVM	EPSILON-SVR-R	16.82	0.692	0.55	0.59
	NU-SVR-R	15.49	0.54	0.49	0.6
Evolutionary Fuzzy	GFS-GPG-R	7.02	0.251	0.59	0.59
	THRIFT	8.06	0.238	0.63	0.66
Fuzzy Symbolic Regression	GFS-GAP-Sym-R	8.78	0.434	0.43	0.49
	GFS-SAP-Sym-R	9.68	0.354	0.45	0.48
Evolutionary Neural	GANN-R	7.81	0.362	0.53	0.51
	NNEP-R	8.22	0.395	0.55	0.59

results, we use non-parametric tests i.e. Friedman test followed by post hoc Nemenyi test to compare the efficiency of various machine learning techniques. This section validate the various hypothesis stated in section 7.3.1.

7.4.3.1 H1 Hypothesis

The first hypothesis was stated to identify the relationship between design metrics and subsequent maintainability. We divided the data into 3:1 ratio between training and testing respectively which is commonly accepted proportion [219]. The value of MMRE represents the goodness of fit of the proposed models. From the Table 7.9 to Table 7.13, it is quite evident that the values of MMRE are significantly better for all the datasets. When we calculated the average MMRE values of all 14 techniques selected in the current empirical study, its value is 0.629, 0.595, 0.486, 0.473 and 0.523 for Poi, Rave, OrDrumBox, HudoKu and JWebUnit respectively. The values of MMRE were quite competitive as per the standards of accuracy measurements set by Kitchenham [110] and followed by researcher's community.

Table 7.11: Results of Various Prediction Techniques on OrDrumBox dataset

Category of the Technique	Name of the Technique	Max MRE	MMRE	Pred (0.25)	Pred (0.30)
Statistical Regression	Linear-LMS-R ProQuardratic-MS-R	11.59 10.92	0.937 0.813	0.51 0.38	0.59 0.43
Decision Tree	CART M5-Rules	9.73 12.34	0.753 0.837	0.44 0.41	0.48 0.49
Neural Networks	Decr-RBFN NLBP	13.47 9.89	0.913 0.743	0.46 0.49	0.53 0.52
SVM	EPSILON-SVR-R NU-SVR-R	15.47 12.73	0.903 0.635	0.52 0.54	0.56 0.59
Evolutionary Fuzzy	GFS-GPG-R THRIFT	11.27 7.86	0.348 0.372	0.67 0.61	0.72 0.68
Fuzzy Symbolic Regression	GFS-GAP-Sym-R GFS-SAP-Sym-R	9.23 8.43	0.402 0.392	0.59 0.55	0.68 0.61
Evolutionary Neural	GANN-R NNEP-R	9.72 8.39	0.782 0.621	0.56 0.49	0.63 0.56

This clearly shows a high degree of relationship between design metrics and maintainability. Hence, we accept the alternate hypothesis and claimed that there exist a strong relationship between OO metrics and maintainability using evolutionary techniques.

7.4.3.2 H2 Hypothesis

The second hypothesis was stated to check whether evolutionary techniques can be applied for software maintainability prediction. In order to check the capabilities of evolutionary techniques for maintainability prediction, the MMRE values of each of the prediction technique on all dataset was compiled and visually represented in figure 7.3. It was quite evident that from the graph that THRIFT under the category of Evolutionary Fuzzy technique was found to be most accurate in predicting software maintainability.

The same process is repeated for another accuracy measure MaxMMRE on both of the datasets and visually presented in figure 7.4. In terms of MaxMRE measure, the GFS-GPG-R technique of evolutionary fuzzy technique category was found to be more accurate than

Table 7.12: Results of Various Prediction Techniques on HuDoKu dataset

Category of the Technique	Name of the Technique	Max MRE	MMRE	Pred (0.25)	Pred (0.30)
Statistical Regression	Linear-LMS-R	11.53	0.918	0.45	0.48
	ProQuardratic-MS-R	9.78	0.813	0.39	0.49
Decision Tree	CART	6.73	0.872	0.37	0.41
	M5-Rules	12.43	0.811	0.24	0.44
Neural Networks	Decr-RBFN	10.72	0.792	0.43	0.5
	NLBP	9.87	0.853	0.48	0.49
SVM	EPSILON-SVR-R	13.47	0.779	0.46	0.53
	NU-SVR-R	10.92	0.562	0.39	0.44
Evolutionary Fuzzy	GFS-GPG-R	6.08	0.341	0.61	0.66
	THRIFT	7.89	0.451	0.58	0.62
Fuzzy Symbolic Regression	GFS-GAP-Sym-R	9.21	0.782	0.55	0.59
	GFS-SAP-Sym-R	10.92	0.984	0.49	0.53
Evolutionary Neural	GANN-R	12.34	0.781	0.46	0.49
	NNEP-R	11.04	0.709	0.42	0.48

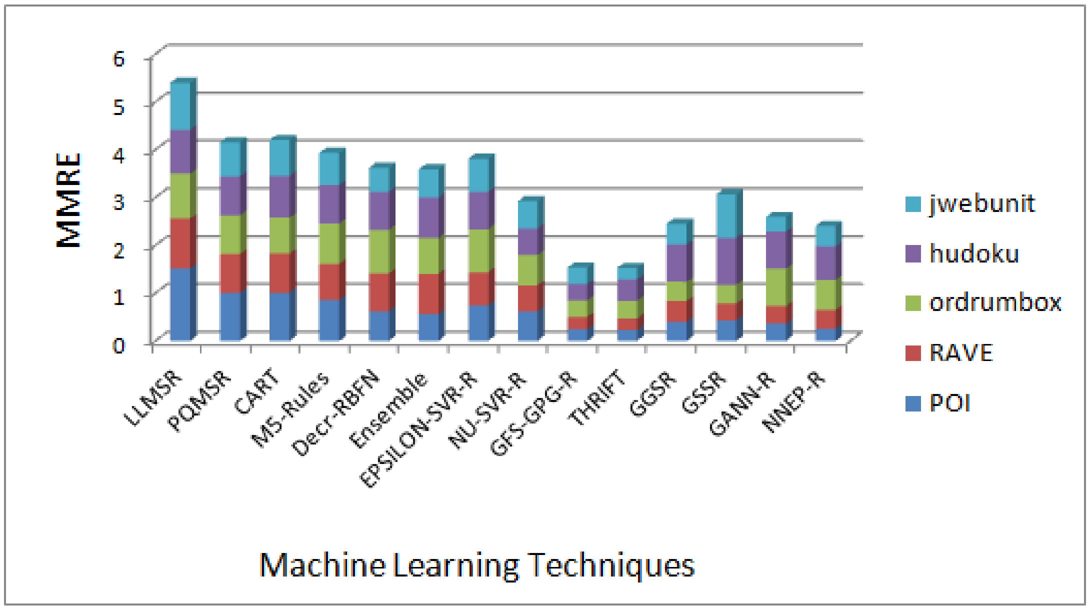

Figure 7.3: Comparison of Various Models with Reference to their MMRE Values

Table 7.13: Results of Various Prediction Techniques on JWebUnit dataset

Category of the Technique	Name of the Technique	Max MRE	MMRE	Pred (0.25)	Pred (0.30)
Statistical Regression	Linear-LMS-R	8.79	0.986	0.57	0.59
	ProQuardratic-MS-R	8.34	0.713	0.61	0.66
Decision Tree	CART	9.3	0.756	0.48	0.52
	M5-Rules	11.09	0.673	0.59	0.63
Neural Networks	Decr-RBFN	13.42	0.509	0.49	0.52
	NLBP	12.44	0.593	0.55	0.58
SVM	EPSILON-SVR-R	10.91	0.692	0.51	0.54
	NU-SVR-R	11.93	0.567	0.46	0.52
Evolutionary Fuzzy	GFS-GPG-R	8.02	0.345	0.63	0.66
	THRIFT	7.73	0.238	0.66	0.71
Fuzzy Symbolic Regression	GFS-GAP-Sym-R	9.98	0.432	0.48	0.52
	GFS-SAP-Sym-R	8.23	0.912	0.58	0.63
Evolutionary Neural	GANN-R	8.29	0.302	0.59	0.65
	NNEP-R	9.13	0.419	0.41	0.49

others techniques. When Pred(0.25) is used as an indicator, both techniques under the category of evolutionary fuzzy technique were found to be the most accurate. Thus, from the results, it is evident that evolutionary fuzzy technique can be used for more precise maintainability predictions. We observed that the evolutionary technique achieved the optimization values more accurately and precisely than the traditional models when they were used for software maintainability prediction. Thus, we accept the null hypothesis and conclude that the evolutionary technique based models developed in the current study can be successfully applied for accurate software maintainability prediction during early phases of the software development cycle.

7.4.3.3 H3 Hypothesis

Third hypothesis checked if the evolutionary techniques perform significantly better or worse than traditional statistical and machine learning methods. In order to validate this hypothesis, we performed extensive statistical tests using Friedman test and post-hoc analysis

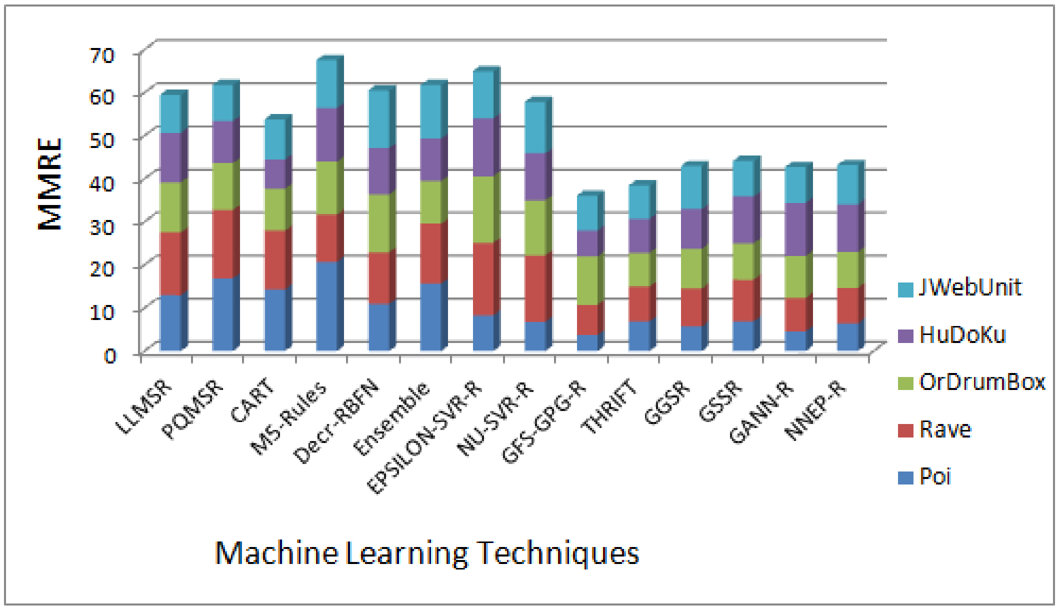

Figure 7.4: Comparison of Various Models with Reference to their Max MRE Values

using Nemenyi test described in chapter 2.

We followed the suggestions given by Demvsar [59] that if the data does not follow normal distributions, its safe to conduct non-parametric tests. Hence, in the current study Friedman test was used to compare the performance of fourteen machine learning techniques repeated over five datasets (Poi, Rave, OrDrumBox, Hudoku and JWebUnit). In this regard, the value of critical region was calculated at 5% significance level. Since there are fourteen techniques, degree of freedom becomes thirteen and for this value the value of $X_{\text{(tabulated)}}$ is obtained from Chi-square table. Further, to rank the performance of each of the machine learning technique, their FIR is calculated using equation 2.15 and compiled in Table 7.14 and Table 7.15, for MMRE and MaxMRE respectively. As discussed earlier, lower the mean rank means better the performance. The outcome of the Friedman test as compiled in Table 7.14 with respect to MMRE measure indicates that the performance of THRIFT technique is the best and GFS-GPG-R is the second best technique. With respect to MaxMRE from the 7.15, we observe that GFS-GPG-R technique is the best and THRIFT as the second best technique for the maintainability prediction of open source software on the basis of their mean rank.

Table 7.14: Mean Ranking of Techniques by applying Friedman Test on MMRE Values

S.No.	Machine Learning Technique	Mean Rank
1.	THRIFT	1.40
2.	GFS-GPG-R	1.80
3.	GANN-R	5.00
4.	GFS-GAP-Sym-R	5.40
5.	NU-SVR-R	6.40
6.	GFS-SAP-Sym-R	7.80
7.	Decr-RBFN	9.00
8.	EPSILON-SVR-R	9.00
9.	NLBP	9.20
10.	M5-Rules	9.80
11.	ProQuardratic-MS-R	10.80
12.	CART	11.40
13.	Linear-LMS-R	13.80
14.	NNEP-R	4.20

Table 7.15: Mean Ranking of Techniques by Applying Friedman Test on MaxMRE Values

S.No.	Machine Learning Technique	Mean Rank
1.	GFS-GPG-R	2.80
2.	THRIFT	3.00
3.	GFS-GAP-Sym-R	5.00
4.	GANN-R	5.10
5.	GFS-SAP-Sym-R	5.30
6.	NNEP-R	5.40
7.	CART	7.20
8.	ProQuardratic-MS-R	8.80
9.	Linear-LMS-R	9.60
10.	NLBP	9.60
11.	NU-SVR-R	9.90
12.	Decr-RBFN	10.20
13.	M5-Rules	11.20
14.	EPSILON-SVR-R	12.00

We reject the null hypothesis and accept the alternate hypothesis and conclude that the evolutionary fuzzy techniques are significantly better than their counterpart. Further, in order to ascertain whether the performance differences which exist between FIR values of various machine learning techniques is statistically significant or not, we proceed towards post hoc analysis. The value of CD is calculated as 8.7 after putting the values of n as 14 (Number of machine learning techniques) and value of k as 5 (number of datasets) into equation (2.14). Next, we make a pair for each machine learning technique with every other to calculate their rank differences (FIR) and compiled in Table 7.16 and Table 7.17 for MMRE and MaxMRE, respectively. In total, 91 such pairs were formed as we have used 14 machine learning techniques in our study.

Table 7.16: Computation of Pair Wise Rank Difference among all Machine Learning Techniques in terms of Mean Magnitude of Relative Error

Technique	THRIFT	GFS-GPG-R	NNEP-R	GANN-R	GFS-GAP-Sym-R	NU-SVR-R	GFS-SAP-Sym-R	Decr-RBFN	EPSILON-SVR-R	NLBP	M5-Rules	Pro Quardratic MS-R	CART	Linear-LMS-R
THRIFT	0.0	0.4	2.8	3.6	4.0	5.0	6.4	7.6	7.6	7.8	**8.4**	**9.4**	**10.0**	**12.4**
GFS-GPG-R	0.0	0.0	2.4	3.2	3.6	4.6	6.0	7.2	7.2	7.4	8.0	**9.0**	**9.6**	**12.0**
NNEP-R	0.0	0.0	0.0	0.8	1.2	2.2	3.6	4.8	4.8	5.0	5.6	6.6	7.2	**9.6**
GANN-R	0.0	0.0	0.0	0.0	0.4	1.4	2.8	4.0	4.0	4.2	4.8	5.8	6.4	**8.8**
GFS-GAP-Sym-R	0.0	0.0	0.0	0.0	0.0	1.0	2.4	3.6	3.6	3.8	4.4	5.4	6.0	**8.4**
NU-SVR-R	0.0	0.0	0.0	0.0	0.0	0.0	1.4	2.6	2.6	2.8	3.4	4.4	5.0	7.4
GFS-SAP-Sym-R	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.2	1.2	1.4	2.0	3.0	3.6	6.0
Decr-RBFN	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.8	1.8	2.4	4.8
EPSILON SVR-R	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.8	1.8	2.4	4.8
NLBP	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	1.6	2.2	4.6
M5-Rules	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.6	4.0
Pro Quardratic MS-R	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	3.0
CART	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.4
Linear-LMS-R	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

In the Table 7.16, we have highlighted those entries which have values greater than CD. It is quite evident that the out of 91 pairs of machine learning techniques, 10 pairs (bold entries) were found to have significant differences among their performances. One pair between GFS-GAP-Sym-R and Linear-LMS-R has attained the difference (8.4) almost touching the CD (8.6). Hence, 10 pairs out of 91 means, the performance of 10.8% of pairs was found to be significantly different using statistical test and not coincidental. Results shown in Table 7.16 also depicts that THRIFT performed better than M5Rule, ProQuardratic-MS-R , CART and Linear-LMS-R. Similarly GFS-GPG-R performed better than ProQuardratic-MS-R , CART and Linear-LMS-R techniques. Hence, on the basis of post hoc analysis of MMRE, we conclude that evolutionary fuzzy techniques outperformed than other machine learning techniques. The difference between the performances of all other machine learning techniques were not found to be significant.

We performed the same procedure for MaxMRE and the rank difference of each pair was calculated and compiled in Table 7.17. Highlighted entries in the Table 7.17 indicate that the difference of FIR between that pair of machine learning technique is greater than CD. It is observed that out of 91 pairs of machine learning techniques, 3 pairs (bold entries) were found to have significant differences among their performances. So, with the help of Nemenyi Test conducted on MaxMRE measure, 3 pairs were found to be significantly different out of 91 pairs which is almost 3.2% of the total pairs. It is also quite apparent that GFS-GPG-R is significantly better than M5Rule and EPSILON-SVR-R techniques.

Table 7.17: Computation of pair wise rank difference among all Machine Learning techniques in terms of MaxMRE

Techniques	GFS-GPG-R	THRIFT	GFS-GAP-Sym-R	GANN-R	GFS-SAP-Sym-R	NNEP-R	CART	Pro Quardratic-MS-R	Linear-LMS-R	NLBP	NU-SVR-R	Decr-RBFN	M5-Rules	EPSILON-SVR-R
GFS-GPG-R	.00	.20	2.20	2.30	2.50	2.60	4.40	6.00	6.80	6.80	7.10	7.40	**8.40**	**9.20**
THRIFT	.00	.00	2.00	2.10	2.30	2.40	4.20	5.80	6.60	6.60	6.90	7.20	8.20	**9.00**
GFS-GAP-Sym-R	.00	.00	.00	.10	.30	.40	2.20	3.80	4.60	4.60	4.90	5.20	6.20	7.00
GANN-R	.00	.00	.00	.00	.20	.30	2.10	3.70	4.50	4.50	4.80	5.10	6.10	6.90
GFS-SAP-Sym-R	5.30	.00	.00	.00	.00	.10	1.90	3.50	4.30	4.30	4.60	4.90	5.90	6.70
NNEP-R	5.40	.00	.00	.00	.00	.00	1.80	3.40	4.20	4.20	4.50	4.80	5.80	6.60
CART	7.20	.00	.00	.00	.00	.00	.00	1.60	2.40	2.40	2.70	3.00	4.00	4.80
Pro Quardratic-MS-R	8.80	.00	.00	.00	.00	.00	.00	.00	.80	.80	1.10	1.40	2.40	3.20
Linear-LMS-R	9.60	.00	.00	.00	.00	.00	.00	.00	.00	.00	.30	.60	1.60	2.40
NLBP	9.60	.00	.00	.00	.00	.00	.00	.00	.00	.00	.30	.60	1.60	2.40
NU- SVR-R	9.90	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.30	1.30	2.10
Decr-RBFN	10.20	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	1.00	1.80
M5-Rules	11.20	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.80
EPSILON-SVR-R	12.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00

Hence, we conclude that the difference in the performance of GFS-GPG-R and THRIFT both under the category of evolutionary fuzzy techniques were significantly better than machine learning and statistical techniques. Difference among the performance of all other machine learning techniques is not found to be significant.

7.5 Discussion

In open source software, practitioners across the globe are allowed to change, expand and redistribute the newly created version without any requirement of the license [196]. Changes in open source software are made continuously in order to remove defects, improve functionality, and increase usefulness [192]. Estimating the maintainability of open source software becomes more challenging due to the lack of technical support and the absence of adequate documentation. In this chapter, we evaluate the performance of evolutionary techniques for software maintainability predictions. We compared the prediction performance of evolutionary fuzzy, evolutionary neural and evolutionary neural symbolic regression methods with traditional statistical and machine learning models. dataset was collected from five open source software systems using OO metrics proposed by Chidamber and Kemerer [43]. Based on publicly available open source dataset, we empirically analyzed the performance of methods using the prevalent accuracy measures. Our main results are as follows:

1. The results indicate evolutionary techniques generally perform better than traditional techniques and they could achieve accuracy within the range of 75% to 78%.

2. We conducted Nemenyi test to compare the performance of the evolutionary techniques performs significantly better than the traditional machine learning and statistical techniques.

3. With the help of post hoc analysis, the performance of evolutionary fuzzy techniques were compared and we found that 10.8% of pairs were significantly better and not coincidental.

4. An important contribution in this chapter is that we have compared results using OO metric suite on five open source software systems. Hence, we can generalize our results and they can be repeated in future empirical studies.

5. This results achieved in this chapter confirms that construction of evolutionary technique for software maintainability prediction is feasible, adaptable and useful in predicting software maintainability.

6. We observed that overall good prediction accuracy is achieved by almost all machine learning techniques, however, the prediction models using evolutionary fuzzy techniques perform better than the other machine learning techniques in the context of open source software systems.

Chapter 8

Empirical Study to Assess Refactoring Effects on Software Maintainability

8.1 Introduction

Once the software product is developed and delivered to the customer, maintenance process keeps modifying it to improve performance, correct fault or adapt the product to a modified environment. Refactoring is the part of maintenance phase in which the design of software is improved and complexity is reduced without affecting its external behavior. Many refactoring methods have been suggested in the literature and each has a particular purpose and corresponding effect. Unfortunately, it is unclear how the specific refactoring method affects software maintainability. The objective of this empirical study was to observe the quantifiable effects of few widely used refactoring methods on maintainability. In this regard, the design metrics of the software were calculated and analyzed before as well as after the application of refactoring on five proprietary systems. Comprehensive reports were prepared to observe the effect of selected refactoring methods on design metrics which were further mapped to maintainability. Findings of this study would be useful to project managers in identifying the opportunities of refactoring in large code so that particular refactoring method could be applied to optimize software maintainability.

During the maintenance phase, we keep modifying the code and the code turns so bad that it really needs major refurbishment. Whether refactoring should be applied at this stage or not, what are the advantages and disadvantages associated with refactoring process at this stage require substantial empirical investigation. In this work we empirically evaluated the consequences of refactoring process on maintainability. The results are validated using five proprietary systems developed in Microsoft Visual Studio (.NET) software using C# language. The values of the OO metrics of source codes for the selected software were collected before and after refactoring methods were applied. Change in the values of the OO metrics helped us in determining its overall effects on maintainability. To the best of our knowledge there have been so far no reports to undertake organized study which classify various commonly used refactoring methods along with their corresponding quantitative effects on the OO metrics. This study guides in taking appropriate direction regarding 'when to refactor', 'what to refactor' and 'how much to refactor'. The main aims of this study are given as follows:

- To find if the impact of refactoring exists significantly on OO metrics?
- To investigate how the refactoring affects software maintainability?
- To ascertain whether the effect of refactoring is same on small, medium and large systems?

The chapter is organized as follows: Section 8.2 presents the basics of refactoring process and ways to ensure safe refactoring into the code. Section 8.3 presents the research methodology followed and state the hypotheses in this chapter. The results of the study are given in section 8.4 and the validation of the hypothesis is presented in section 8.5 and finally, discussion of the work done in this chapter is presented in section 8.6.

The results of this chapter have been reported in [144].

8.2 Refactoring Process

Refactoring is a process in which for any given software, its internal structure is improved, complexity is reduced and external behavior remains the same. Due to refactoring treatment provided into the source code, it becomes simpler and easier to maintain as the changes are very systematic in nature. There are many methods of refactoring like dead code elimination, clone code removal, extract method, lazy classes, pull up method, push down method, hide methods, renaming etc and each method has its effects on software quality attributes. It is very important to critically examine and quantify the effects of refactoring on software maintainability. Generally, during the SDLC good design is developed first followed by the coding process; whereas in refactoring, improvement in the design is performed after the coding of the software [69]. Various advantages of refactoring includes enhancement of new features and improvements in understandability, readability and maintainability by enforcing fine-grained encapsulation in to the code.

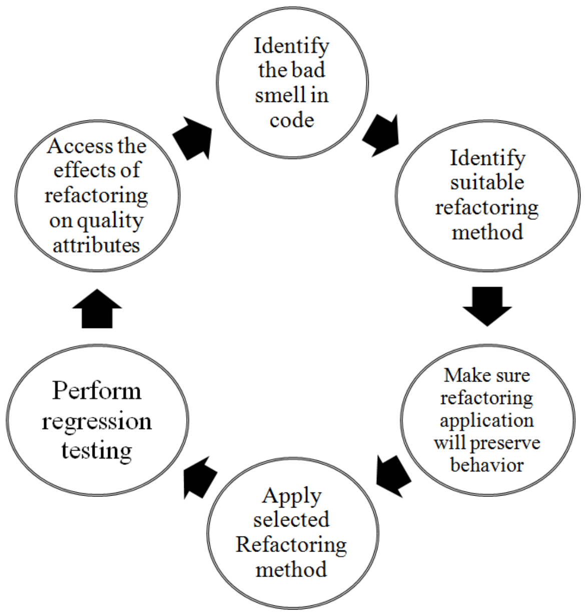
Identify the bad smell in code
Identify suitable refactoring method
Make sure refactoring application will preserve behavior
Apply selected Refactoring method
Perform regression testing
Access the effects of refactoring on quality attributes

Figure 8.1: Steps Undertaken during the Refactoring Process

During the refactoring process, various elements of source code such as fields, methods, classes and packages are redesigned to improve the overall code quality of the software. Although automated application of refactoring is provided by many IDEs nowadays, however, refactoring is not as easy as it seems. In general the steps undertaken during the refactoring process are shown in figure 8.1.

While refactoring the software, first of all the bad smells present into the code needs to be identified. Bad smells are actually the structural characteristics of software that indicate problem in code or design due to which software become complex thus hard to understand and maintain. Bad smells increases the overall cost of the software, hence, whenever code smells are recognized, then actions need to be taken to improve it by applying appropriate refactoring treatment. So we can say that that bad smells are the indicator for the need of refactoring. Moreover, if refactoring is not applied properly with utmost care, than it may lead to the accumulation of technical debt, hence results in failure of the software.

Table 8.1: Bad Smells and Respective Refactoring method

S.No.	**Bad Smell**	**Definition**	**Refactoring**
1.	God Class	It is the class which tends to centralize the system intelligence.	Extract Class
2.	Long Method	It is the method which is long that makes it difficult to modify and understand.	Extract Method
3.	Type Checking	In this bad smell, the function is split into multiple functions so to handle a single type and thus increases redundancy.	Replace Type code with State/Strategy
4.	Feature Envy	It means that a method is interested more in other class than the one where it is currently located.	Move Method

Initially, Fowler [69] introduced 22 different kind of bad smells which are further enhanced by many researchers. Few very common bad smells are God class (violates the decomposition design principles), Long Method (method is very large), Data Class (indicates bad data abstraction), Refused Bequest Class (class inherits another class but does not use it), Duplicate Code (code appears in more than one place) etc.

For each kind of the bad smell, different kind of refactoring method is suggested. For example, if there is bad smell such as “God Class”, then the class is split into two or many classes and decomposition design principle is induced into the code. Code becomes modular which further increases understandability and reusability. In Table 8.1, we have given four such examples of bad smell and respective refactoring method.

After identifying the corresponding refactoring method for the particular bad smell, it is applied into the code. In order to ensure that that there is no side effect on external behavior of the software due to the application of the refactoring method, regression testing is conducted. In the current study, we further analyze the effect on OO metrics to investigate the effects of refactoring on software quality attributes.

8.3 Research Methodology

This section discusses the research methodology adopted to investigate the effect on software maintainability by the application of specific refactoring methods. As shown in figure 8.2, we begin our work with the selection of the OO metric suite to measure maintainability. Important refactoring methods were also selected that redistribute responsibilities among classes. The research hypothesis was stated and experiments were conducted. Values of each of the metric was recorded before and after the application of refactoring treatment into the code. Individual effects of refactoring method on the OO metric suite were analyzed and finally, the cumulative effect of this process is calculated on maintainability.

The methodology adopted for the current study is presented in figure 8.2. Firstly, we identify 'Bad Smell' present in the code using Jdeodrant tool. Next, we identify and select corresponding refactoring method to remove this smell. The OO metrics of the code were collected before applying the particular refactoring method. In the next process, refactoring was applied and the values of the OO metrics were again collected. Change in the values of each OO metric for each class corresponding to the application of each refactoring method was tabulated. Finally a comprehensive report was prepared for each refactoring method to observe its subsequent effects on each of the system. Cumulative effects on the OO metrics for each refactoring method was evaluated and mapped to maintainability and finally

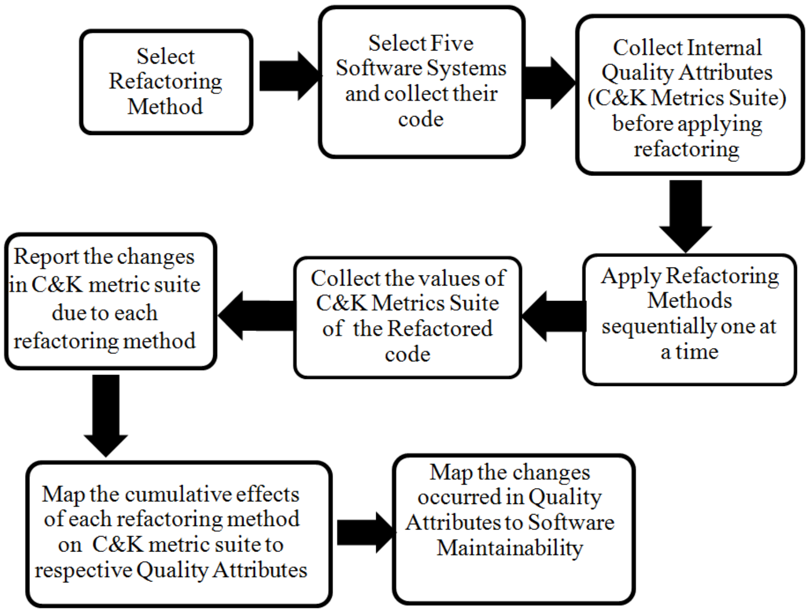
Select Refactoring Method
Select Five Software Systems and collect their code
Collect Internal Quality Attributes (C&K Metrics Suite) before applying refactoring
Apply Refactoring Methods sequentially one at a time
Collect the values of C&K Metrics Suite of the Refactored code
Report the changes in C&K metric suite due to each refactoring method
Map the cumulative effects of each refactoring method on C&K metric suite to respective Quality Attributes
Map the changes occurred in Quality Attributes to Software Maintainability

Figure 8.2: Research Methodology

null/alternative hypothesis was accepted/ rejected.

8.3.1 Relationship of Object-Oriented Metrics and Maintainability

As discussed in the previous section, in this study the metric suite proposed by Chidamber and Kemerer [43] is used to capture the design characteristics. We have chosen this metric suite as it is validated by many researchers [6, 53, 62, 118, 219]. In their respective empirical studies, they were able to find a significant positive correlation between OO metric suite and maintainability. The values for these OO metrics are obtained for each class of the software before as well as after the application of refactoring process and the changes observed in the values of the OO metrics were further mapped to maintainability. The formal definition of Chidamber and Kemerer metrics suite and their effects on maintainability are given below.

8.3.1.1 Relationship of WMC with Maintainability

It is calculated as the sum of McCabe's Cyclomatic Complexities of all local methods defined in a class. Many empirical studies such as Koten and Gray [118], Aggarwal et al. [6]

and Li and Henry [127], suggested keeping its values as much less as possible for achieving more maintainability. More methods if packed in one class would not only reduce its reuse, but the class also becomes more complex which in turn increases maintainability.

8.3.1.2 Relationship of DIT with Maintainability

It measures the maximum inheritance path from any class to the root class. As we increase depth by implementing inheritance in our code, it increases reusability and maintainability but beyond a certain depth, the code becomes very complex and hard to maintain as suggested by Daly et al. [55].

8.3.1.3 Relationship of NOC with Maintainability

It is used to measures the breadth present in the class hierarchy. It is counted as the number of immediate child classes derived from the base class. The depth which is measured through DIT is always preferred over breadth since it promotes reusability as empirically proved by Li and Henry [127].

8.3.1.4 Relationship of CBO with Maintainability

It measures the coupling present between the given class and other classes. Two classes are said to be 'coupled' if methods declared in one class uses methods or instance variables which are defined in other class. The value of CBO should be kept as lower as possible in order to have a good maintainable system. Sahraoui et al. [191] suggested a maximum CBO value to be less than 14 as increased value means there is very high coupling between the classes which would have negative impact on maintainability. Also from the point of view of 'Reusability' excessive coupling should be avoided.

8.3.1.5 Relationship of RFC with Maintainability

It is counted as the number of methods which gets executed whenever a message is communicated to any object of that class. Its value should be kept as low as possible to keep the system maintainable as suggested by Li et al.[127]. In their study, it is also empirically proved that large RFC means more and more methods get executed in response to any message received by an object of that class. Due to this phenomenon, tracing an error would be extremely difficult and so the maintenance process.

8.3.1.6 Relationship of LCOM with Maintainability

It actually measures the cohesiveness present in a given class. It is calculated by counting the number of disjoint sets of local methods present in the class with respect to the member variables used in each function. More cohesive classes are easier to maintain and as the value of LCOM increases, classes become harder to maintain. Although many deficiencies were identified in LCOM metric by Mayer and Hall [155] and new metric LCOM3 is proposed, however current study uses original definition of LCOM given by Chidamber and Kemerer [43].

8.3.2 Selection of Refactoring Methods

Fowler [69] has defined more than 70 methods of refactoring along with their motivation and a step-by-step description of execution. Few are very simple in nature such as renaming, code extraction or pull-up methods where as few are very complex such as exchanging risky and long language idioms with safer alternatives or the code optimization. In the current study, we have chosen five refactoring methods and evaluated their effect on maintainability. Only those refactoring methods were chosen which either redistributes the responsibilities within classes or operate at methods' level so that their effects on the OO metrics could be observed. Our criteria is also influenced by the tool being used for implementing the refactoring methods. Very primitive methods such as renaming etc have been deliberately avoided since they do not have any significant impact on the OO metric suite and we will not be able to ascertain its effects on maintainability. The five refactoring methods chosen in the current study are explained in brief as follows:

8.3.2.1 Consolidate Conditional Expression

Consolidate Conditional Expression (CCE) refactoring method combines many consecutive conditional statements into a single statement which contains the corresponding conditional expression.

8.3.2.2 Encapsulating Field

If a data field is public and accessed directly in the program it violates data hiding principal of the OO programming. Encapsulating Field (EF) refactoring process converts public data members in to private data members and provides two extra member functions to 'get' and 'set' their values.

8.3.2.3 Extract Class

If the existing class becomes too complex, Extract Class (EC) refactoring method creates a new class and move the relevant fields and methods from the source class into the newly created class.

8.3.2.4 Extract Method

In Extract Method (EM) refactoring method first we identify a piece of code that can be grouped together; we extract those groups of statements, put them into a new method and give some sensible name to the newly created method.

8.3.2.5 Hide Method

In Hide Method (HM) refactoring process, the visibility of member function is changed from public mode to private mode before ensuring that it is not used by any other class in the system.

8.3.3 Research Hypotheses

To focus our study on measuring the effects of refactoring methods on maintainability, we set up Null Hypothesis and Alternative Hypothesis as follows:

8.3.3.1 Consolidated Conditional Expression Hypothesis

Null Hypothesis: A Class on which CCE refactoring method is applied does not alter its Maintainability.

Alternate Hypothesis: A Class on which CCE refactoring method is applied, its code quality improves which further enhances its Maintainability.

8.3.3.2 Encapsulating Field Hypothesis

Null Hypothesis: A Class on which EF refactoring method is applied does not alter its Maintainability.

Alternate Hypothesis: A Class on which EF refactoring method is applied, its code quality improves which further enhances its Maintainability.

8.3.3.3 Extract Method Hypothesis

Null Hypothesis: A Class on which EM refactoring method is applied does not alter its Maintainability.

Alternate Hypothesis: A Class on which EM refactoring method is applied, its code quality improves which further enhances its Maintainability.

8.3.3.4 Extract Class Hypothesis

Null Hypothesis: A Class on which EC refactoring method is applied does not alter its Maintainability.

Alternate Hypothesis: A Class on which EC refactoring method is applied, its code quality improves which further enhances its Maintainability.

8.3.3.5 Hide Method Hypothesis

Null Hypothesis: A Class on which HM refactoring method is applied does not alter its Maintainability.

Alternate Hypothesis: A Class on which HM refactoring method is applied, its code quality improves which further enhances its Maintainability.

These hypotheses are further tested at 5% level of significance using Z-test statistical measure to figure out whether the individual effects of each refactoring methods on maintainability is significant or not. The sample mean and population mean is calculated. Z-test is applied on the final compiled data using equation (8.1).

$$Z = \frac{X - \mu}{\frac{\sigma}{\sqrt{n}}} \tag{8.1}$$

Where X is Sample Mean, μ is population mean, σ is population standard deviation and n is the size of the population.

8.3.4 Empirical Data Collection

The current study was undertaken to establish the effect of refactoring process on the simple software system as well as on more complex system software. With an aim to accommodate beginners as well as professionals, two software codes were chosen from the projects undertaken by the students of B.Tech and three software codes were taken from the software industry. These three complex systems were developed and maintained by professionals. First, two software's FLM system and SMS were medium in size whereas IMS, ABP and EASY classes online services were large systems.

8.3.5 Descriptive Statistics

Five proprietary systems are used for the validation of new metric suite namely FLM system, EASY system, SMS system, IM system and ABP system as described in chapter 2. They consists of 233, 292, 129, 96 and 114 classes respectively. Descriptive statistics such as Max, Min, Mean, Median and Std Dev were calculated for FLM system, EASY system, SMS system, IM system and ABP system and presented in Table 8.2, Table 8.3, Table 8.4, Table 8.5 and Table 8.6 respectively. These statistics are closely examined and analyzed for easy and correct comparison between different case studies. Following are the observations made:

Table 8.2: Descriptive Statistics of FLM System

S.No.	Metric	Max	Min	Mean	Median	Std Dev
1.	WMC	16	1	6.276	5	4.97
2.	DIT	7	1	4.379	5	1.32
3.	NOC	7	0	3.1	3	1.67
4.	CBO	50	3	26.14	30	13.85
5.	RFC	67	12	25.16	18	7.89
6.	LCOM	0	0	0	0	0
7.	SCCR	5	2	3.276	3	2.97
8.	NODBC	12	0	2.483	0	3.53
9.	MI	91	40	61.14	56	18.04
10.	CC	29	1	19.31	16	13.76
11.	Change	95	5	41.98	67	45.67

Table 8.3: Descriptive Statistics of EASY System

S.No.	Metric	Max	Min	Mean	Median	Std Dev
1.	WMC	23	1	10.5	9.5	8.57
2.	DIT	5	1	3.6	4	2.50
3.	NOC	8	0	4.23	3	2.91
4.	CBO	54	0	33.73	38.5	21.58
5.	RFC	78	21	37.73	27	4.89
6.	LCOM	0	0	0	0	0
7.	SCCR	7	3	4.57	5	5.57
8.	NODBC	7	0	2.79	0.5	3.43
9.	MI	94	43	64.14	56.5	17.91
10.	CC	22	1	20.6	19	14.26
11.	Change	87	9	52.52	63	43.23

- First two projects namely FLMS and SMS systems are small sized projects and last three projects namely IM system, ABILL system and EASY classes are medium sized software.

- The maximum value of NOC for FLMS, SMS, IMS, ABILL and EASY are 7, 8, 11, 7 and 9 respectively which means reusability is properly implemented.

- The minimum value of LCOM is zero for all the software systems which means the classes are quite cohesive. To calculate LCOM, number of pairs of methods which have common attribute among themselves are subtracted from the number of pairs of

Table 8.4: Descriptive Statistics of SMS System

S.No.	Metric	Max	Min	Mean	Median	Std Dev
1.	WMC	29	2	16.63	17.5	9.17
2.	DIT	6	1	3.25	4	2.12
3.	NOC	11	0	4.85	4	2.67
4.	CBO	59	3	45.38	52.5	18.66
5.	RFC	83	19	37.09	31	5.87
6.	LCOM	0	0	0	0	0
7.	SCCR	6	2	4.625	16.5	9.17
8.	NODBC	6	0	3.89	3	2.50
9.	MI	81	49	55.25	52	10.56
10.	CC	27	1	21.50	19.5	19.61
11.	Change	79	13	67.89	47	32.43

Table 8.5: Descriptive Statistics of IMS System

S.No.	Metric	Max	Min	Mean	Median	Std Dev
1.	WMC	12	0	3.147	3	2.57
2.	DIT	5	4	4.029	4	0.17
3.	NOC	7	0	2.81	3	1.91
4.	CBO	30	2	13.08	13.5	8.09
5.	RFC	43	18	21.09	27	5.07
6.	LCOM	0	0	0	0	0
7.	SCCR	12	0	3.147	3	2.57
8.	NODBC	5	0	2.118	1	3.85
9.	MI	100	48	71.79	67	17.84
10.	CC	13	2	10.79	7	12.78
11.	Change	213	18	79.87	103	67.93

methods having no common attributes among them. Its negative value is also considered as 0.

- Maximum values of DIT are 7, 5, 6, 5 and 6 respectively for FLMS, SMS, IMS, ABILL and EASY systems which means inheritance is properly exploited in most of the systems to their best.

- Mean values of CBO are 26.14, 33.73, 45.38, 13.08 and 14.93 respectively for FLMS, SMS, IMS, ABILL and EASY systems. These values are observed as comparatively less in all the systems which means coupling is comparatively towards the lower side.

Table 8.6: Descriptive Statistics of ABP System

S.No.	Metric	Max	Min	Mean	Median	Std Dev
1.	WMC	11	1	2.483	2	1.84
2.	DIT	6	3	4.017	4	0.13
3.	NOC	9	0	5.25	5	1.09
4.	CBO	29	4	14.93	17	8.56
5.	RFC	49	21	26.83	31	9.89
6.	LCOM	6	0	0.155	0	0
7.	SCCR	11	1	2.483	2	1.84
8.	NODBC	8	0	4.931	1	1.04
9.	MI	100	40	69.5	61	21.03
10.	CC	14	2	10.33	8.5	8.88
11.	Change	189	19	91.23	78	45.63

Classes with less coupling are always preferable as they are easy to understand, maintain and reuse.

Mapping of changes observed in the values of independent variables to maintainability is performed using the outcome of existing studies.

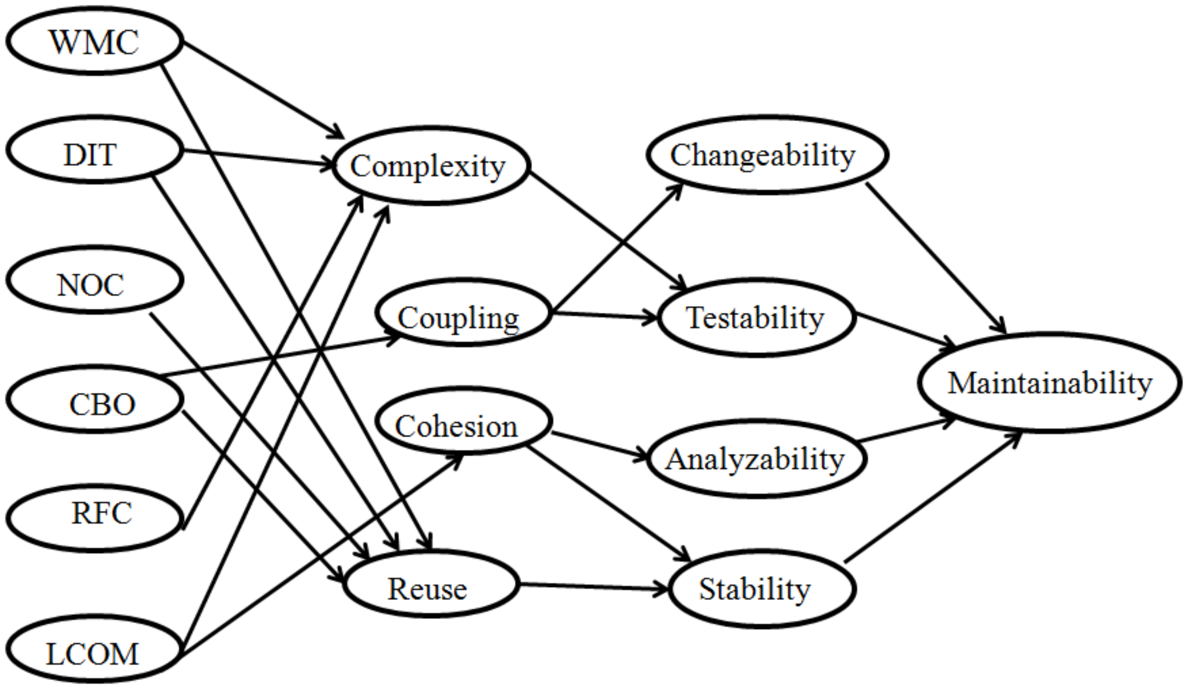

Figure 8.3: Roadmap to Link Changes in OO Metric to Maintainability

Measuring the external quality attributes is very difficult due to the subjective nature involved as suggested by Moser et al. [163]. However, in this chapter, four external quality attributes were carefully selected as they found to be directly correlated with maintainability and measured using independent variables discussed in the previous section. The six important software quality attributes specified in ISO 9126 [95] Quality Model are 'Functionality', 'Reliability', 'Usability', 'Efficiency', 'Maintainability' and 'Portability. Selected OO metrics are measured before refactoring is applied for software code and collected again after the particular refactoring method is applied. This enables us to observe the change in the values of the OO metrics due to the application of specific refactoring method. At a given time only one refactoring method is applied and its effects were observed. The process is repeated each time for all refactoring methods one by one. Since this study is primarily related with 'maintainability', road map presented in figure 8.3 is used to map any change in the values of the OO metric suite to its subsequent effect on maintainability.

8.4 Results and Analysis

In this section we present the results consisting of the changes observed in the OO metrics due to the application of particular refactoring method. First we observe the amount of changes taken place in the values of each the OO metrics due to the application of selected refactoring method one by one. Further, cumulative effects consisting of positive and negative direction due to the application of each refactoring method on the OO metrics were analyzed and judged at 95% level of significance. Validation of hypothesis depending upon the cumulative effects of particular refactoring method on all the OO metrics is discussed in the last subsections.

8.4.1 Effects of Extract Method Refactoring

In this refactoring method, no new class is created. As evident from the Table 8.7, for all the software systems there is no direct effect on the DIT, NOC and CBO metrics. The newly created method increases the mean value of WMC metric for all systems. For small systems, the mean value of WMC is increased by more than 2% whereas for large systems it is increased by less than 1%. Creation of extracted method means there would be more number of methods in each class. For any message received by any object of that class, more number of methods would be executed hence proportionate increase in RFC for all systems is observed. An increase in the LOC metric is also observed due to the addition of newly extracted methods.

Table 8.7: Mean Effects of Extract Method Refactoring on OO Metrics

S.No.	Refactoring	Mean WMC	Mean DIT	Mean NOC	Mean CBO	Mean RFC	Mean LCOM	Mean LOC
1.	Before refactoring	34	3	4	4.86	36.48	2	1484
2.	After refactoring	39	3	4	4.86	42.69	1	1504
3.	Mean change	0.294	0.000	0.000	0.000	0.365	-0.059	1.176
4.	% Change	0.754	0.000	0.000	0.000	0.856	-5.882	0.078

8.4.2 Effects of Encapsulating Field Refactoring

The Encapsulating Field refactoring method converts public data members to private data members and creates two additional methods to get and set their values. As evident from the Table 8.8 that the values of DIT, NOC and CBO metrics remain same when encapsulating field refactoring method was applied for all systems. The mean value of the WMC metric is increased by 4.71% and 2.76% for FLMS system and SMS system respectively. For larger systems also increase in the mean value of WMC is observed. For IM system 2.09%, for ABILL system 1.48% and for EASY system 2.07% change is observed. The value of LOC is also increased for all the systems.

8.4.3 Effects of Consolidate Conditional Expression Refactoring

In this method many conditional statements are replaced by a single conditional statement and the code becomes more compact. The effect of Consolidate Conditional Expression on OO metrics is compiled in Table 8.9. We found that the effect of CCE refactoring method on

Table 8.8: Mean Effects of Encapsulating Field Refactoring on OO Metrics

S.No.	Refactoring	Mean WMC	Mean DIT	Mean NOC	Mean CBO	Mean RFC	Mean LCOM	Mean LOC
1.	Before refactoring	34	3	4	4.86	36.48	2	1484
2.	After refactoring	42	3	4	4.86	39.4	1	1518
3.	Mean change	0.47 1	0.000	0.000	0.000	0.172	-0.059	2.000
4.	% Change	1.12	0.000	0.000	0.000	0.436	-5.882	0.132

the value of the LOC metric is negative for all systems as expected because many conditional statements were replaced by a single statement. As evident from the Table 8.9 that, for smaller systems more than 2% increase in WMC is observed whereas for larger systems this increase is less than 1%. No effect on the mean values of DIT, NOC and CBO metric is observed. Cohesiveness increases by 5%, 10%, 7%, 9% and 3% due to the creation of new methods in few classes for FLMS, SMS, IMS, ABILL and EASY systems respectively.

Table 8.9: Mean Effects of Consolidate Conditional Expression Refactoring on OO Metrics

S.No.	Refactoring	Mean WMC	Mean DIT	Mean NOC	Mean CBO	Mean RFC	Mean LCOM	Mean LOC
1.	Before refactoring	41	7	7	5.69	28.4	6	5941
2.	After refactoring	49	7	7	5.69	31.8	1	5771
3.	Mean change	0.148	0.000	0.000	0.000	0.063	-0.093	-3.148
4.	% Change	0.302	0.000	0.000	0.000	0.198	-9.259	-0.055

8.4.4 Effects of Extract Class Refactoring

We observed that the mean values of DIT, NOC and CBO metric increases significantly whereas the increase in the mean value of NOC is very marginal for simple as well as complex systems. As evident from the Table 8.10 that, in the context of simple systems for FLM systems increase in the mean value of NOC is 2 and for SMS system it is 3. In the context of complex systems, for IM system, an increase in the mean value of NOC is 3, for ABILL system increase in the mean value of NOC is 5 and for EASY system increase in the mean value of NOC is 5.

Table 8.10: Mean Effects of Extract Class Refactoring on OO Metrics

S.No.	Refactoring	Mean WMC	Mean DIT	Mean NOC	Mean CBO	Mean RFC	Mean LCOM	Mean LOC
1.	Before refactoring	27	5	6	2.48	24.62	4	1729
2.	After refactoring	22	7	9	3.19	21.44	7	1889
3.	Mean change	-0.172	0.069	0.103	0.024	-0.110	0.103	5.517
4.	% Change	-0.784	0.985	1.149	0.767	-0.511	1.478	0.292

8.4.5 Effects of Hide Method Refactoring

These refactoring methods neither create new classes nor redistribute responsibilities among classes/ methods. As evident from the Table 8.11, it was observed that the OO metrics were not at all affected by the application of this refactoring method, however, it is always advisable to make use of 'Hide Method' refactoring process as it constantly enforces the principle of OO paradigm into the code.

Table 8.11: Mean Effects of Hide Method Refactoring on OO Metrics

S.No.	Refactoring	Mean WMC	Mean DIT	Mean NOC	Mean CBO	Mean RFC	Mean LCOM	Mean LOC
1.	Before refactoring	34	3	4	4.86	36.48	2	1484
2.	After refactoring	39	3	4	4.86	42.69	1	1504
3.	Mean change	0.294	0.000	0.000	0.000	0.365	-0.059	1.176
4.	% Change	0.754	0.000	0.000	0.000	0.856	-5.882	0.078

8.4.6 Cumulative Effect of Refactoring Method on Object Oriented Metrics

In this section we first find cumulative effects of each refactoring method on the OO metrics. To achieve this, the percentage change in the mean value of each metric due to the application of all refactoring method is compiled for each software system and presented in Table 8.12.

Table 8.12: Z-Test Results for 5% Level of Significance Against the Change in the Values of OO Metrics

S.No.	Statistical Measures	WMC	DIT	NOC	CBO	RFC	LCOM	LOC
1.	Standard Deviation	0.518	0.405	0.530	0.428	0.483	3.942	0.152
2.	Sample Mean	0.268	0.199	0.263	0.153	0.114	-2.664	0.096
3.	Population Mean	-0.223895						
4.	Z-test Values	0.192	0.132	0.167	0.366	0.505	1.638	1.867

These results are further checked at 5% level of significance for all the five systems under study. A cumulative effect of each refactoring method is calculated by summing the values of effects for each system using Table 8.7 to Table 8.11.

They are further checked for the mean change in the value of each OO metrics and divided into three categories i.e. less than 5%, 5% to 10% and more than 10%. Finally we also analyze if the effects of refactoring method are positive or negative and presented in Table 8.13. The up direction of the arrow represents positive effect of refactoring on maintainability whereas the down arrow represents the negative effects of refactoring on maintainability.

↑ if the cumulative effect is positive and less than 5%

↑↑↑ if the cumulative effect is positive and between 5% to 10%

↑↑↑↑↑if the cumulative effect is positive and more than 10%

Table 8.13: Cumulative Effects of all Refactoring Methods on OO Metrics

S.No.	Refactoring	Mean WMC	Mean DIT	Mean NOC	Mean CBO	Mean RFC	Mean LCOM	Mean LOC
1.	Encapsulate Field	↑↑↑↑↑	-	-	-	↑↑↑	↓↓↓	↑
2.	Extract Method	↑↑↑↑↑	-	-	-	↑↑↑	↓↓↓↓	↑
3.	Hide Method	-	-	-	-	-	-	-
4.	Consolidate Conditional Expression	↑↑↑	-	-	-	↑	↓	↓↓↓
5.	Extract Class	↓↓↓↓	↑↑↑↑↑	↑↑↑	↑↑↑	↓↓↓	↑	↑↑↑↑

↓ if the cumulative effect is negative and less than 5%

↓↓↓ if the cumulative effect is negative and between 5% to 10%

↓↓↓↓↓ if the cumulative effect is negative and more than 10%

− means there is no change

A line graph is also plotted to observe and compare the impact of each refactoring method on each of the OO metrics. Figure 8.4 presents the cumulative effects of each refactoring method on the OO metrics. It can easily be observed that CCE method introduces new methods in the class which uses instances of same data members and class becomes more cohesive. This can be observed in figure 8.4 that the value of LCOM is reduced due to the application of CCE. The impact of CCE on LOC is also visible as many lines are replaced by one or two conditional expressions. The introduction of new class due to the application of 'Extract Class' refactoring method, increases the values of the DIT, NOC and RFC metrics. Refactoring Method 'Encapsulating Field' increases the values of WMC as well as RFC. Due to the application of two refactoring methods namely 'Extraction of class' and 'Extraction of method' the value of LOC increases. 'Hide method' does not have any direct impact on the values of the OO metrics.

We have also compared our results with other related studies. Further, the effects of selected refactoring method on the OO metrics and subsequently on maintainability were also analyzed one by one along with their cumulative effects on the maintainability which

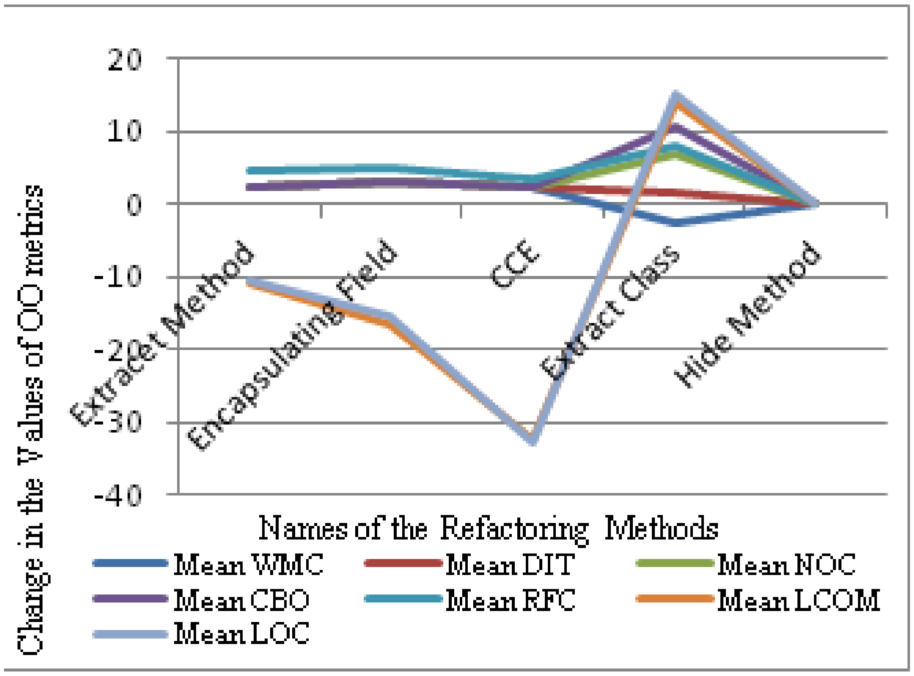

Figure 8.4: Change in Metrics due to Refactoring

are briefly discussed in subsequent section.

8.5 Validation of Hypotheses

In this section we analyze the effects in depth and validate the hypothesis given in section 4.3. Before accepting or rejecting any hypothesis, the effects of applied refactoring methods are evaluated on internal quality attributes as well as on external quality attributes.

8.5.1 Consolidated Conditional Expression Hypothesis

Many lines of source code are merged into a single statement which reduces the value of LOC. Many studies [83, 127, 156, 168, 203] found that the LOC metric has a direct effect on maintainability. WMC and RFC increases due to an additional method being added in the same class and it has a negative impact on maintainability. This newly added method accesses existing instances of data member which results in a reduction of LCOM value and the class becomes more cohesive. There is no change on DIT, NOC and CBO as any new class are not introduced. The cumulative effect on maintainability would be positive as LOC and LCOM found to be a more significant predictor of maintainability in comparison with WMC and RFC. Therefore, we reject the Null Hypothesis and accept Alternate Hypothesis with the conclusion that CCE would certainly have a positive effect on maintainability.

8.5.2 Encapsulating Field Hypothesis

In this refactoring method, public data members are converted in to private and two additional methods (to get and set the value) are created. The values of WMC, RFC and LOC metrics are increased which is attributed to two newly created methods. There was no change in the values of CBO, NOC and DIT as any new class was not introduced. LCOM value reduces as these newly created methods to ‘get’ the values of a member variable and ‘set’ the values of member variables uses the same instances of variable and class becomes more cohesive. The LOC also increases due to newly created methods to get and set the values of private members. Stroggylos and Spinellis [212] and Wie et al. [127] found that if we add any method in to a class, this in turn add responsibilities to the class and it becomes more complex and harder to maintain. However, our study found that it does not necessarily reflect in the corresponding decline in maintainability as the level of abstraction also increases which in turn contributes towards augmented understandability and overall maintainability. The positive effects of ‘understandability due to abstraction’ are much more than the negative effects of WMC, LOC and RFC on maintainability. Hence, we accept alternative hypothesis and reject the null hypothesis and state that EF would have a positive effect on maintainability.

8.5.3 Extract Method Hypothesis

This refactoring method is extremely easy where a large method is decomposed into smaller ones and offers several benefits. The code becomes more readable and the next developer needs less time in investigating the existing code before modifying it during the maintenance phase. It increases the values of the WMC, RFC and LOC metrics because new methods are extracted from existing methods in the same class. There is no effect on the values of the CBO, DIT and NOC metrics as any new class is not created. The class becomes more cohesive as the value of the LCOM metric is reduced due to a newly created method which uses same instances. This study found that even though increased values of the WMC, RFC and LOC metrics gives negative impact on maintainability, yet overall maintainability improves. This can be understood by the beautiful example of a house where although walls made in the house consumes space but it clearly separates each room and the overall house becomes more organized. Hence, we accept the alternative hypothesis and reject the null hypothesis and state that EM would have a positive effect on maintainability.

8.5.4 Extract Class Hypothesis

In this refactoring method new class is created which moves the relevant fields and methods from source classes into a new class. It increases the values of the DIT, CBO and NOC metrics due to the addition of a new class. However reduction in the values of the RFC, WMC and LOC metrics is observed due to additional new classes in which common code of few classes was moved upward with an endeavor to achieve reusability. Further it increases the coupling between the classes as the extracted class and source class shares same data which is visible through the increased value of CBO metric. Since common attributes and methods of many classes are extracted in to a new class in this refactoring process with the sole aim to achieve 'Reusability', overall the value of LOC metric is reduced. Many studies [43, 55, 127] found a reduction in maintainability due to increased values of the DIT, NOC and CBO metrics. In the current study we also found that this refactoring method decreases 'Maintainability' at the cost of gaining 'Reusability'. Hence, we accept the Null Hypothesis and reject the Alternate Hypothesis and state that EC would certainly have a negative effect on maintainability. Here, the software practitioner has to make a cautious choice between 'Maintainability' and 'Reusability'.

8.5.5 Hide Method Hypothesis

This refactoring method only changes the visibility of the member function from public mode to private mode after ensuring that it is not used by any other class. There is no change recorded on the value of internal the OO metric when this refactoring method was applied. It is so because this process does not redistribute responsibilities assign to any class and only the visibility of one method was converted from public to private. Results are in full agreement with Elish and Elish [62]. This refactoring process neither alters the structural design nor do amends responsibilities assign to any class hence we accept the Null Hypothesis and reject the Alternate Hypothesis. Even though this refactoring method found to be not related to the OO metrics and further to maintainability in any circumstances, yet it is an important refactoring process which ensures that codes follow data hiding principle which is an important part of ‘Encapsulation’.

At the end we would like to mention very interesting observation that decision about refactoring should not be made only on the basis of change in the values of the OO metrics, instead while calculating the final impact on maintainability, all the internal quality metrics along with the external quality attributes should also be given enough weightage and based upon cumulative effects of internal and external quality factors, overall final impact on maintainability should be calculated. Programmers cannot blindly apply refactoring each and every day, instead enough time should be spent to find pros and cons of refactoring process and the final call should be adopted.

8.6 Discussion

The effects of few selected refactoring techniques on maintainability have been investigated in this chapter. In the process, a customized source code was used to perform a refactoring process using automated tool. Values of the OO metrics proposed by Chidamber and Kemerer [43] were calculated before and after refactoring process and changes in metrics were further mapped to maintainability. The results achieved in this chapter are based on the experiment's output obtained by applying refactoring methods on real life applications. Since these applications have specific development environment to achieve particular behavior, therefore results cannot be generalized. Based on another empirical investigations carried by us [137], it is assumed that the OO metrics used in the current study to capture software design characteristics of the systems have significant relationship with maintainability. Experiments conducted in controlled environment would be more preferable where all the OO metrics will be kept constant except the one under investigation. To measure the cohesion present in the classes, LCOM metric proposed by Chidamber and Kemerer is used in the current study although it has been criticized by researchers [155]. It is also an important point worth mentioning that code used in the current study is developed in C# and we assume it is equally valid for other OO methodologies, however, further research is required to be undertaken to verify these results on other OO languages such as Java. Five real life systems were used to analyze and empirically validate the effect of refactoring process on maintainability and following results were obtained:

- First observation is that although 'Encapsulating Field' and 'Extract Method' increases values of WMC, LOC and RFC which means reduced maintainability however it is not so because of the increased level of abstraction, increased understandability and increased reusability.
- Second observation is that 'Extract Class' increases maintainability as classes becomes clearer, crisper and organized which further increases reusability understandability, modifiable and maintainability.
- The third observation is that although 'Hide Method' does not change any metrics value and do not have direct effects on maintainability but OO principles are injected into the code and code becomes modifiable.
- Finally, we conclude that decision about maintainability of any given code should not be made on the basis of sheer reflection of values of OO metrics but other external

quality attributes such as ‘Level of Abstraction’, ‘Understandability’, ‘Modifiability’, ‘Extensibility’ and ‘Reusability’ should also be taken in to account while judging the maintainability of the code as they have much larger impact.

Chapter 9

Comparative Analysis of Agile Methods and Iterative Enhancement Model in Assessment of Software Maintenance

9.1 Introduction

Achieving the desired quality of software becomes difficult for developers and they often overstep budgetary constraints, therefore, it is an ongoing endeavor to find a most appropriate solution with minimum cost. The agile approach helps in business to address the problem of unpredictability. The software industry is moving swiftly towards agile methodology to ensure quality, reliability and scalability of software products delivered since it provides alternatives to the traditional project management techniques.

The current chapter provides insight into the impact of the agile framework using scrum on the deliverable as compared to the IEM. Scrum is the most commonly used agile method applied to the projects having fast changing requirements. Development of this method is implemented through a series of iterations known as sprints. In this study, the same product is developed using scrum methods as well as IEM Method and various metrics were used to compare both the products such as the stage of SDLC when the defect was identified, the number of defects identified, the number of change requests received etc. All these metrics are carefully selected as they are significant while analyzing the maintainability of any given software.

In this chapter, we investigate the following issues:

1. How quickly the defects can be identified in the product if it developed using agile based development instead of traditional development.

2. What is the difference in features provided when the product is rolled out using agile methods instead of traditional development with respect to time?

3. Does the number of change requests decrease during the maintenance phase, if the product is based on agile methods instead of traditional methods?

4. Are we able to identify the presence of error sooner if the product is developing in the presence of customer using agile methods?

5. How accurately and precisely the errors can be identified when development is based on scrum method.

In the current chapter, an attempt has been made to quantitatively compare the performance of the agile method and IEM method on software maintenance using maintenance metrics. The chapter is organized as follows: Section 9.2 states the characteristics of agile methodologies, their advantages and disadvantages. Section 9.3 describes various methods covered under the umbrella of agile methodologies. Section 9.4 presents the experimental design, problem statement and design description of two methods IEM and scrum selected in the current study. The results of the study are analyzed in section 9.5 where scrum model is evaluated and compared with IEM method. Finally, discussions of the research is presented in section 9.6.

The results of this chapter have been reported in [143].

9.2 Comparison of Agile Development with Traditional Development

There are many differences between traditional software development methods and agile software development methods. As suggested by Boehm [28], the primary objective of traditional development is high assurance whereas the primary objective of agile software development is rapid value. Traditional systems are designed with meticulous planning hence they are very predictable in nature, however, in the case of agile software, they are developed by small teams imparting the continuous improvements in the system, based on rapid feedback hence they are unpredictable in nature.

9.2.1 Major Characteristics of Agile Technology

Agile methodology results in the addition of new features in a flexible manner when applied to any software development domain. Agile is iterative, incremental and evolutionary which consists of short time frames. It gives special emphasis on face-to-face communication in the presence of a customer representative. It does not involve long-term planning; instead creates a working model and adapts to change requirement quickly as per the continuous feedback received from the customer. It is focused on the quality of the products and the process of unit testing is automated to efficiently run regression suites as and when required. Cross Functional Teams (CFT) are formed in the beginning which includes members with functional expertise in different areas. Pair programming practice is strictly followed to ensure that every modification of addition, deletion or update of the LOC is peer reviewed before it is sent for production. Brief characteristics of agile methodology are described as follows:

1. In agile development, a cross functional team is created which consists of team members with varied functional expertise and specialist knowledge. Most agile methods divide the tasks into smaller increments. Duration of one iteration is typically 15-30

days and it involves a team working in the required functions like planning, requirement elicitation, system design, coding, unit testing and user acceptance testing. After the completion of one iteration, working segment of the product that has been developed is demonstrated to the stakeholders. This helps risk mitigation and allows the project to adjust to changes quickly. A single iteration might not be cumbersome enough to add a major chunk of functionality that could ensure a market release, but after each iteration, the aim is to have a correct working product segment. The release of an entire product with the required feature set is only possible after a number of iterations.

2. Agile is methodical and involves face-to-face communication; hence, it can be used for the development in all domains. However the presence of a customer representative is imperative. For example, in the scrum methodology of agile the product owner acts as the customer representative. The product owner becomes an active participant in the scrum team. Any business related query that affects the development is answered by the product owner. He is the single point of contact for all the stakeholders as well. A continuous re-evaluation of priorities, in terms of the features required to be developed, are initiated after each iteration to maximize the returns and adapt to customer changes efficiently. This also helps in keeping up with the company goals.

3. Agile adapts to requirement changes quickly and aids in continuous feedback. In order to keep track of the progress of the work allocated to a team, daily status meetings are conducted. In the scrum framework these status meetings are referred to as stand ups or the daily scrum. In these meetings the team members notify the other team members of daily progress on their assigned tasks, their plan for that day and if there is some delay in functionality and how they are going to deal with it.

4. The quality of the end product is a prime deciding factor in the success of any development approach based on agile methods. To ensure quality, the team follows certain

best practices and adheres to coding standards. The code from the branched working copy is merged back into the trunk periodically to ensure that the working copy of the developers is up to date and hence the code impact is minimal. The process of unit testing is automated to efficiently run regression suites as and when required. The pair programming practice is followed which ensures that every modification of addition of LOC is reviewed by a peers before being sent to production.

9.2.2 Principles of Agile Technology

Agile methodology has emerged as development paradigm with following set of principles which give an upper hand over the traditional SDLC approaches:

1. Involvement of the user in all development and delivery activities is imperative.
2. The decision-making power should lie with the team.
3. The team's analysis on the time-lines should be trusted.
4. The timescale of iteration is fixed even if the requirements get mature.
5. The requirements should be captured at a high level.
6. The process should always be flexible.
7. Development should be done in small and additive releases.
8. The focus should always remain on frequent roll out of releases and delivery of products
9. The feature being worked upon should be completed before any new feature is picked up for implementation.
10. Pareto principle of 80:20 rules is applied in all cases.
11. The Quality Assurance (QA) activities are introduced early in the development cycle.

12. Stakeholders always remain in perfect synchronization about the requirements and expectations.

9.2.3 Advantages of Agile Methodology

The software industry is moving towards the agile methodology to ensure quality, reliability and scalability of software products delivered as it provides alternatives to the traditional project management techniques. Moreover, the agile approach also helps in the business to respond to the problem of unpredictability. Scrum is most commonly used agile method that can be applied to a project which has rapidly changing requirements because development is implemented through a series of iterations known as sprints.

Agile methodologies always have up-front requirements. Instead of sheer development, it focuses more on the developers' and customers' relationship. Agile methods are iterative in nature which helps the organization to maintain their software in more flexible manner. Due to these continuous iterations that too in the presence of customer, product quality and performance enhances remarkably. With agile methods put in place, customers are more satisfied and maintenance work consumes lesser time and cost.

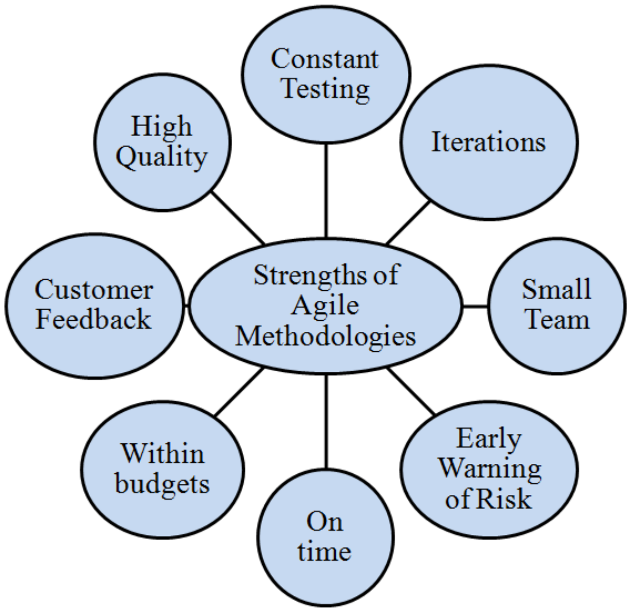

Figure 9.1: Advantages of Agile Methodologies

Agile releases short prototypes after each cycle, so that the users can continuously review it and monitor the development to provide much needed continuous feedback. With the rapidness in the product delivery, the transition of maintenance from waterfall to agile methodology environment is increasingly faster. The advantages of agile methodologies are summarized in figure 9.1.

9.2.4 Disadvantages of Agile Methodology

One of the major disadvantages of agile methods arises due to the presence of customer that it require a big commitment for the duration. Miscommunication is another major factor that leads to the problem during the implementation of the agile methodologies in the SDLC. Testing is conducted throughout the SDLC, therefore, it requires the testers to be at the same place during the lifespan of the project development. This unnecessarily increases the resources of the project which in turn increases the overall cost. It becomes difficult to find the pace for the software development. The overall weaknesses of agile are summarized in figure 9.2.

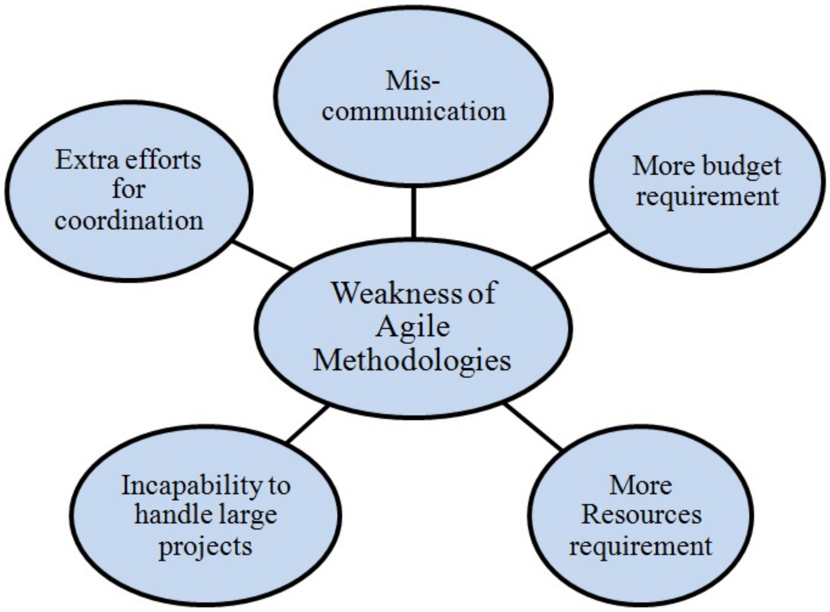
Mis-communication
Extra efforts for coordination
More budget requirement
Weakness of Agile Methodologies
Incapability to handle large projects
More Resources requirement

Figure 9.2: Disadvantages of Agile Methodologies

9.3 Types of Agile Methodologies

There are various methodologies covered under the umbrella of the agile methodology such as crystal programming, scrum, pair programming etc. In figure 9.3 we have summarized all those technologies which are covered under agile methods. However, there are two characteristics which are common to all methods; firstly, they all are iterative, incremental and evolutionary in nature and secondly, customer involvement throughout the SDLC is obligatory. All these technologies are explained in brief as under:

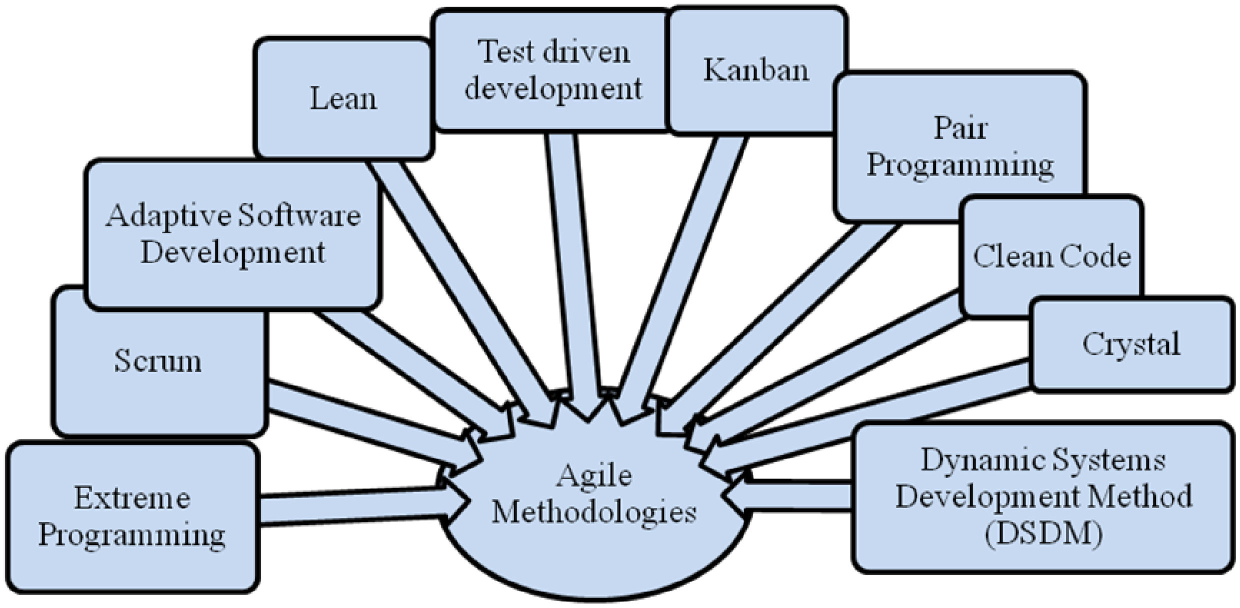

Figure 9.3: Various Technologies Cover under Agile Method

1. Kanban: It acts as a pull system for work-in-progress stories. The visual work-flow is very important and the billboard is prepared to elucidate the status of all the stories. Its aim is to control and manage the business chain instead of simple book keeping by setting an upper cap on the number of work in progress stories.

2. Scrum: It is an iterative model particularly concentrates on performing development tasks in a team based domain. A Cross Functional Teams (CFT) works in collaboration

with other CFTs on a framework consists of a set of rules, a defined pattern of roles, artifacts and planned meetings.

3. eXtreme Programming: Relatively stringent, as the name suggests it goes to the extremes of any applied process. It has a major focus on perpetual reviews, pair programming, code refactoring activities, testing activities, and code reviews which results in a better quality product.

4. Test-driven development: In this approach test cases are written even before the coding itself. Software development involves short cycles wherein quality is always kept as a top most priority. Refactoring into the existing code is frequently carried out till code quality reaches to an acceptable level.

5. Lean: It is specially designed for the application developed under extreme economic conditions. Instead of formal business plans, it starts with plan-as-you-go approach; hence it's a quickest and most effective way to build a new business application.

6. Crystal: It is considered as a lightweight method which can be applied to the development of software systems that are not life-critical. It concentrates on frequent code delivery to the user with the provision of enhancement. Ease of access to the expert user is the most important principle.

7. Adaptive Software Development: Major focus is on handling rapidly changing environments during any stage of the product development. The system is developed by the small team and focus is to embrace, rather than reject, higher rates of change.

8. Dynamic Systems Development Method: It's mainly based on rapid application development with continuous and active customer involvement. It concentrates on cooperation and collaboration between all stakeholders and removes any communication barrier between them. The software system always delivered on time, within budget and meet almost all of the customer requirements.

9. Clean code: Clean codes are based on three principles i.e. developer should use the right tool before writing the code. They should optimize the signal to noise ratio and code should always be self documented. Clean code always presses for small functions and only one function for one job. This principle also highlights the importance of comments and suggests that every code must be written along with clear understandable comments.

10. Pair Programming: Pair programming is one of the famous agile methods originated from XP. As the name suggests, in this technique two developers work together on the single computer during the development phase. One developer is called as ‘driver’ whereas another is called ‘navigator’. Duties of driver involves looking after the techniques, syntaxes, and semantics part whereas the duties of navigator are more towards the level of abstraction. Navigator always think from the tester point of view for example ‘time elapsed since last test run’, ‘type of test cases which have to be passed’, ‘technical tasks to be delivered next’ etc. The basic idea of implementing pair programming is that pairing always results in improved design, less bugs, more functionality and compliance to standards.

9.4 Discussion

There are so many agile methodologies available in the industry as discussed in the previous section and choosing the one was real herculean task. From all these available options, Scrum technique is selected in the current study due to numerous reasons. The main reason of using Scrum is because it is the only method which offers time bound delivery. It is also the most transparent agile method and offers high quality product. Fundamental principal of Scrum is ‘stop starting start finishing’ [97, 112, 178]. It also allows client to change priorities and requirements quickly as per the limitation of the time.

When the product is developed using Scrum, it becomes more stable. As the scrum method require strict adherence to the time-line and schedules, team members can achieve sustainable peace with scrum methods. Although, the framework of the Scrum is very simple, yet it works very well for any complex and innovative projects. It emphasizes team collaboration and provides a small set of rules that create boundaries for teams members and they can focus on problem solving. Scrum gives power to client that he can prioritize the requirements, or even change the requirements during development. At the same time, it gives power to the team to commit to the requirements per their capability. All the work done in Scrum is iterative and incremental, and it time boxes the process. Scrum also emphasizes on feedback, hence the cross functional team get feedback from the client as early as possible and deliver a working product that will actually be used. It also allows the team members to review the product after each sprint so that the improvement can be made within short cycles.

9.5 Experimental Design

This section first explains the difference between scrum and IEM methodology and then presents the problem statement and experiment setup for both the methods.

9.5.1 Scrum versus Iterative Enhancement Method

In this section, we have compared the agile approach with the non-agile approach. For agile based approach, one of the famous agile methods ‘Scrum’ is selected and for non-agile based method IEM is selected. Both the methods are described as follows:

9.5.1.1 Iterative Enhancement Method

The progress is made through successive refinements in IEM Method during which developmental team tries to put first cut into the system knowingly well that many areas are incomplete. The product is improved through refinement in greater detail with each iteration till it reaches a satisfactory level.

As depicted in figure 9.4, massive planning for each sub-part is planned first by creating a working prototype. After the requirement, analysis and design phases for the whole system

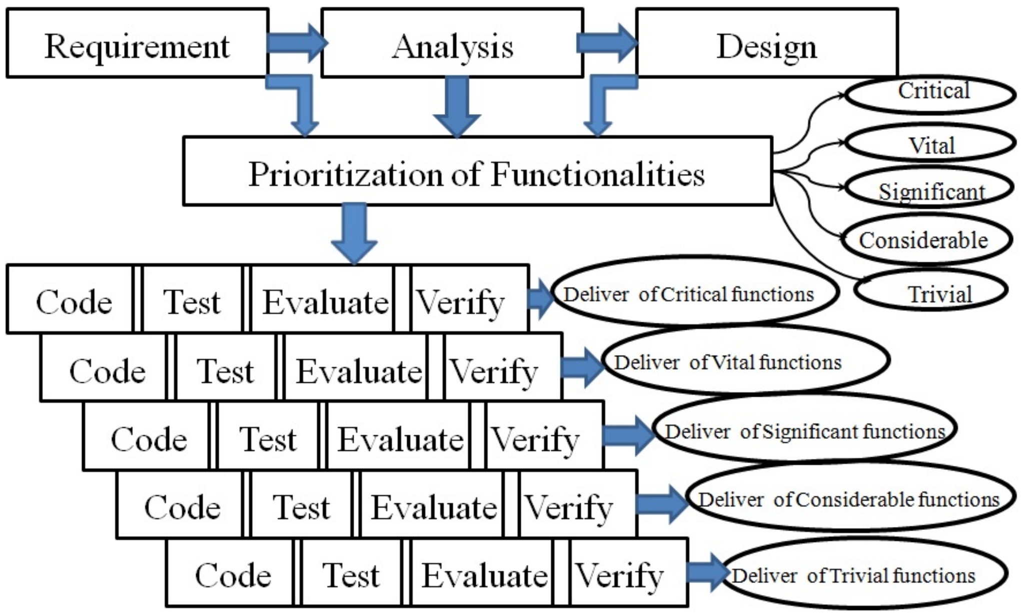
Requirement
Analysis
Design
Prioritization of Functionalities
Critical
Vital
Significant
Considerable
Trivial
Code
Test
Evaluate
Verify
Deliver of Critical functions
Deliver of Vital functions
Deliver of Significant functions
Deliver of Considerable functions
Deliver of Trivial functions

Figure 9.4: Architecture of Iterative Enhance Model

are put in place, functionality expected out of the system are divided into critical, vital, significant, considerable and trivial. Code, test, evaluate and verify phase for each of the subsystems are achieved in next successive phases in sequence. Hence, the team does not focus on tracking the progress of individual feature; instead the focus is on refining the whole product with each phase.

9.5.1.2 Scrum Method

Scrum, one of the most popular agile methods is used in the current study which is based on various concepts such as user stories, daily scrum meetings, product backlog, sprint backlog, sprints, and delivery-ready after each sprint [112, 178, 185]. Developed by Schwaber and Sutherland [199], scrum word is originated from the popular sport ‘rugby’ because the strategy to describe development process is exactly the way it is created in rugby. As shown in figure 9.5, it reduces the planning overhead as the product is very flexible and easily accommodates the changing needs of stakeholders at any developmental stage. Each short cycle known as sprint includes requirement gathering, analysis, design, testing, evaluation and prioritization of features.

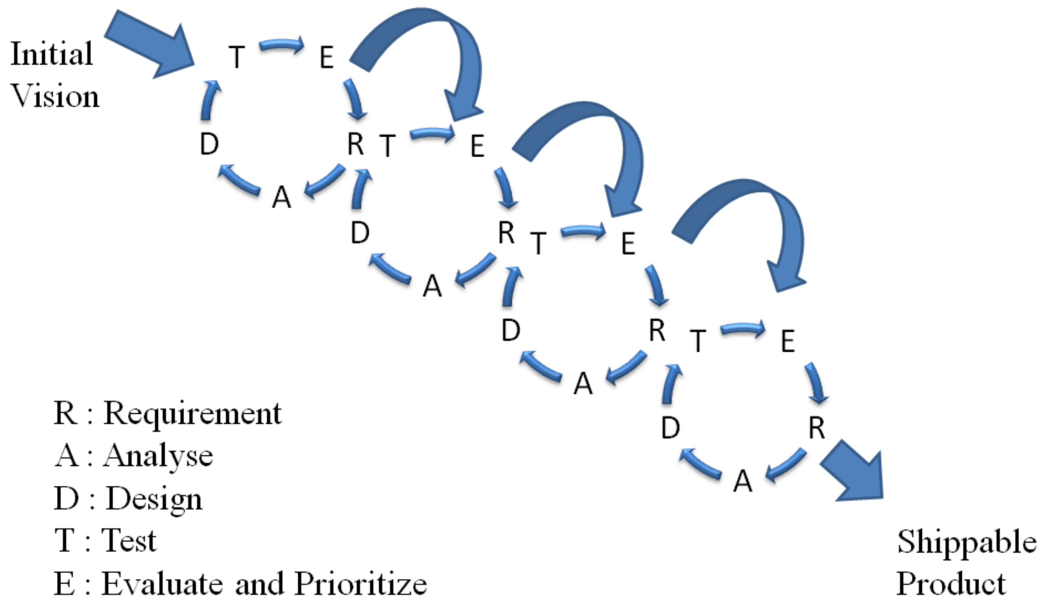

Figure 9.5: Architecture of Scrum Model

9.5.1.3 The Framework of Scrum

There are three major components in scrum framework i.e. roles, ceremonies, and artifacts'. Three major roles in the scrum are the product owner, team and the scrum master. Duties of the product owner involve initial and on-going funding, listing of requirements and release plans. Duties of the teams involve design and implementations of the listed requirements in the presence of the customer. These teams are intentionally made cross functional in order to maximize the performance because they are self-organizing and self managing. Success or failure of any project solely depends on the teams' functionality. One person is appointed as scrum master to ensure that scrum values, practices, and rules are properly enforced. He is also responsible for removing if there are any impediments imposed on the developers.

9.5.2 Problem Statement

In a bid to compare the results provided by the scrum method with IEM method, the first step was to identify the problem statement to be created and developed. The idea was to work on the same problem statement with two different approaches and compare the quality of the product through both methodologies. The aim was to develop an application with an average size and complexity so that the focus could be on implementing the SDLC process efficiently.

The requirements were further scaled down to one project and it was finalized that a hotel management system would be developed using IEM method as well as scrum method. Once the project was selected and finalized, the next step was to decide the requirements of the application which would be similar for both the development methodologies. This led to the requirement gathering phase which was similar for both the development methodologies.

9.5.3 Experiment Setup

The requirement gathering process started with identifying the purpose and scope of the application. It was identified that the purpose of this application was to gradually manage the activities of hotel electronically. A hotel deals with multiple check-in and check-out of guests every day wherein keeping data pertaining to all guests, their daily expenditure incurred on messing, laundry and services have to be adjusted in their final bill at the time of check-outs. It leads to a lot of data management which was required to be developed and the Software Requirement Specification (SRS) for the project was finalized. After the requirement gathering, the design documents were prepared for IEM method and scrum method which were entirely different.

9.5.4 Development using Iterative Enhancement Method

As we finished with requirement gathering, working on IEM method started with the development of the analysis and design documents of the product. The technical design and the architecture of all modules were included during this phase. All functionalities were defined for the product. The next step in this model was to convert the specified design into application code. As we begin with code and test part, first of all, the functionality for add room, display room, and the delete room was coded and tested. Unit testing was done to check for any defects, the bugs were logged, reported and the defect sheet was prepared. In the next phase, we included the Utility Module and now Check-in, Check-out, and Room-occupancy Status functionality were added. This phase also started and continued in the same fashion as the previous two phases. In the next phase, restaurant-dining functionality was added. The last phase included the Report generation for modules on Occupied-Rooms, Free-Rooms, and List-of-All-Rooms. Finally, the documentation of all the phases was completed and it was marked as the end of the first part of the research work.

9.5.5 Development using Scrum Method

In scrum development, a team consisting of six individuals was formed and respective roles were assigned as the main principle involves working in a team. The team includes one product owner, one designer, two developers, one tester and one quality analyzer. The team members acted as the stakeholders and they were empowered to take the decisions for the application. They were provided with the design requirement and given the authority to schedule and prioritize the stories on the basis of the team's bandwidth and the inter-dependencies between stories. The hardware and software requirements, technical specifications of design and architecture were also fixed. At the inception of the project, it was also planned that Agilefant tool will be used. It is a tracking and planning software tool for agile projects which provides end-to-end project management, portfolio management and support product planning. It helps in project scheduling and tracking by managing various activities such as track the progress, check individual assignments, update status and verify burn downs.

9.5.5.1 Activities in Sprint 0

To begin with, requirements were defined in the product backlog tab in Agilefant. It can hold the description of all the new features that have to be incorporated, the bugs which have been identified in the product and the ideas that can be introduced as an improvement to the existing product. The parent level is the product level which holds the name and description of the product while the second level is a project and multiple projects created under a product. A product backlog has a number of stories which have to be completed before a product is considered as completed. The third level is that of iteration, known as a sprint in scrum method. The team decides the stories that can be picked up and completed from the backlog in a sprint. These stories were made as one of the constituents of the sprints. In addition to these stories, a sprint also contains tasks that are required to be performed in order to accomplish these stories. The selection of stories was based on various factors, like their priority for the customer or their dependency on another story.

9.5.5.2 Activities in Sprint 1

The first sprint was used for planning by the team to understand the flow and structure of the application. The team members analyzed the stories that could be started and completed in the first sprint. By this way, they had a fair idea of the product backlog and the requirements of the product owner. They were introduced to the stakeholders of the application. They also attended a story grooming session so that the expectations from various stories were clear to the entire team. At the end of this sprint, the team knew the technology that had to be used to build this application, the stories in the product backlog, the scheduled stories, the stakeholders and Agilefant tool. A spiked story was created to show the occupancy of the users in this Sprint, however, no story points or estimation hours were assigned.

9.5.5.3 Activities in Sprint 2

The second sprint started with the story planning meet during which story points are assigned to the scheduled stories. For example, the add room story remained open due to the defects identified in it and hence the tasks were not deemed complete. The story cannot be considered as closed unless all the constituent tasks of the story get complete individually.

9.5.5.4 Activities in Sprint 3

Similar to the process followed in the earlier sprint, the third sprint started with the planning meet and assigning the story points to the scheduled stories. The team started the development activities on the scheduled stories. The stories for which all acceptance tests correctly passed were successfully closed (marked as done) and the others for which some alternate case flows failed were re-opened and assigned back to the scrum master for further action. For example, the Guest Dining story was successfully closed as no defects were pending to be completed. The story was closed, marked as done and assigned back to the scrum master. Towards the closure of the sprint, the functionality of the done stories was demonstrated to the product owner (PO). The stories are marked as delivered when the PO approves the functionality of the application. With the end of this sprint, the application was developed using both the methodologies as defined above.

9.6 Result Analysis

In the current study, an endeavor was made to compare the same product developed, using IEM method as well as scrum method. In order to identify the cost and other benefits using the agile method, both products were compared using various metrics such as the number of defects identified, time and stage of SDLC when the defects were identified, number of change requests received, time of error detection and the features rolled out by both methods. The results provide evidence that scrum method is beneficial both in terms of fast development and cost reduction without compromising on the quality issue. It also emerged that scrum method starts early on cracking the problem without wasting much time on requirements gathering and initial planning. As the development cycles are short in scrum method, it was also observed that due to the frequent end-user interaction, early feedback was extremely helpful in implementing the corrective maintenance procedures. Surprisingly, the defect rates were reduced and the teams' productivity was also enhanced using scrum method.

The under mentioned four metrics were used to compare both the processes to measure wherever any improvement was provided by scrum over the IEM method:

9.6.1 Number of Defects Identified in the Product

The defect is defined as a condition in software system when it does not meet the requirement or expectations of the customer due to malfunctioning of the program because of the error in coding or logic. In this study, this metric counts the number of defects that were identified in the product using both methods. In Table 9.1, we have compiled the number of defects identified in each of the phases for both methods.

Table 9.1: Number of Defects Identified in IEM and Scrum Method

S.No.	Phase	IEM Method	Sprints	Scrum Method
1.	Phase 1	6	Sprint 0	2
2.	Phase 2	9	Sprint 1	5
3.	Phase 3	1	Sprint 2	4
4.	Phase 4	1	Sprint 3	2

In IEM method, as evident from Table 9.1, the number of the defect was very high in the first two phases and after that decline was observed. However, when the development was done using scrum method, the numbers of defects identified during all the sprints were found to be relatively consistent. When we delve deeper into the statistics and understood the type of defects identified, we realized that the scrum method poses a clear advantage over IEM method because it involves demonstration to the customer and constantly considering his feedback. For example, in the first sprint, the majority of bugs identified dealt with field validations and negative scenarios of junk data entries which were not stated specifically in the SRS. While working in the scrum method, this gap in the requirements was easily noticed just after the first sprint because of the presence of the customer. Hence, the timely feedback from the customer was incorporated in all further stories. However, this advantage could not be derived with the IEM method and the team could not bridge the gap between the stipulated requirements and customer expectations.

9.6.2 Number of Change Requests Received

This metric counts the time (in days) when the change requests were logged by the customer during the product development using the IEM and scrum method which is compiled in Table 9.2.

As evident from the Table 9.2, scrum method detects change requests earlier than the

Table 9.2: Description of the Change Request Received

S.No.	Change Request Description	Day When Change Request Received	
		IEM Method	Scrum Method
1.	Change the date format from dd-mm-yyyy to mm-dd-yyyy	15th days	9th day
2.	Provision for inventory control of the items needed to manage the hotel on a daily basis.	15th days	11th day
3.	Round off for the bill to nearest rupees	15th days	16th day
4.	Provision of foreign currency such as Pound, Dollars, and Yen	30th days	21st day
5.	Trend analysis for the yearlong occupancy for launching the promotional schemes	30th day	26th day
6.	The Rooms that are deleted in the application should be tracked and audited	45th day	32nd day

IEM method as the customer was present constantly during development. For example, the date format used in the application was expected to be changed because the dimensions of the business had changed before the application was finally ready and ported to the customer for use. The customer wanted to sell this product to a third party in the UK which follows a strict mm-dd-yyyy date format. This way the above change in business requirement was addressed at 15th day with IEM method whereas with scrum method, it was detected on 9th day itself. Consider the second case, wherein the customer realized that after deletion of a room from the system, the data was not being audited. This deletion activity was not being tracked whereas it might be required at a later stage for maintenance purpose. Accordingly it was requested that this data should be removed from the active database of the application and stored in a different set of files which can be accessed on as on required basis. This change was actually an addition of a new feature in the application, unlike the previous change requests which was a change in the business requirement. As the product using scrum method was more flexible, accommodating such a request at a later stage was comparatively easier to address than IEM method.

9.6.3 Features Rolled Out with Respect to Time

This metric compares the number of features which were made available to the customer in the product for final use. Table 9.3 and 9.4 highlights the number of features developed and delivered with respect to time at the end of each iterations with IEM and scrum method respectively. The data represented in tables highlight how the major features were developed and delivered.

Table 9.3: Features Rolled Out With Respect to Time in IEM Method

S.No.	**Phases**	**Time Spent**	**Nos. of features**	**Features**
1.	Phase 1	15 days	3	Add Room, Delete Room, Display Room
2.	Phase 2	15 days	3	Check-in, Check-out, Bills
3.	Phase 3	5 days	1	Room Occupied
4.	Phase 4	15 days	3	Restaurant dinning

The scrum method followed a strict demo and release pattern at the end of every sprint; hence the stories completed in each sprint were made available to the customer. However, in the case of the IEM method, each feature was delivered and without a comprehensive feedback cycle the team started working on the development of next phase for delivery. For example, it can be seen that the added room and display room features were delivered to the customer within 10 days using the scrum method. However, it took 15 days to deliver the same features using the IEM method. Hence, it was found that with the use of scrum method, faster delivery of the developed software can be achieved to the customer in comparison of IEM method.

9.6.4 Time of Error Detection

It's well documented that error correction gets costly as their detection is delayed in the SDLC [171]. This metric compares the time of defect detection in IEM and scrum method of development. Table 9.5 compiles the type of defect and time of detection of defects using both the methods separately. As evident from the Table, the scrum method facilitated the

Table 9.4: Features Rolled Out with Respect To Time in Scrum Method

S.No.	Sprint	Time Spent	Nos. of Features	Features Description
1.	Sprint 0	10 days	0	Spike only sprint. This sprint is not billed
2.	Sprint 1	9 days	2	Add Room, Display Room
3.	Sprint 2	12 days	8	Delete Room, Check-in, Check-out, Room Occupied, Free Rooms, List all rooms, Bills
4.	Sprint 3	8 days	8	Restaurant dining room

identification of errors much earlier in the cycle as compared to their time of identification in the IEM method.

This difference is seen because of fixed length iterations and the competency of the team members participating in scrum method. The team members were comfortable with their respective software development technology which further improves the overall productivity of the software.

9.7 Discussion

Agile methodology is becoming the de-facto standard in the software development industry. In the current empirical study, an attempt was made to validate its usefulness for the customer as well as the developer. The benefits that agile methodologies provide needs to be weighed against the code and design quality of the software produced using these methods. In this empirical study, given problem was solved using two approaches, agile based and non-agile based. For the agile based approach, scrum method was selected and for non-agile based approach, IEM was selected to develop the product. Further, their quality was compared against various metrics such as the number of defects identified, time when the defect was identified, stage of the SDLC when the defect was found, number of change requests received, and the features rolled out by both methods. The main findings of the work are summarized below:

1. It clearly emerges that agile methodology encourages better planning due to the cus-

Table 9.5: Time of Defect Detection With IEM and Scrum Method

S.No.	**Defect Title Description**	**Time of Error Detection**	
		IEM Method	**Scrum Method**
2.	During Room Addition time, Invalid inputs are getting accept in Room No. Field.	15	10
3.	Duplicate Room no. Values are getting accept while Room Addition.	15	9
4.	On entering numeral value in ”More Rooms” option, the value is getting accept	15	9
5.	Numeral values are getting accepted in room type while adding Room.	15	8
6.	Crash is observed on entering alphabets while adding Room	30	12
7.	On entering alphabets in ”Menu-Admin-Delete Room” Delete Room functionality is not working.	30	14
8.	Wrong error message ”The rooms of this type are full” is displayed on ”check-in” room type which is not available	30	16
9.	Rooms are getting ”check out” without getting ”check in” and junk values are displayed in phone no. And today’s date.	45	22
10.	Occupant name is displayed as ”@” in ”List of Rooms” displayed.	45	21
11.	Check in date is getting accepted as date higher than the current date.	15	7
12.	On entering invalid Room no. For checkout, no error message is displayed.	15	8
13.	Spelling error of ”Restaurant” observed at the home page.	30	22
14.	At ”Check-in” option, on entering numerals value in Name, wrong value is displayed in List of Rooms available[name + address value appended]	30	18
15.	At check-out option, in phone no. Option, the date is appended with phone no.	45	17
16.	On entering invalid numeral input in ”Number of days” at ”Check out” time, everything is calculated as 0	45	19
17.	On entering alphabet input in ”Number of days” at ”check out” time, wrong results are displayed.	30	21
18.	All available room of the desired type are not displayed for check in.	30	20

tomer involvement thus more amenable to accommodate the desired changes easily.

2. Agile methodology is also beneficial as it overtly emphasizes on highly interactive and frequent communication between developers and customers. This method also ensures dynamic development and facilitates quick delivery with a scope of continuous enhancement because the product is very flexible in nature.

3. We found that defects are identified at early stages of SDLC thereby avoiding any cost overrun. Software maintenance is the most expensive phase of SDLC as it utilizes the maximum share of the overall project costs.

4. Collaboration among various teams was seen as an issue in agile methodology because of the scattering of information in various systems which sometimes make it difficult for making daily decisions.

5. Further, velocity was also seen as an issue because composite applications, applications that consist of multiple components impose challenges. It may be difficult to track and resolve defect within time constraints in the products developed using agile methodology because a defect can have its roots spread in multiple parts of the system of the enterprise environment. Further, regression testing also becomes an issue post defect fixing which sometimes severely affects the quality of a software system.

6. We observed that product developed using scrum method was more interactive as the customer participate more during scrum process and the product was found to be more flexible. It was developed faster with scrum method because not much time was invested on initial planning and analysis. Feedback from the customer was constantly available throughout the product development which facilitated early defect detection.

7. We could clearly notice that the maintenance phase in IEM method was by and large costlier due to the less involvement from a customer during the development phase.

8. The product was also found to be less flexible using IEM method because initial planning was done in advance and accommodating the change requests was comparatively difficult.

9. It was also ascertained that defects detection using scrum was made available earlier than IEM method which further assisted in an overall reduction in maintenance cost. Hence, agile methods have a positive impact on maintenance due to the early defect detection.

Chapter 10

Conclusions

10.1 Introduction

The need of change performed during maintenance phase is essential for a software system to reside longer in the market. Whenever some maintenance activity is performed into the code and it is modified, it gives birth to a new software release that is a refined from the previous one. This phenomenon is known as the evolution of the software. Software developers involved into the maintenance task have to get familiar themselves with the source code and documentation of the software before modifying it. So, thorough understanding of source code is necessary for effective change implementation.

Software maintainability prediction is gaining more and more significance in the context of continuously evolving software to meet the customer's expectations. The main aim of this research was to develop and investigate various models, methods and metrics to improve overall maintainability of the software and reduce the maintenance cost. Empirical studies on proprietary software and open source software were conducted to build prediction models using OO metrics. The software developers can use these maintainability prediction models during the early phases of project development to measure the maintainability in advance. Project managers can use this information in planning proper resource allocation. Prediction models created in this research also provide subset of metrics which are more significant among large set of available metrics and any software development organization can use them in keeping their values under threshold in order to enhance life of the software. Various statistical techniques ranging from simple linear regression to more complex and non-linear machine learning techniques have been proposed in this research. Since there is no well accepted theory for predicting maintainability, more and more empirical studies should be conducted to guide the choice of technique and creating a generalized prediction model. In this research, comparative analysis of all prediction techniques have also been carried out in order to analyze whether one technique outperforms the another. Since the characteristics of open source software systems are entirely different from proprietary software systems, hence in this research a prediction model for open source software was also developed and validated separately which investigate the quality in general and maintainability of the open source software in particular using OO metrics. The conclusions derived from the research and the industrial applications of the work done in this thesis are summarized in the following subsections:

10.2 Major Findings

An overview of existing literature on OO metrics and maintainability prediction is presented in this thesis. Research methodology to carry out the empirical study while making prediction model for the purpose of maintainability prediction is also stated. The initial steps that need to be performed in any empirical studies have been described in detail. The importance of descriptive statistics as it helps to present the data in a more meaningful way is elaborated along with the method for simpler interpretations. The process of FSS which selects a subset of relevant features for the use in model construction is explained. Various advantages of FSS such as shorter training time, enhanced generalization, easier interpretation and reduction in over fitting are elaborated. Various steps to carry out an extensive empirical study such as identification of the dependent variables and independent variables, method for empirical data collection and selection of machine learning techniques have been described. The method for identifying whether there exists any correlation between independent and dependent variables is explain followed by the interpretation of various parameters used in this regard. Various statistical, machine learning and evolutionary techniques used for model prediction are described. We have also described various performance measures which are used for the model evaluation. Different statistical tests used for hypothesis testing are being explained in greater detail and finally, in this thesis we have also explained many methods to establish the significance of the test results using post hoc analysis such as Wilcoxon method, Friedman Method and Nemenyi testing methods.

In order to find out existing research done in the field of software maintenance, an overview of the work done in the literature is presented in an established systematic form. Databases of Inspec, IEEE Xplore and ACM Digital Library were searched to find the empirical studies related to this field. These studies were further explored and compiled from numerous points of view such as the use of metrics, datasets, methods, tools etc. Major thrust areas of the said field were identified which provided us the future directions. We have systematically summarized the empirical evidence obtained from the existing literature and provided a brief description of each study in terms of the independent variables, the data analysis techniques, the performance measures and the software used. We obtained the answers to various RQs which primarily focus on the following issues: 1) use of various techniques for predicting software maintainability; 2) type of design metrics used for predicting software maintainability 3) Kind of datasets used for predicting maintainability; 4) type of validation technique for validating the models; 5) various tools used to collect significant metrics and 6) use of performance measures for evaluating the performance of prediction models;

According to the systematic review results, we found that the use of machine learning techniques in predicting maintainability has increased since 2005. The use of evolutionary techniques has also begun in related sub-fields after 2010. We have observed that design metrics is still the most favoured option to capture the characteristics of any given software before deploying it further in prediction model for determining the corresponding software maintainability. A significant increase in the use of public dataset for making the prediction models has also observed and in this regard two public datasets UIMS and QUES proposed by Li and Henry [127] is quite popular among researchers. Although machine learning techniques are still the most popular methods, however, we observed that researchers engaged in research on software maintainability may experiment by using datasets from open source software systems with hybrid techniques. We concluded the review with an observation that more empirical studies are also required to be conducted on a large number of datasets so that a generalized theory could be made.

The GMDH technique is ideal for complex, unstructured system in obtaining a high order input-output relationship as it is heuristic in nature and not based on a solid foundation such as is regression analysis. The GMDH technique and its modified versions have been previously applied to a wide array of problems to ascertain predictions such as bio-informatics, weather forecast, software reliability etc. In this research study an attempt was made to apply this model perhaps for the first time for the task of software maintainability prediction using OO software design metrics and compare with prevailing prediction models proposed in last decade to ascertain their performance. GMDH technique could achieve 79% accuracy for QUES dada set 68% accuracy for UIMS dataset. It was also found that almost 69% predictions are less than the error of 0.25 prediction accuracy with GMDH technique and 72% predictions are less than the error of 0.30 prediction accuracy. In order to verify whether these results are significant and not coincidental, Wilcoxon significance test is also conducted and results are compared with fifteen machine learning techniques available in literature on the same dataset UIMS and QUES. GMDH was found to be significantly superior to eight techniques out of fourteen techniques.

There are a number of internal quality attributes such as coupling, cohesion, size, complexity etc. of a software, that are used to predict many external quality attributes of a software and maintainability is most significant one of them. However, with the advent of internet technologies, drastic increase in the use of mobiles and mobile based applications is observed. The metric suite developed by Chidamber and Kemerer [43] is not sufficient to capture the internal characteristics of the software build these days. Data intensive OO applications should be measured differently than the way they are currently measured. Hence in this study, deficiencies in Chidamber and Kemerer metric suite is identified and two new metrics NODBC and SCCR are being proposed to measure new applications. Univariate analysis found strong correlation (p-value as 0.000) between dependent variable change and new metrics. Multivariate analysis found the values of correlation coefficient as 0.915 (with NODBC) and 0.858 (with SCCR). Accuracy in MMRE is observed at 23%, 7%, 14%, 11%, and 19% for ABP system, FLM, EASY, SMS and IMS system respectively.

Recently interest in open source software has significantly increased in public and private sector of software organizations. It has numerous advantages such as less cost, more innovation, no copyright issues, better security, higher quality, faster adoption etc. In this research, we have explored various widely used open source software systems such as Ivy, Adbera, OpenCV, Poi, Rave, DrumKit, OrDrumBox, Log4J, JEdit, JUnit, JWebUnit an HuDoKu for predicting their software maintainability. Use of machine learning techniques in prediction modeling in various fields such as actuarial science, financial services, telecommunications, retail, travel, capacity planning, marketing, insurance, health-care etc. is increasing day by day. However, literature shows that maintainability prediction models are mostly based on statistical techniques and machine learning techniques are not much used for predicting software maintainability. In this view, we have compared the performance of 13 machine learning techniques on seven open source datasets. This extensive comparison provides an opportunity to fairly evaluate all the techniques and enables to judge the performance of one technique over the other. Prediction models were developed using seventeen most commonly used OO metrics and compared using four prediction accuracy measures MARE, RMSE, Pred(0.25) and Pred(0.75).

We found that FSS could reduced the size by 17%, 17%, 41%, 35%, 23%, 41% and 35% for Drumkit, OpenCV, Abdera, Ivy, Lo4j, JEdit and JUnit datasets respectively. The accuracy of all the machine learning techniques lies between the ranges of 39-77% on all datasets. It was also found that 72-78% predictions are less than the error of 25% and 66-89% predictions are less than the error of 75%. We also found that GGAL and GMDH techniques perform better than other machine learning techniques as GMDH technique achieved more than 70% accuracy with four out of seven datasets. It is also observed that four machine learning techniques PNN, GMDH, GRNN and GGAL have achieved less than 30% error. The outcome of the Friedman test indicates that the performance of GGAL and GMDH is best among all machine learning techniques as 17.9% of pairs was found to be significantly different using statistical test and not coincidental. With the help of post hoc analysis using Nemenyi Test, 12 pairs were found to be significantly different out of 78 pairs which is almost 15.3% of the total pairs. GGAL performed significantly better than K Star, Bagging, Kohonen Network, DT, JERN and SVM machine learning technique and GMDH performed significantly better than DT, JERN and SVM techniques.

One of the major contribution of this thesis is to analyze the predictive capability of evolutionary techniques for maintainability prediction. Although evolutionary techniques have been successfully applied in many fields other than software engineering with great success, an efficient and reliable evolutionary technique based prediction model for software maintainability is created for the first time. Prediction performance of evolutionary fuzzy, evolutionary neural and evolutionary neural symbolic regression methods with traditional statistical and machine learning models were compared and it was found that evolutionary techniques perform better than traditional techniques. Accuracy within the range of 75% to 78% could be achieved using evolutionary techniques. With the help of post hoc analysis using Nemenyi test the performance of evolutionary fuzzy techniques were compared and we found that 10.8% of pairs were significantly better and not coincidental. The results achieved in this chapter confirms that construction of evolutionary technique for software maintainability prediction is feasible, adaptable and useful in predicting software maintainability.

Refactoring is a technique that transforms various types of software artifacts for the purpose of improvement in internal structure of the software without affecting its external behavior. Although, it is is commonly applied after a significant amount of features are added, however the ripple effects of the applications of refactoring on maintainability of the software are yet to be explored. We conducted an empirical study in which the code of proprietary software was modified using refactoring methods and the effects on maintainability were adjudged. To achieve this goal, same software is measured before the application of certain refactoring method as well as after the application. Change in the values of the OO metrics is observed and mapped to the ascertain nature of maintainability behaviour. We observed that although 'Encapsulating Field' and 'Extract Method' refactoring increases values of WMC, LOC and RFC, however, maintainability increased because of the increased level of abstraction, increased understandability and increased reusability. It was also found that 'Extract Class' increases maintainability as classes becomes clearer, crisper and organized. There was no effect of 'Hide Method' refactoring on OO metrics. We found that decision about maintainability of any given code should not be made on the basis of only internal quality attributes measured though OO metrics but equal attentions should be given to external quality attributes such as Abstraction, Understandability, Modifiability, Extensibility, Reusability etc.

Agile methodologies are widely applied in software development nowadays both in small as well as large-scale organizations. Even though traditional life cycle processes such as the V model are quite predominant, however, the adoption rate of the agile methodology is also rapidly increasing. We found that there are no precise study available in literature which study can analyze its effects on software maintainability. Therefore, the goal of one of the case study was to render what the agile methods are and how can they can be applied while development of the software so as to maximize the maintainability of the software. In this empirical study, effects of agile processes on software maintainability were observed. We developed the same software using agile as well as non-agile method and compared both of them using certain metrics which are important from maintenance point of view. The results from this case study show that agile methods have positive impact on maintainability. We identified that the maintenance work is strongly influenced by how well the software was developed. We found that the agile processes are very helpful to keep track of software systems lifetime and its architecture and functionality. It was found that if this methodology is adopted during development time, error rate is reduced and less number of change request are received once the product enter into maintenance phase.

10.3 Applications of the Work

Software maintainability is one of the key quality attribute which determines the success of any software product. Since software maintainability is an important attribute of software quality, accurate prediction of it can help the practitioners to improve the overall quality of a software. In this research study, various ways including methods, metrics and models are suggested to improve the software maintainability of a given software. The results of this research would be helpful to developers, practitioners, project managers and academicians in following ways:

1. Feature subset selection using genetic algorithms as discussed in chapter 2^{nd} can be used by the developers to identify a subset of significant metrics instead of working on large set of available metrics. This data reduction not only increases the speed, but also the accuracy would also be increased.

2. The software maintainability prediction model created in the 4th and 7th chapter of this thesis using various machine learning techniques and evolutionary techniques can be used by the software professionals to identify the classes requiring more maintenance effort in the earlier stages of software development. Maintainability prediction has numerous applications such as schedule planning, cost estimation, quality assurance

testing, software debugging, budget preparation, and software performance optimization. The outcome of this investigation would be helpful for developers in order to predict maintenance behavior of the software at the earlier stages of SDLC and accordingly, they can optimize their resource allocations, prioritize maintenance tasks and produce high-quality low maintenance software systems.

3. With the help of software maintainability prediction model created in the 4^{th} and 7^{th} chapter, project managers can actually predict the the maintenance effort required during the maintenance phase. They can take benefit from this available information and plan the utilization of the resources accordingly. As the first hand information about the maintenance efforts is available at development stage only, better staff deployment can be planned. Practitioners can use and adapt the validated machine learning and evolutionary techniques based prediction models in this thesis for predicting software maintainability in the software industry.

4. Research on the usefulness of software metrics to adjudge the quality of the software has come a long way. Right from the initial days when traditional structural metrics were proposed for analyzing a given software till today when modern OO metrics are used for the same job, various metrics have been suggested and validated. In the 5^{th} chapter of this thesis, we have proposed a new set of metrics which is more significant for the data-intensive applications because instead of measuring only the object-oriented attributes, this new metric suite also measures the understandability of the databases. This newly proposed metrics suite for data intensive applications in this thesis can be use by the software industry to precisely predict software maintainability for mobile based applications where the amount of database handling is very very large.

5. In the 6^{th} chapter, we have provided empirical evidences that overall good prediction accuracy can be achieved by using the prediction models based on evolutionary

fuzzy techniques and they perform better than the other machine learning techniques in the context of open source software systems. During the development of traditional software, maintenance is considered as important factors and special efforts are taken during the early phase of software development process by the developers to control the maintaining efforts. In order to reduce the maintenance, closeness between the client and developer is highly emphasized so that correct requirements could be captured. Unfortunately, development process of open source software is entirely different from traditional software. The development of open source software is more difficult because of the absence of direct communication between the client and the developers. Thus the client does not know whether the developers have considered important aspects of software development in order to keep the maintenance cost under control or not. The study conducted in chapter 6, has identified empirical evidence which can possibly be used by vendor to predict the maintainability of particular open source software before adopting and deploying it at the client site.

6. In the 8^{th} chapter of this thesis, we have explored the effects of various refactoring methods on maintainability. In the software industry, re-engineering is always preferred, however, special care is taken to avoid over-engineering. In this chapter we have exactly calculated the effects which can be utilized by the software practitioners and researchers. They can use the guidelines and results obtained from the study to understand the effects of refactoring on each object-oriented metrics and they can make right decisions regarding the selection of appropriate refactoring method.

7. In the 9^{th} chapter of this thesis, a case study was conducted and presented to understand the effects of agile methodology on software maintainability. The information provided in this chapter would be helpful to projects managers as it demonstrate the usefulness of agile techniques in enhancing the software maintainability. We have also explained various metrics which should be considered to explore the quality of newly

developed software product in this chapter.

10.4 Future Work

There is enormous scope for empirical studies to know the level of maintenance requirement during the development phase of software product. One of the weaknesses of the current research is that the maintenance prediction models consider cumulative effects of all OO metrics on change. So investigating how individual OO metric affects the maintenance effort is another possible future extension of this work. Similar type of empirical validations with different metrics, datasets and prediction techniques need to be carried out so that generalized conclusions can be drawn based on the obtained results.

In this research, most of the OO metrics used while creating software maintainability prediction models are evaluated from the static source code based analysis. However, static metrics are insufficient in evaluating the dynamic behavior of a software application because the behavior of a software application is not only influenced by the complexity but also by the operational or run time environment of the source code. Therefore it is important to develop models which can provide us optimal maintainability prediction duly incorporating the dynamic metrics. This is one of the possible future extensions of our work. In the last couple of decades, many metrics have developed to measure the quality of OO software. Over the years, large number of such metrics have been proposed, developed and empirically validated. However, for the recently developed mobile based applications which are purely communication oriented, new metrics needs to be proposed and validated.

Bibliography

[1] ABDELMOEZ, W., GOSEVA-POPSTOJANOVA, AND AMMAR, H. Maintainability based risk assessment in adaptive maintenance context. In *2nd International Predictor Models in Software Engineering Workshop, PROMISE* (2006), pp. 1-17, Philadelphia, PA.

[2] ABREU, F., AND CARAPUCA, R. Candidate metrics for object-oriented software within a taxonomy framework. *Journal of Systems and Software 26*, 1 (1994), pp. 87-96.

[3] AGGARWAL, K. K., AND SINGH, Y. Software engineering programs documentation, operating procedures. *New Age international publishers* (2008).

[4] AGGARWAL, K. K., SINGH, Y., CHANDRA, P., AND PURI, M. Measurement of software maintainability using a fuzzy model. *Journal of Computer Sciences 1*, 4 (2005), pp. 538-542.

[5] AGGARWAL, K. K., SINGH, Y., AND CHHABRA, J. K. An integrated measure of software maintainability. In *Proceeding of Annual Reliability and maintainability symposium* (2002), IEEE, pp. 235-241, Seattle, WA.

[6] AGGARWAL, K. K., SINGH, Y., KAUR, A., AND MALHOTRA, R. Application of artificial neural network for predicting maintainability using object- oriented metrics. *World Academy of Science 15*, 1 (2006), pp. 285-289.

[7] Alba, E. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation 6*, 5 (2002), pp. 443-462.

[8] Arisholm, E., and Sjoberg, D. Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented software. *IEEE Trans. Software Engineering 30*, 8 (2004), pp. 11-25.

[9] Azar, D., and Vybihal, J. An ant colony optimization algorithm to improve software quality prediction models. *Case of class stability, Journal of Information and Software Technology 53*, 4 (2011), pp. 388-393.

[10] Babu, S., and Parvathi, R. Design dynamic coupling measurement of distributed object oriented software using trace events. *Journal of Computer Science 7*, 5 (2011), pp. 770-778.

[11] Back, T., Fogel, D. B., and Michalewicz, Z. *Handbook of evolutionary computation.* IOP Publishing Ltd., 1997.

[12] Balogh, G., Zoltan, A., and Baszedes, A. Prediction of software development effort enhanced by a genetic algorithm. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)* (2015), pp. 28–30, Bremen, Germany.

[13] Bandi, R., Vaishnavi, V., and Turk, D. Predicting maintenance performance using object-oriented design complexity metrics. *IEEE Transaction on Software Engineering 29*, 1 (2003), pp. 77-87.

[14] Banker, R. D., M, M. S., Kemerer, C. F., and Zweig, D. Software complexity and maintenance costs. *Communications of the ACM 36*, 11 (1993), pp. 81-95.

[15] Bansiya, J., and Davis, C. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering 28*, 1 (2002), pp. 4-17.

[16] BAQAIS, B., ALSHAYEB, M., AND BAIG, Z. Hybrid intelligent model for software maintenance prediction. In *Proceedings of the International Conference on World Congress on Engineering* (2013), pp. 358–362, London, UK.

[17] BASGALUPP, M. P., BARROS, R. C., AND RUIZ, D. D. Predicting software maintenance effort through evolutionary-based decision trees. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing* (2012), ACM, pp. 1209-1214, Riva (Trento), Italy.

[18] BECK, K. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2000.

[19] BELADY, L., AND LEHMAN, M. A model of large program development. *IBM Systems journal 15*, 3 (1976), pp. 225-252.

[20] BENLARBI, S., AND MELO, W. Polymorphism measures for early risk prediction. In *Proceedings of the 21st International Conference on Software Engineering (ICSE)* (1999), pp. 335–344, Los Angeles, USA.

[21] BENNETT, K. H., AND RAJLICH, V. T. Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (2000), ACM, pp. 73-87, Limerick, Ireland.

[22] BERNS, G. M. Assessing software maintainability. *Communications of the ACM 27*, 1 (1984), pp. 14-23.

[23] BHATT, P., SHROFF, G., AND MISRA, A. Dynamics of software maintenance. *ACM SIGSOFT Software Engineering Notes 29*, 5 (2004), pp. 1-5.

[24] BHATTACHARYA, P., AND NEAMTIU, I. Assessing programming language impact on development and maintenance: A study on c and c++. In *Proceedings of the 33rd*

International Conference on Software Engineering (ICSE) (2011), IEEE, pp. 171-180, Honolulu, Hawaii, USA.

[25] BIEMAN, J. M., AND KANG, B. K. Cohesion and reuse in an object-oriented system. *ACM SIGSOFT Software Engineering Notes 20*, SI (1995), pp. 259-262.

[26] BIEMAN, J. M., AND OTT, L. M. Measuring functional cohesion. *IEEE Transactions on Software Engineering 20*, 8 (1994), pp. 644-657.

[27] BLAND, J. M., AND ALTMAN, D. G. Multiple significance tests: the bonferroni method. *Bmj 310*, 6973 (1995), pp. 170-191.

[28] BOEHM, B. Get ready for agile methods with care. *IEEE Computer 35*, 1 (2002), pp. 64-69.

[29] BOIS, B. D., DEMEYER, S., AND VERELST, J. Refactoring-improving coupling and cohesion of existing code. In *Proceedings of the 11th Working Conference on Reverse Engineering, WCRE* (2004), pp. 144–151, Delft, The Netherlands.

[30] BOIS, B. D., AND MENS, T. Describing the impact of refactoring on internal program quality. In *Proceedings of the International Workshop on Evolution of Large-scale Industrial Software Applications ELISA* (2003), pp. 37–48, Amsterdam, The Netherlands.

[31] BOSCH, J., AND BENGTSSON, P. Assessing optimal software architecture maintainability. In *Fifth European Conference on Software Maintenance and Reengineering* (2001), IEEE, pp. 168-175, Lisbon, Portugal.

[32] BRAVO, F. A. *Logic Meta-Programming Framework for Supporting the Refactoring Process*. PhD thesis, Vrije University, Brussel, 2003.

[33] BREIMAN, L., FRIEDMAN, J., C.J.STONE, AND RICHARD, A. *Classification and Regression Trees*. CRC press, 1984.

[34] BRIAND, L. C., BUNSE, C., AND DALY, J. A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs. *IEEE Transactions on Software Engineering 27*, 6 (2001), pp. 513-530.

[35] BRIAND, L. C., DEVANBU, P., AND MELO, W. An investigation into coupling measures for c++. In *Proceedings of the International Conference on Software Engineering ICSE* (1997), pp. 513-530, Bostan, USA.

[36] BROOMHEAD, D. S., AND LOWE, D. Radial basis functions, multi-variable functional interpolation and adaptive networks. Tech. rep., DTIC Document, 1988.

[37] BROY, M., DEISSENBOECK, F., AND PIZKA, M. Demystifying maintainability. In *Proceedings of the international workshop on Software quality* (2006), ACM, pp. 21-26,Shanghai, China.

[38] BRYSON, A. E. *Applied Optimal Control: Optimization, Estimation and Control*. CRC Press, 1975.

[39] BURKI, C. J., AND HARALD, H. V. How to save on software maintenance costs. *Technical Report : Omnext White Paper* (2014), pp. 1-16.

[40] CAPER, J. *The Economics of Software Maintenance*. Twenty First Century Version 3, 1950. http://www.compaid.com/ caiinternet/ ezine/ capersjones.

[41] CHAPIN, N. Do we know what preventive maintenance is? In *In Proceedings of International Conference on Software Maintenance,ICSM* (2000), IEEE, pp. 1–15, San Jose, CA, USA.

[42] CHEN, J., AND LUM, J. A new metrics for object-oriented design. *Journal of Information of Software Technology 35*, 4 (1993), pp. 232-240.

[43] CHIDAMBER, S. R., AND KEMERER, C. R. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering 20*, 6 (1994), pp. 476-493.

[44] CHOW, T. S., AND CAO, D. B. A survey study of critical success factors in agile software projects. *Journal of System Software 81*, 6 (2008), pp. 961-971.

[45] CHUCHER, N. I., AND MARTIN, J. S. Comments on a metrics suite for object-oriented design. *IEEE Transaction on Software Engineering 21*, 3 (1995), pp. 263-265.

[46] C.J.BURGESS, AND M.LEEY. Can genetic programming improve software effort estimation? a comparative evaluation. *Information and Software Technology 43*, 14 (2001), pp. 863-873.

[47] COCKBURN, A. *Crystal clear: A human-powered Methodology for Small Teams*. Pearson Education, 2004.

[48] COHN, M., AND FORD, D. Introducing an agile process to an organization. *Computer 36*, 6 (2003), pp. 74-78.

[49] COLEMAN, D., ASH, D., LOWTHER, B., AND OMAN, P. Using metrics to evaluate software system maintainability. *Computer 27*, 8 (1994), pp. 44-49.

[50] CONTE, S. D., DUNSMORE, H. E., HUBERT, E., AND Y.VINCENT. *Software engineering metrics and models*. Benjamin-Cummings Publishing Co., Inc., 1986.

[51] CORTES, C., AND VAPNIK, V. Support-vector networks. *Machine learning 20*, 3 (1995), pp. 273-297.

[52] CUNNINGHAM, W., AND BECK, K. Using pattern languages for object-oriented programs. In *Proceedings of Object-Oriented Programming, Systems, Languages & Application, OOPSLA* (1987), pp. 87–130, Orlando, Florida.

[53] DAGPINAR, M., AND JAHNKE, J. H. Predicting maintainability with object-oriented metrics-an empirical comparison. In *Proceedings of the 10th IEEE Working Conference on Reverse Engineering, WCRE* (2003), pp. 155–170, Victoria, B.C., Canada.

[54] DAHIYA, S. S., CHHABRA, J. K., AND KUMAR, S. Use of genetic algorithm for software maintainability metrics conditioning. In *Proceeding of International Conference on Advanced Computing and Communications (ADCOM)* (2007), IEEE, pp. 87-92, Guwahati, India.

[55] DALY, J., BROOKS, A., MILLER, J., ROPER, M., AND WOOD, M. Evaluating inheritance depth on the maintainability of object-oriented software. *Empirical Software Engineering 1*, 2 (1996), pp. 109-132.

[56] DAVIS, N. Driving quality improvement and reducing technical debt with the definition of done. In *Proceedings of the IEEE Agile Conference (AGILE)* (2013), pp. 164–168.

[57] DEISSENBOECK, F., WAGNER, S., PIZKA, M., TEUCHERT, S., AND GIRARD, J.-F. An activity-based quality model for maintainability. In *Proceeding of International Conference on Software Maintenance,ICSM 2007* (2007), IEEE, pp. 184-193, Paris, France.

[58] DELIGIANNIS, I., SHEPPERD, M., ROUMELIOTIS, M., AND STAMELOS, I. An empirical investigation of an object-oriented design heuristic for maintainability. *Journal of Systems and Software 65*, 2 (2003), pp. 127-139.

[59] DEMVSAR, J. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research 7* (2006), pp. 1-30.

[60] DONELL, S. G. M. Establishing relationships between specification size and software process effort in case environments. *Information and Software Technology 39*, 1 (1997), pp. 35-45.

[61] DUBEY, S. K., RANA, A., AND DASH, Y. Maintainability prediction of object oriented software system by using artificial neural network approach. *International Journal of Soft Computing and Engineering (IJSCE) 2*, 2 (2012), pp. 420–423.

[62] ELISH, M. O., AND ELISH, K. O. Application of treenet in predicting object-oriented software maintainability: A comparative study. In *Proceedings of the 13th European Conference on Software Maintenance and Reengineering, CSMR* (2009), pp. 69–78, Kaiserslautern, Germany.

[63] ENGINEERING, I. S. S. *IEEE Std. 828-1998 IEEE Standard for Software Configuration Management Plans standard.* Standards Committee of the IEEE Computer Society, 1998.

[64] FAN, R. E., AND P. H. CHEN, C. J. L. Working set selection using second order information for training support vector machines. *The Journal of Machine Learning Research 6* (2005), pp. 1889-1918.

[65] FENTON, N., AND BIEMAN, J. *Software Metrics: A Rigorous and Practical Approach: Brooks*. CRC Press, 2014.

[66] FERNANDO, A. B., AND ROGERIO, C. Candidate metrics for object-oriented software within a taxonomy framework. *Journal of Systems Software 26*, 1 (1994), pp. 87-96.

[67] FERNELEY, E. H. Design metrics as an aid to software maintenance: an empirical study. *Journal of Software Maintenance: Research and Practice 11*, 1 (1999), pp. 55-72.

[68] FIORAVANTI, F., AND NESI, P. Estimation and prediction metrics for adaptive maintenance effort of object-oriented systems. *IEEE Transactions on Software Engineering 27*, 12 (2001), pp. 1062–1084.

[69] FOWLER, M. *Refactoring: Improving the Design of Existing Code*. Pearson Education India, 1999.

[70] FOWLER, M., AND HIGHSMITH, J. The agile manifesto. *Journal of Software Development 9*, 8 (2001), pp. 28-35.

[71] FREUND, Y., SCHAPIRE, R., AND ABE, N. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence 14*, 5 (1999), pp. 771-780.

[72] FRIEDMAN, M. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics 11*, 1 (1940), pp. 86-92.

[73] GAMALIELSSON, J., AND OLSSON, B. Evaluating protein structure prediction models with evolutionary algorithms. In *Book Series: Information Processing with Evolutionary Algorithms* (2005), pp. 143–158.

[74] GARCIA-PEDRAJAS, N., GARCAOSORIO, C., AND FYFE, C. Nonlinear boosting projections for ensemble construction. *The Journal of Machine Learning Research 8* (2007), pp. 1-33.

[75] GAWALI, A. R. *Impact of Agile Software Development Model on Software Maintainability*. PhD thesis, Walden University, 2012.

[76] GENERO, M., PIATTINI, M., MANSO, E., AND CANTONE, G. Building uml class diagram maintainability prediction models based on early metrics. In *Proceedings of the 9th international symposium on software metrics, METRICS'03* (2003), IEEE Computer Society, pp. 263-275, Washington, DC.

[77] GEPPERT, B., MOCKUS, A., AND ROBLER, F. Refactoring for changeability: A way to go? In *Proceedings of the 11th IEEE International Symposium on Software Metrics* (2005), pp. 10–14, Como, Italy.

[78] GHOSH, S., RANA, A., AND KUMAR, A. Comparative study of the factors that affect maintainability. *International Journal on Computer Science and Engineering 3*, 12 (2011), pp. 3763-3769.

[79] GOODMAN, P. *Practical Implementation of Software Metrics*. London: McGraw-Hill, 1993.

[80] GRADY, R. B. Successfully applying software metrics. *Computer 27*, 9 (1994), pp. 18-25.

[81] GROUP, W. S. Artificial intelligence, genetic algorithm and neural network software for predicting, forecasting, classification and optimization. *Mathwork systems* (2012).

[82] HALL, M. A. *Correlation-based Feature Selection for Machine Learning*. Doctoral dissertation at the University of Waikato, Hamilton, New Zealand, 1999.

[83] HALSTEAD, H. *Elements of Software Science*. Elsevier North-Holland, ISBN 0-444-00205-7, 1977.

[84] HANENBERG, S., KLEINSCHMAGER, S., ROBBES, R., TANTER, E., AND STEFIK, A. An empirical study on the impact of static typing on software maintainability. *Empirical Software Engineering 19*, 5 (2014), pp. 1335-1382.

[85] HARRISON, R., COUNSELL, S., AND NITHI, R. Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems. *Journal of Systems and Software 52*, 2 (2000), pp. 173–179.

[86] HATTON, L. How accurately do engineers predict software maintenance tasks? *Computer 40*, 2 (2007), pp. 64–69.

[87] HAYES, J. H., AND ZHAO, L. Maintainability prediction: a regression analysis of measures of evolving systems. In *Proceedings of the 21st International Conference on Software Maintenance, ICSM* (2005), IEEE, pp. 601-604, Budapest, Hungary.

[88] HEGEDUS, P. Revealing the effect of coding practices on software maintainability. In *29th IEEE International Conference on in Software Maintenance (ICSM)* (2013), IEEE, pp. 578–581, Eindhoven, Netherlands.

[89] HENDERSON-SELLERS, B. *Object-oriented Metrics: Measures of Complexity*. Prentice-Hall, Inc., 0-13-239872-9, 1996.

[90] HENRY, S., AND KAFURA, D. Software structure metrics based on information flow. *IEEE Transactions on Software Engineering 7*, 5 (1981), pp. 510-518.

[91] HIROTA, T., TOHKI, M., OVERSTREET, M., HASHIMOTO, M., AND CHERINKA, R. An approach to predict software maintenance cost based on ripple complexity. In *Proceedings of Firrst Asia-Pacific Software Engineering Conference* (1994), IEEE, pp. 439-444, Tokyo, Japan.

[92] HITZ, M., AND MONTAZERI, B. Measuring coupling and cohesion in object-oriented systems. *IEEE Transactions on Software Engineering 22*, 4 (1996), pp. 267-271.

[93] HOFFMANN, J. P., AND SHAFER, K. *Linear regression analysis: Assumptions and applications*. Department of Sociology Brigham Young University, 2005.

[94] HU, M. C. J. *Application of the adaline system to weather forecasting*. Master Thesis, Technical Report, Stanford Electronics Laboritries, Department of Sociology, Brigham Young University, 1964.

[95] ISO-9126, I. S. *Software Production Evaluation Quality Characteristics and guidelines for their Use*. Standards Committee of the ISO Society, 1991.

[96] IVAKHNENKO, A. G., AND KOPPA, Y. U. Regularization of decision functions in the group method of data handling. *Soviet Automatic Control 15*, 2 (1970), pp. 28-37.

[97] JAMES, M. http://scrumreferencecard.com/scrum-reference-card.

[98] JEET, K., DHIR, R., AND VERMA, H. A comparative study of bayesian and fuzzy approach to assess and predict maintainability of the software using activity-based quality model. *ACM SIGSOFT Software Engineering Notes 37*, 3 (2012), pp. 1-9.

[99] JIEPING, Y., JANARDAN, R., AND LI, Q. Two-dimensional linear discriminant analysis. In *Advances in neural information processing systems* (2004), pp. 1569–1576, University of Pennsylvania, Philadelphia, USA.

[100] JIN, C., AND LIU, J. A. Applications of support vector mathine and unsupervised learning for predicting maintainability using object-oriented metrics. In *Proceedings of the Second International Conference on Multimedia and Information Technology (MMIT)* (2010), pp. 24–27, Kaifeng, China.

[101] JORGENSEN, M. Experience with the accuracy of software maintenance task effort prediction models. *IEEE Transactions on Software Engineering 21*, 8 (1995), pp. 674-681.

[102] KABAILI, H., KELLER, R., AND LUSTMAN, F. Cohesion as changeability indicator in object-oriented systems. In *Fifth European Conference on Software Maintenance and Reengineering* (2001), IEEE, pp. 39-46, Lisbon, Portugal.

[103] KAJKO-MATTSSON, M., AND NYFJORD, J. A model of agile evolution and maintenance process. In *Proceedings of the 42nd Hawaii International Conference on System Sciences, HICSS* (2009), pp. 1–10, Waikoloa, Hawaii.

[104] KATAOKA, Y., IMAI, T., ANDOU, H., AND FUKAYA, T. A quantitative evaluation of maintainability enhancement by refactoring. In *Proceedings of the IEEE International Conference on Software Maintenance ICSM* (2002), pp. 576–585, Montreal, Canada.

[105] KAUR, A., AND KAUR, K. Statistical comparison of modelling methods for software maintainability prediction. *International Journal of Software Engineering and Knowledge Engineering 23*, 06 (2013), pp. 743-774.

[106] KAUR, A., KAUR, K., AND MALHOTRA, R. Soft computing approaches for prediction of software maintenance effort. *International Journal of Computer Applications 1*, 16 (2010), pp. 80-86.

[107] KAUR, A., KAUR, K., AND PATHAK, K. Software maintainability prediction by data mining of software code metrics. In *International Conference on Data Mining and Intelligent Computing ICDMIC)* (2014), pp. pp. 1-6, New Delhi,India.

[108] KAUR, K., AND SINGH, H. Determination of maintainability index for object oriented systems. *ACM SIGSOFT Software Engineering Notes 36*, 2 (2011), pp. 1-6.

[109] KEMERER, C. F., AND SLAUGHTER, S. Determinants of software maintenance profiles: An empirical investigation. *Journal of Software Maintenance 9*, 1 (1997), pp 235–251.

[110] KITCHENHAM, B. A., PICKARD, L. M., LESLEY, M., MACDONELL, S. G., AND SHEPPERD, M. J. What accuracy statistics really measure software estimation. *IEEE software Proceedings 148*, 3 (2001), pp. 81-85.

[111] KITCHENHAM, B. A., RIALETTE, P., DAVID, B., BRERETON, O. P., TURNER, M., NIAZI, M., AND LINKMAN, S. Systematic literature reviews in software engineering a tertiary study. *Information and Software Technology 52*, 8 (2010), pp. 792–805.

[112] KNIBERG, H., AND FARHANG, R. Bootstrapping scrum and xp under crisis a story from the trenches. In *Proceedings of the Agile (AGILE)* (2008), pp. 436–444, Girona, Spain.

[113] KNIPPERS, D. *Agile Software Development and Maintainability*. PhD thesis, Universiteit Twente, 2011.

[114] KOH, T. W., SELAMAT, M. H., GHANI, A., AZIM, A., AND RUSLI, A. Review of complexity metrics for object oriented software products. *IJCSNS Int J of Computer Science and Network Security 8*, 11 (2008), pp. 314-320.

[115] KOHAVI, R., AND JOHN, G. H. Wrappers for feature subset selection. *Artificial intelligence 97*, 1 (1997), pp. 273–324.

[116] KOHAVI, R., AND SOMMERFIELD, D. Targeting business users with decision table classifiers. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining, KDD* (1998), pp. 249–253, New York, USA.

[117] KOHONEN, T. The self-organizing map. *Proceedings of the IEEE 78*, 9 (1990), pp. 1464–1480.

[118] KOTEN, V. C., AND GRAY, A. R. An application of bayesian network for predicting object-oriented software maintainability. *Information and Software Technology 48*, 1 (2006), pp. 59-67.

[119] K.SHIBATA, RINSAKA, K., DOHI, T., AND OKAMURA, H. Quantifying software maintainability based on a fault-detection / correction model. In *Proceedings of Symposium on 13th Pacific Rim International on Dependable Computing* (2007), pp. 35-42, Melbourne, Victoria, AUSTRALIA.

[120] KUMAR, L. Predicting object-oriented software maintainability using hybrid neural network with parallel computing concept. In *Proceedings of the 8th India Software Engineering Conference, ISEC* (2015), pp. 100–109, Banglore, India.

[121] KUMAR, R., AND DHANDA, N. Maintainability measurement model for object oriented design. *International Journal of Advanced Research in Computer and Communication Engineering 4*, 5 (2015), pp. 331-340.

[122] LEE, S. W., AND SONG, H. H. A new recurrent neural-network architecture for visual pattern recognition. *IEEE Transactions on Neural Networks 8*, 2 (1997), pp. 331-340.

[123] LEE, Y., LIANG, B., WU, S., AND WANG, F. Measuring the coupling and cohesion of an object-oriented program based on information flow. In *In Proceedings of the International Conference on Software Quality* (1995), pp. 81–90, Maribor, Slovenia.

[124] LESSMANN, S., BAESENS, B., MUES, C., AND PIETSCH, S. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering 34*, 4 (2008), pp. 485-496.

[125] LI, F., MORGAN, R., AND WILLIAMS, D. Hybrid genetic approaches to ramping rate constrained dynamic economic dispatch. *Electric Power Systems Research 43*, 2 (1997), pp. 97-103.

[126] LI, W. Another metric suite for object-oriented programming. *Journal of Systems and Software 44*, 2 (1998), pp. 155–162.

[127] LI, W., AND HENRY, H. Object-oriented metrics that predict maintainability. *Journal of systems and software 23*, 2 (1993), pp. 111-122.

[128] LI, W., HENRY, S., KAFURA, D., AND SCHULMAN, R. Measuring object-oriented design. *Journal of Object-Oriented Programming 8*, 4 (1995), pp. 48-55.

[129] LIENTZ, B. P., AND SWANSON, E. B. Problems in application software maintenance. *Communications of the ACM 24*, 11 (1981), pp. 763-769.

[130] LIM, J., JEONG, S., AND SCHACH, S. An empirical investigation of the impact of the object-oriented paradigm on the maintainability of real-world mission-critical software. *Journal of System Software 77*, 1 (2005), pp. 131–138.

[131] LIN, J.-C., AND WU, K.-C. A model for measuring software understandability. In *The Sixth IEEE International Conference on Computer and Information Technology* (2006), IEEE, pp. 192-192, Seoul, South Korea.

[132] LORENZ, M., AND KIDD, J. *Object-oriented software metrics: A Practical Guide*. Prentice-Hall, Inc., 1994.

[133] LUCIA, A. D., POMPELLA, E., AND STEFANUCCI, S. Assessing effort estimation models for corrective maintenance through empirical studies. *Information and Software Technology 47*, 1 (2005), pp. 3-15.

[134] MALHOTRA, R. *Empirical Validation of Object-Oriented Metrics for Predicting Quality Attributes*. PhD thesis, University School of Information Technology, Guru Gobind Singh Indraprastha University, Kashmere Gate, Delhi-110403, (2009).

[135] MALHOTRA, R. *Empirical Research in Software Engineering: Concepts, Analysis, and Applications*. Chapman and Hall, CRC Press, ISBN 9781498719728, 2015.

[136] MALHOTRA, R., AND CHUG, A. An empirical validation of group method of data handling on software maintainability prediction using object oriented systems. In *Proceedings of the 6th International Conference on Quality, Reliability and Infocom Technology (ICQRIT)* (2012), pp. 348–351, New Delhi, India.

[137] MALHOTRA, R., AND CHUG, A. Software maintainability prediction using machine learning algorithms. *Software Engineering: An International Journal (SEIJ) 2*, 2 (2012), pp. 19-36.

[138] MALHOTRA, R., AND CHUG, A. Metric suite for predicting software maintainability in data intensive applications. *Book Chapter in Transactions on Engineering Technologies, Springer 5*, 2 (2013), pp. 165-173.

[139] MALHOTRA, R., AND CHUG, A. Application of evolutionary algorithms for software maintainability prediction using object-oriented metrics. In *Proceedings of the 8th International Conference on Bioinspired Information and Communications Technologies* (2014), pp. 348–351, Boston, USA.

[140] MALHOTRA, R., AND CHUG, A. Application of group method of data handling model for software maintainability prediction using object oriented systems. *Inter-*

national Journal of System Assurance Engineering and Management 5, 2 (2014), pp. 165-173.

[141] MALHOTRA, R., AND CHUG, A. An empirical study to redefine the relationship between software design metrics and maintainability in high data intensive applications. In *Proceedings of the World Congress on Engineering and Computer Science* (2014), pp. 161–175, San Francisco, USA.

[142] MALHOTRA, R., AND CHUG, A. Benchmarking framework for maintainability prediction of open source software using object oriented metrics. *International Journal of Innovative Computing, Information and Control 12*, 2 (2016), pp.615-634.

[143] MALHOTRA, R., AND CHUG, A. Comparative analysis of agile methodology and iterative enhancement model in assessment of software maintenance. In *Proceedings of the Computing for sustainable Global Development,IndiaCom 2016, Conference ID 37465* (2016), pp. 1–7, New Delhi, India.

[144] MALHOTRA, R., AND CHUG, A. An empirical study to assess the effects of refactoring on software maintainability. In *International Conference on Advances in Computing, Communications and Informatics, ICACCI-2016, IEEE Conference No. 38419* (2016), pp. 110–117, Jaipur, India.

[145] MALHOTRA, R., AND CHUG, A. Software maintainability: Systematic literature review and current trends. *International Journal of Software Engineering and Knowledge Engineering 26*, 8 (2016), pp. 1221-1253.

[146] MALHOTRA, R., CHUG, A., AND KHOSLA, P. Prioritization of classes for refactoring: A step towards improvement in software quality. In *Proceedings of the Third International Symposium on Women in Computing and Informatics* (2015), pp. 228–234, Kotchi, India.

[147] MALHOTRA, R., PRITAM, N., NAGPAL, K., AND UPMANYU, P. Defect collection and reporting system for git based open source software. In *Proceedings of the International Conference on Data Mining and Intelligent Computing (ICDMIC)* (2014), pp. 1–7, New Delhi, India.

[148] MARCO, T. D. *Controlling Software Projects, Management Measurement & Estimation*. Prentice Hall PTR Upper Saddle River, NJ, USA, 1986.

[149] MARI, M., AND EILA, N. The impact of maintainability on component-based software systems. In *Proceedings of the 29th Conference on EUROMICRO* (2003), IEEE Computer Society, pp. 25-31, Washington, DC, USA.

[150] MARTIN, J., AND MCCLURE, C. L. *Software Maintenance: The Problems and Its Solutions*. Prentice Hall Professional Technical Reference, 1983.

[151] MARTIN, R. O o design quality metrics. *An analysis of dependencies 12* (1994), pp. 151-170.

[152] MARTIN, R. C. *Clean code: A handbook of agile software craftsmanship*. Pearson Education, 2009.

[153] MARTINEZ, E. A., MARTINEZ, F. E., MARTINEZ, C. H., AND GARCIA, N. P. Evolutionary product unit based neural networks for regression. *Neural Networks 19*, 4 (2006), pp. 477-486.

[154] MATTSSON, M. K. Can we learn anything from hardware preventive maintenance? In *iceccs* (2001), IEEE.

[155] MAYER, T., AND HALL, T. A critical analysis of current oo design metrics. *Software Quality journal 8*, 2 (1999), pp. 97-110.

[156] MCCABE, J. T. A complexity measure. *IEEE Transactions on Software Engineering 2*, 4 (1976), pp. 308-320.

[157] McCall, J., Richards, P. K., and Walters, G. F. *Factors in Software Quality*. Information Systems Programs, General Electric Company, 1977.

[158] Mens, T., and Tourwe, T. A survey of software refactoring. *IEEE Transactions on Software Engineering 30*, 2 (2004), pp. 126-139.

[159] Miller, G. F., Todd, P. M., and Hegde, S. U. Designing neural networks using genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms* (1989), pp. 379–384, Virginia, USA.

[160] Miller, J. *Techniques of program and system maintenance*. Winthrop Publishers, 1981.

[161] Mishra, S., and Sharma, A. Maintainability prediction of object oriented software by using adaptive network based fuzzy system technique. *International Journal of Computer Applications 119*, 9 (2015), pp. 50-73.

[162] Misra, S. C. Modeling design/coding factors that drive maintainability of software systems. *Software Quality Journal 13*, 3 (2005), pp. 297-320.

[163] Moser, R., Sillitti, A., Abrahamsson, P., and Succi, G. Does refactoring improve reusability? In *Reuse of Off-the-Shelf Components* (2006), pp. 287–297.

[164] Muthanna, S., Kontogiannis, K., Ponnambalam, K., and Stacey, B. A maintainability model for industrial software systems using design level metrics. In *Proceedings of Seventh Working Conference on Reverse Engineering, 2000* (2000), IEEE, pp. 248-256, Washington, DC, USA.

[165] Myrtveit, I., Stensrud, E., and Shepperd, M. Reliability and validity in comparative studies of software prediction models. *IEEE Transactions on Software Engineering 31*, 5 (2005), pp. 380-391.

[166] NIESSINK, F., AND VLIET, H. V. Predicting maintenance effort with function points. In *Proceedings of the International Conference on Software Maintenance ICSM 1997* (1997), IEEE, pp. 32-39, Bari, Italy.

[167] NOSEK, J. T., AND PALVIA, P. Software maintenance management: changes in the last decade. *Journal of Software Maintenance: Research and Practice 2*, 3 (1990), pp. 157-174.

[168] OMAN, P., AND HAGEMEISTER, J. Metrics for assessing a software system's maintainability. In *Proceedings of the International Conference on Software Maintenance, ICSME* (1992), pp. 337–344, Goteborg, Sweden.

[169] OMAN, P., AND HAGEMEISTER, J. Construction and testing of polynomials predicting software maintainability. *Journal of Systems and Software 24*, 3 (1994), pp. 251-266.

[170] OPDYKE, W. F. *Refactoring: A program restructuring aid in designing object-oriented application frameworks*. PhD thesis, University of Illinois at Urbana-Champaign, 1992.

[171] PENG, W. W., AND WALLACE, D. R. *Software Error Analysis*. Silicon Press, 1995.

[172] PING, L. A quantitative approach to software maintainability prediction. In *Proceedings of the International Forum on Information Technology and Applications (IFITA)* (2010), vol. 1, pp. 105–108, Kumning, China.

[173] PIZKA, M., AND DEISENBOCK, F. How to effectively define and measure maintainability. *Software Management European Forum* (2007), 21–28.

[174] POLO, M., PIATTINI, M., AND RUIZ, F. Using code metrics to predict maintenance of legacy programs: A case study. In *Proceedings of the IEEE International*

Conference on Software Maintenance (ICSM'01) (2001), IEEE Computer Society, pp. 202-208, Washington, DC, USA.

[175] POOLE, C., AND HUISMAN, J. W. Using extreme programming in a maintenance environment. *IEEE Software 18*, 6 (2001), pp. 42-50.

[176] PRASANTH, N. N., RAJA, S., BIRLA, X., NAVAZ, K., AND RAHUMAN, S. Improving software maintainability through risk analysis. *International Journal of Recent Trends in Engineering 2*, 4 (2009), pp. 198-200.

[177] PRECHELT, L., UNGER, B., PHILIPPSEN, M., AND TICHY, W. Re-evaluating inheritance depth on the maintainability of object-oriented software. *International Journal of Empirical Software Engineering* (1998), pp. 1-16.

[178] PRIES-HEJE, L., PRIES-HEJE, J., AND DALGAARD, B. Scrum code camps. In *Proceedings of the Agile Conference (AGILE)* (2013), pp. 64–73, Bishopsgate, London, UK.

[179] RAJARAMAN, C., AND LYU, M. R. Reliability and maintainability related software coupling metrics in c++ programs. In *Third International Symposium on Software Reliability Engineering* (1992), IEEE, pp. 303-311, Research Triangle Park, NC, USA.

[180] RAMIL, F., JUAN, LOZANO, A., WERMELINGER, M., AND CAPILUPPI, A. Empirical studies of open source evolution. In *Book Chapter : Software evolution, Springer* (2008), pp. 263–288.

[181] RAMIL, J. F., AND SMITH, N. Qualitative simulation of models of software evolution. *Software Process: Improvement and Practice 7*, 3-4 (2002), pp. 95-112.

[182] R.E.SCHAPIRE, AND Y.SINGER. Improved boosting algorithms using confidence rated predictions. *Journal of Machine Learning Reseach 37*, 5 (1999), pp. 297-336.

[183] RIAZ, M., MENDES, E., AND TEMPERO, E. A systematic review of software maintainability prediction and metrics. In *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM, Orlando* (2009), pp. 367–377, Florida, USA.

[184] RIAZ, M., TEMPERO, E., SULAYMAN, M., AND MENDES, E. Maintainability predictors for relational database-driven software applications: Extended results from a survey. *International Journal of Software Engineering and Knowledge Engineering 23*, 04 (2013), pp. 507-522.

[185] RISING, L., AND JANOFF, N. S. The scrum software development process for small teams. *IEEE software 1*, 4 (2000), pp. 26-32.

[186] ROMBACH, H. D. Design measurement: Some lessons learned. *Software, IEEE 7*, 2 (1990), pp. 17-25.

[187] ROYCE, W. W. Managing the development of large software systems. *proceedings of IEEE Western Electronic Show and Convention, WESCON 26*, 8 (1970), pp. 328-388, LA, USA.

[188] RUMELHART, D., HINTON, G., AND WILLIAMS, R. *Learning Internal Presentation by Back Propagating Errors*. The PDP research Group, Parallel Distributing Processing, Exploration in the Microstructure of cognition, MIT Press, MA, 1994.

[189] RUSTAGI, J. S. *Optimization techniques in statistics*. Elsevier, 2014.

[190] SAED, A., ADIL, A., KADIR, W., AND WAN, M. Applying particle swarm optimization to software performance prediction an introduction to the approach. In *5th Malaysian Conference in Software Engineering (MySEC)* (2011), IEEE, pp. 207-212, Johor Bahru, Malaysia.

[191] SAHRAOUI, H. A., GODIN, R., AND MICELI, T. Can metrics help to bridge the gap between the improvement of oo design quality and its automation? In *Proceedings of International Conference on Software Maintenance (ICSM)* (2000), IEEE, pp. 154-162, San Jose, CA, USA.

[192] SAMOLADAS, I., STAMELOS, I., ANGELIS, L., AND OIKONOMOU, A. Open source software development should strive for even greater code maintainability. *Communications of the ACM 47*, 10 (2004), pp. 83-87.

[193] SANCHEZ, L., AND COUSO, I. Fuzzy random variables-based modeling with ga-p algorithms. *Information, uncertainty and fusion* (2000), pp. 245-256.

[194] SARAIVA, J. A roadmap for software maintainability measurement. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)* (2013), IEEE, pp. 1453–1455 , San Francisco, CA, USA.

[195] SARAIVA, J., SOARES, S., AND CASTOR, F. Towards a catalog of object-oriented software maintainability metrics. In *Proceedings of the 4th International Workshop on Emerging Trends in Software Metrics (WETSoM)* (2013), pp. 84–87, San Francisco, CA, USA.

[196] SCACCHI, W. Understanding the requirements for developing open source software systems. *Software IEE Proceedings 149*, 1 (2002), pp. 24-39.

[197] SCHNEBERGER, S. L. Distributed computing environments: effects on software maintenance difficulty. *Journal of Systems and Software 37*, 2 (1997), pp. 101-116.

[198] SCHNEIDEWIND, N. Software quality control and prediction model for maintenance. *Annals of Software Engineering 9*, 1 (2000), pp. 79-101.

[199] SCHWABER, K., AND SUTHERLAND, J. The scrum guide. *Scrum Alliance* (2011).

[200] SHELDON, F. T., JERATH, K., AND CHUNG, H. Metrics for maintainability of class inheritance hierarchies. *Journal of Software Maintenance and Evolution: Research and Practice 14*, 3 (2002), pp. 147-160.

[201] SINGH, Y., AND GOEL, B. A step towards software preventive maintenance. *ACM SIGSOFT Software Engineering Notes 32*, 4 (2007), pp. 10-21.

[202] SINGH, Y., AND MALHOTRA, R. *Object-Oriented Software Engineering*. PHI Learning Pvt. Ltd., 2012.

[203] SNEED, H. M. A cost model for software maintenance & evolution. In *Proceedings of 20th IEEE International Conference on Software Maintenance (ICSM)* (2004), IEEE, pp. 264-273, Chicago, USA.

[204] SNEED, H. M., AND MERAY, A. Automated software quality assurance. *IEEE Transactions on Software Engineering 11*, 9 (1985), pp. 909-935.

[205] SONI, N., AND KHALIQ, M. Maintainability estimation of object-oriented software: Design phase perspective. *International Journal of Advanced Research in Computer and Communication Engineering 4*, 3 (2015), pp. 52-57.

[206] SOUNDARARAJAN, S., ARTHUR, J. D., AND BALCI, O. A methodology for assessing agile software development methods. In *Proceedings of the Agile Conference (AGILE)* (2012), pp. 51–54, Bengaluru, India.

[207] SPECHT, D. F. A general regression neural network. *IEEE Transactions on Neural Networks 2*, 6 (1991), pp. 568-576.

[208] SPECHT, D. F., AND SHAPIRO, P. D. Generalization accuracy of probabilistic neural networks compared with backpropagation networks. In *Proceedings of the International Joint Conference on Neural Networks, IJCNN* (1991), vol. 1, pp. 887–892, Seattle, USA.

[209] STARK, G. E., KERN, L. C., AND VOWELL, C. A software metric set for program maintenance management. *Journal of Systems and Software 24*, 3 (1994), pp. 239-249.

[210] STAVRINOUDIS, D., XENOS, M., AND CHRISTODOULAKIS, G. D. Relation between software metrics and maintainability. In *Proceedings of the International Conference, Federation of European Software Measurement Associations, FESMA* (1999), pp. 465–476, Amsterdam, The Netherlands.

[211] STONE, M. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological) 36*, 2 (1974), pp. 111-147.

[212] STROGGYLOS, K., AND SPINELLIS, D. Refactoring–does it improve software quality? In *Proceedings of the 5th International Workshop on Software Quality, WoSQ* (May 2007), pp. 1–6, Minnesota, USA.

[213] SUN, P., AND WANG, X. Application of ant colony optimization in preventive software maintenance policy. In *Proceedings of the International Conference on Information Science and Technology (ICIST), Guangdong, China* (2012), pp. 141–144, Guangdong, China.

[214] SVENSSON, H., AND HOST, M. Introducing an agile process in a software maintenance and evolution organization. In *Proceedings of the Ninth European Conference on Software Maintenance and Reengineering, CSMR* (2005), pp. 256–264, Manchester, UK.

[215] SWANSON, E. B. The dimensions of maintenance. In *Proceedings of the 2nd international conference on Software engineering, ICSE* (1976), pp. 492–497, Estoril, Portugal.

[216] TANG, M. H., KAO, M. H., AND CHEN, M. H. An empirical study on object-oriented metrics. In *Proceedings of the Sixth International Symposium on Software Metrics, BocaRaton* (1999), pp. 242–249, Florida, USA.

[217] THONGMAK, M., AND MUENCHAISRI, P. Maintainability metrics for aspect-oriented software. *International Journal of Software Engineering and knowledge Engineering 19*, 03 (2009), pp. 389-420.

[218] THRIFT, P. R. Fuzzy logic synthesis with genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms, ICGA* (1991), pp. 509–513, San Diego, CA, USA.

[219] THWIN, M. M., AND QUAH, T. S. Application of neural networks for software quality prediction using object-oriented metrics. *Journal of systems and software 76*, 2 (2005), pp. 147-156.

[220] UPADHYAY, N., DESHPANDE, B., AND AGARWAL, V. Developing maintainability index of a software component: a digraph and matrix approach. *ACM SIGSOFT Software Engineering Notes 35*, 5 (2010), pp. 1-11.

[221] VELMOUROUGAN, S., DHAVACHELVAN, P., BASKARAN, R., AND RAVIKUMAR, B. Software development life cycle model to improve maintainability of software applications. In *Fourth International Conference on Advances in Computing and Communications* (2014), IEEE, pp. 270–273, Kotchi, India.

[222] VIVANCO, R., AND PIZZI, N. Finding effective software metrics to classify maintainability using a parallel genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation, GECCO* (2004), pp. 1388–1399, Seattle, WA, USA.

[223] WANG, Z., AND XU, J. Predicting protein contact map using evolutionary and physical constraints by integer programming. *Bioinformatics 29*, 13 (2013), pp. 266-273.

[224] WELKER, K. D., OMAN, P. W., AND ATKINSON, G. G. Development and application of an automated source code maintainability index. *Journal of Software Maintenance: Research and Practice 9*, 3 (1997), 127-159.

[225] WEN-HUA, Y. Predicting object-oriented software maintainability using projection pursuit regression. In *The 1st International Conference on Information Science and Engineering (ICISE)* (2009), pp. 3827-3835, Wuhan, China.

[226] WILKING, D., UMAR, F., AND KOWALEWSKI, S. An empirical evaluation of refactoring. *e-Informatica 1*, 1 (2007), pp. 27-42.

[227] XIA, F., AND SRIKANTH, P. A change impact dependency measure for predicting the maintainability of source code. In *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC)* (2004), pp. 258–262, Hong Kong.

[228] XING, Z., AND STROULIA, E. Refactoring practice: How it is and how it should be supported-an eclipse case study. In *Proceedings of the 22nd IEEE International Conference on Software Maintenance, ICSM* (2006), pp. 458–468, Eindhoven, The Netherlands.

[229] YACOUB, S. M., AMMAR, H. H., AND ROBINSON, T. Dynamic metrics for object oriented designs. In *Proceedings of Sixth International Software Metrics Symposium* (1999), IEEE, pp. 50-61, Boca Raton, FL, USA.

[230] YAMASHITA, A., AND MOONEN, L. Do code smells reflect important maintainability aspects? In *28th IEEE International Conference on Software Maintenance (ICSM)* (2012), IEEE, pp. 306–315, Trento, Italy.

[231] YANG, J., AND HONAVAR, V. Feature subset selection using a genetic algorithm. *Feature extraction, construction and selection 453*, 1 (1998), pp. 117-136.

[232] Yau, S. S., and Collofello, J. S. Some stability measures for software maintenance. *IEEE Transactions on Software Engineering 6*, 6 (1980), pp. 545-552.

[233] Yau, S. S., Collofello, J. S., and MacGregor, T. Ripple effect analysis of software maintenance. In *Proceedings of the Second IEEE International conference on Computer Software and Application Conference, COMPSAC* (1978), pp. 60–65, Chicago, USA.

[234] Ye, F., Zhu, X., and Wang, Y. A new software maintainability evaluation model based on multiple classifier combination. In *International conference on Quality, Reliability, Maintenance and Safety Engineering* (2013), pp. 1588–1591, Chengdu,China.

[235] Ying, A. T., Murphy, G. C., Ng, R., and Chucarroll, M. Predicting source code changes by mining change history. *IEEE Transactions on Software Engineering 30*, 9 (2004), pp. 574-586.

[236] Zhang, W., Huang, L. G., Vincent, N., and Jidong, G. Smplearner: learning to predict software maintainability. *Automated Software Engineering 22*, 1 (2015), pp. 111–141.

[237] Zhou, Y., and Leung, H. Predicting object-oriented software maintainability using multivariate adaptive regression splines. *Journal of Systems and Software 80*, 8 (2007), pp. 1349-1361.

[238] Zhou, Y., and Xu, B. Predicting the maintainability of open source software using design metrics. *Wuhan University Journal of Natural Sciences 13*, 1 (2008), pp. 14-20.

SUPERVISOR’S BIOGRAPHY

Ruchika Malhotra

Associate Head and Assistant Professor

Department of Software Engineering

Delhi Technological University Delhi-110042, India

Email: ruchikamalhotra2004@yahoo.com

Educational Qualification:

Post Doctoral (Indiana University-Purdue University Indianapolis, USA), Ph.D. (Information Technology), MCA(SE), BIS(H)

Dr. Ruchika Malhotra is an assistant professor in the Department of Computer Science & Engineering, Delhi Technological University (formerly Delhi College of Engineering), Delhi, India. She is a Raman Scholar and was awarded the prestigious UGC Raman Postdoctoral Fellowship by the government of India, under which she pursued postdoctoral research in the Department of Computer and Information Science, Indiana University - Purdue University Indianapolis, Indiana. She earned her master's and doctorate degrees in software engineering from the University School of Information Technology, Guru Gobind Singh Indraprastha University, Delhi, India. She was an assistant professor at the University School of Information Technology, Guru Gobind Singh Indraprastha University, Delhi, India. Dr. Malhotra received the prestigious IBM Faculty Award in 2013 and has received the Best Presenter Award in Workshop on Search Based Software Testing, ICSE, 2014, Hyderabad, India. She is an executive editor of Software Engineering: An International Journal and is a coauthor of the book, Object-Oriented Software Engineering. Dr. Malhotra's research interests are in empirical research in software engineering, improving software quality, statistical and adaptive prediction models, software metrics, the definition and validation of software metrics, and software testing. Her H-index as reported by Google Scholar is 18. She has published more than 120 research papers in international journals and conferences, and has been a referee for various journals of international repute in the areas of software engineering and allied fields. She is guiding several doctoral candidates and has guided several undergraduate projects and graduate dissertations. She has visited foreign universities such as Imperial College, London, UK; Indiana University - Purdue University Indianapolis, Indiana; Ball State University, Muncie, Indiana; and Harare Institute of Technology, Zimbabwe. She has served on the technical committees of several international conferences in the area of software engineering (SEED, WCI, ISCON).

AUTHOR'S BIOGRAPHY

Anuradha Chug

Assistant Professor

University School of Information and Communication Technology

Guru Gobind Singh Indraprasth University

Dwarka, New Delhi-110077, India

Email: anuradha@ipu.ac.in, a_chug@yahoo.co.in

Educational Qualification:

Bachelor of Computer Science(Banasthali Vidyapith), Master of Computer Science (Banasthali Vidyapith), M.Tech Information Technology (GGSIP University, Dwarka)

Anuradha Chug has long teaching experience of almost 20 years to her credit as faculty and in administration at various educational institutions in India. She has worked as guest faculty in Netaji Subhash Institute of Information and Technology, Dwarka, New Delhi and Regular Faculty at Government Engineering College, Bikaner. Before picking the current assignment as Assistant Professor at USICT, GGSIP University, she has also worked as Academic Head, Aptech, Meerut and Program Coordinator at Regional Centre, Indira Gandhi National Open University (IGNOU), Meerut.

In academics, she has achieved top rank in her M.Tech (IT) degree and conferred the University Gold Medal in 2006 from Guru Gobind Singh Indraprastha University. Previously she has acquired her Master's degree in Computer Science from Banasthali Vidyapith, Rajasthan in the year 1993. Her H-index as reported by Google Scholar is 4. She has published more than 20 research papers in international journals and conferences.