Design and Analysis of Some Software Metrics Using Soft Computing Approaches

A Thesis

Submitted in fulfillment of the
requirements for the award of the degree of

Doctor of Philosophy

Submitted by

Vijai Kumar
(Registration No. 950911009)

Under the supervision of

Dr. Rajesh Kumar
Associate Professor and Head,
School of Mathematics and Computer Applications, Thapar University,
Patiala-147004, India

Dr. Arun Sharma
Professor and Head,
Department of Computer Science and Engineering,Krishna Institute of Engineering and Technology, Ghaziabad-201206, India

School of Mathematics and Computer Applications
Thapar University,
Patiala–147004 (Punjab), India.

August 2014

Certificate

I hereby certify that the work which is being presented in this thesis entitled **Design and Analysis of Some Software Metrics using Soft Computing Approaches**, in fulfillment of the requirements for the award of degree of **Doctor of Philosophy** submitted in School of Mathematics and Computer Applications (SMCA), Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Dr. Rajesh Kumar and Dr. Arun Sharma, and refers the work of other researchers, which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

Vijaikumar

(Vijai Kumar)
Registration No. 950911009

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge and belief.

27/08/2013

(Rajesh Kumar)
Associate Professor and Head,
School of Mathematics and Computer Applications,
Thapar University,
Patiala–147 004 (INDIA)

Supervisor

(Arun Sharma)
Professor,
Department of Computer Science and Engineering, Krishna Institute of Engineering and Technology, Ghaziabad-201206 (INDIA)

Supervisor

Abstract

1. Introduction

Software metrics are the integral part of software engineering to quantify activities during the different phases of software development life cycle. The quantification processes consist of the measurements of entities which need to be analyzed during the software development processes. In software engineering, these entities are called software metrics, which are measured in all the phases of software development life cycle. There are defined processes and product metrics which contain various attributes to be measured. Direct measurement is not possible for some software metrics; hence these metrics are derived using a combination of different metrics attributes. For example, size metric is a direct measure and complexity is indirect measure, which depends on a combination of different factors.

Software metrics measurement has been a great research interest area for last four decades. The different conventional statistical methods have been proposed for analysis and measurements of the software attributes. Regression analysis is frequently used method for software attributes analysis. Although conventional methodologies are most widely used in the industries worldwide but recently the research communities are moving towards interdisciplinary streams and have started exploring the unconventional techniques such as fuzzy logic, artificial neural network, neuro-fuzzy etc. There are some metrics attributes, which needs forecasting i.e., the occurrence in future for better process improvement in advance. Some of these metrics are software quality, reliability, maintainability, reusability, defect density and many more. There is no direct measurement possible with the fixed mathematical concept for most of these attributes. So, the term "prediction" comes into consideration in software engineering. The prediction is nothing but the forecasting and quantification of these attributes which are based on the available information during the software development process at any point of time.

The regression analysis has been used from early days for prediction of software metrics. Recently soft computing paradigms attracted the research community and foreseeing a huge potential of research using soft computing techniques in engineering

disciplines. Soft computing has been used extensively in engineering and science but a great momentum has been seen in the area of software engineering during the last decade. The reason of this momentum is the predictive characteristics of soft computing approaches. There are good numbers of evidences where soft computing outperforms the conventional techniques in prediction of an attribute or future event quantification.

In the proposed study, soft computing techniques have been used to analyze software quality, maintainability, reliability, reusability, defects of software products and systems. The fuzzy logic, artificial neural network and adaptive neuro fuzzy inference system have been used in proposed study. The different soft computing techniques have different characteristic and hence some pros and cons. In literature survey, this has been observed that the proposed conventional techniques works efficiently in specific domain and in case of some variations, the conventional techniques do not perform upto expectation. The soft computing techniques tried to solve this problem and able to adapt the environment changes. The fuzzy logic is proven tool for decision making problem analysis and artificial neural networks become popular because of its learning and adaptive capability. Neuro-fuzzy combines and inherits the advantage of both.

In the proposed research work, we have used the soft computing techniques to predict the software attributes as per the applicability and importance. The fuzzy toolbox, artificial neural network simulator and adaptive neuro-fuzzy inference system of MATLAB™ software is used for experimentation of the proposed soft computing approaches. Although there are plenty of research contents of the prediction of above mentioned software metrics but there is a scope of improvements of the existing approaches in terms of feasibility, efficiency, accuracy, generic framework for all domains, applicability, practical ability etc. All of them may not be feasible to include in a single approach but there can be a balance of these so that industries can adapt in their software engineering processes during the software development life cycle. Most of the soft computing techniques need some history data either to design the model or validate the model, fuzzy logic can be built with less data or even without any data. The research in software engineering using soft computing approaches is still in progress because a specific model or metric works well either in specific domain type project or with particular data set.

Here, the conclusion throws out the fact that there is always a need of designing the soft computing based software metrics and model due to these limitations and practicality.

2. Objectives of the Proposed Work

In the objective, focus is to analyze the existing soft computing based software metrics and propose some soft computing based approaches to address the prediction requirements of software defects, reliability, quality, reusability, maintenance severity and maintainability as well as validate them with real project data set to show quantitative analysis. The followings are the set of objectives which are analyzed and explored in this study:

Objective 1: *Study and analysis of the existing metrics and approaches.*

The existing software metrics and approaches have been analyzed and explored. The conventional, nonconventional approaches and models have been thoroughly studied for software metrics such as quality, defects, reliability, reusability and maintainability. Based on analysis, the future prospects of soft computing techniques have been discussed.

Objective 2*: To design some software metrics such as maintainability, defect density, reusability, reliability for component-based systems and other domains.*

The soft computing based software metrics and approaches have been proposed for prediction of software defects, reusability, maintenance severity, maintainability and reliability. The proposed approaches are defined for the appropriate software engineering paradigms as per importance and need.

Objective 3: *Evaluation of software metrics by using soft computing approaches such as fuzzy logic, artificial neural network and neuro-fuzzy.*

All the proposed metrics are simulated using soft computing approaches such as fuzzy logic, artificial neural network and neuro-fuzzy. The evaluation is done for these proposed metrics approach using root mean square error and error performance. The proposed defect and reliability approaches are evaluated for the generic software engineering paradigms. The reusability metrics are evaluated for component based systems. The maintenance severity is also evaluated independent of used methodology in software development. The fuzzy toolbox, artificial neural network simulator and adaptive

neuro fuzzy inference system of MATLAB™ software is used for experimentation of the proposed approaches.

Objective 4: *Validation of proposed metrics.*

The empirical validation of the defect and reliability metrics carried out using the real project data from two industrial projects which are based on mixed software engineering methodologies i.e., structural, object and component oriented for a complex telecommunication software system. The proposed reusability software metrics are empirically validated against the component based system project data collected from live project. For reusability, a comparative analysis and validation is done between different soft computing approaches as per applicability. The maintenance severity metric is validated with three project data sets of structural, component and object oriented development.

3. Thesis Outline

Thesis is structured into seven chapters as per following scheme of chapters:

Chapter 1 covers the basic understanding and introduction about the software engineering and soft computing basics. It includes some basic concepts for the need and importance of prediction of software attributes as well as the usefulness of soft computing techniques. It covers brief definitions of software engineering methodologies and the metrics proposed in the current study. It also contains the fundamental concepts of soft computing approaches i.e., fuzzy logic, artificial neural network and neuro-fuzzy techniques. This has been concluded that there is a need of prediction of software metrics, where soft computing can play an important role.

Chapter 2 contains the analysis and review of the existing conventional and nonconventional approaches. The initial section gives the introduction about the basic software metrics and need of soft computing to solve the prediction problems in software metrics. The detailed analysis of conventional metrics is carried out considering the maintainability, defect, reliability, quality, and reusability metrics. Also, the detailed analysis of soft computing approaches such as fuzzy logic technique, artificial neural network technique, neuro-fuzzy techniques and other hybrid methodologies have been carried out for the set of defined metrics. The chapter concludes with the future prospects

and possibilities of soft computing intervention in prediction of software metrics and to propose the soft computing based software metrics and model.

Chapter 3 discusses the defect metric using soft computing approach. Existing work in literature regarding defects prediction has been explored in this chapter. As software defect cannot be measured directly, it needs the prediction using some technique. The defect density metric is designed using the combination of software complexity, size and number of defects observed before customer delivery. Here, the defect density metrics is designed considering any type of software engineering methodology but we need to capture the input factor values such as lines of count, complexity and defects count before release. These metrics can be designed considering the history data from any project, which can be applied to other similar projects for defect prediction. Here, we have proposed fuzzy logic and artificial neural network based defect density metric with the three independent factors. The validation is done using the data set of two different projects P_1 and P_2 from telecom domain, and the better accuracy has been observed through results. Project P_1 is an optical telecom application project, which is used as an optical communication platform across the cities. It has been implemented using object oriented based system design as well as structural design and developed in 4 years timeframe. Project P_2 is a 4G telecom application project which is mix of all three main software engineering methodologies conventional, object oriented and component based application project. The validation has been performed for proposed approach across various domain projects.

Chapter 4 proposes a unique and new quality and reliability management framework across the subsequent releases of software product. In starting sections, it gives the detailed analysis of the proposed techniques for reliability and quality relationship with defects. The basic reliability growth model is used to design the reliability and quality prediction framework. The proposed reliability and quality model is implemented using fuzzy logic and artificial neural network. The developed approach is validated using three release data from two different projects. Multiple releases of project P_1 from optical telecom project are used for validation. Three release of Project P_2 from 4G telecom software domain were used for cross validation across different project. The validation and experimentation is done with the different combination of fuzzy logic

and artificial neural network architectural attributes. The reliability factor is calculated on the basis of predicted defects for multiple releases and the model is able to predict the defects and reliability with a good accuracy. Using the model across releases, the quality and reliability management framework is proposed which is a new concept across the subsequent releases for large and complex projects. The discussion has been organized for the practical importance and application of the developed framework in the software quality assurance and software process improvements.

Chapter 5 contains the proposed fuzzy logic, artificial neural network, neuro-fuzzy based approach for prediction of component reusability. Six independent factors, which influence most to component reusability, are identified for the software components. These factors are release version, existing defects, portability, interface complexity, customizability and understandability. Several real-life components are used for the training and testing of soft computing techniques. The data for several components is collected from web sources and in-house development. These components include very simple calculator application to complex inventory management system. These components have been developed using different technologies ranging from Java beans, .Net to open source technologies. The validation is done using the components' data and the quantified results show that adaptive fuzzy inference system performs better than other two soft computing approaches in terms of root mean square error.

Chapter 6 presents the exhaustive analysis of artificial neural network approach to design the software maintenance severity and maintainability metrics independent of component, procedural and object oriented based development. This chapter discusses soft computing based approach for maintenance severity prediction and maintainability prediction of software system modules. In the proposed maintenance severity metrics, artificial neural network approach is used. Six influencing basic metrics for maintenance severity are used as independent factors which are easily available and captured during the software development process. These factors are halstead difficulty, multiple condition count, decision count, cyclomatic complexity, design complexity and lines of count for component based, structural or object oriented software development strategy. In case of maintainability prediction, four basic metrics are considered to formulize the maintainability prediction approach. These four simple metrics are: multiple condition

count, node count, percentage comments and lines code which can be easily collected by analyzing the code. The different possible combination of artificial neural network architectures and two learning algorithms are used to get the better results. For the variance in architecture and algorithm, different numbers of neurons nodes ranging from 5 to 25 are selected for the analysis. The artificial neural network approach needs a good amount of input and output data vector sets. In this case, artificial neural network is used, which is based on three projects data set to train and test the proposed metrics. The data of three projects PC5, PC4, PC2 from PROMISE repository of empirical software engineering data is used for validation and training.

The conclusion of the thesis has been presented in Chapter 7, which is summarized form of major contribution of presented work. The direction for future work has also been detailed in this chapter.

Acknowledgement

The work leading to this doctoral thesis has been carried out by me as a research scholar at the School of Mathematics and Computer Applications (SMCA) of Thapar University, Patiala. This duration has been very stimulating and instructive for me.

At the outset, I am highly indebted to my revered supervisors, Dr. Rajesh Kumar and Dr. Arun Sharma for their untiring support, encouragement and able guidance at each and every step throughout this research endeavor. I admire the knowledge of both of my supervisors. It is really fortunate that I got such an opportunity to learn the ABC of research from them, which will definitely go a long way in my professional career. Further, I find no words to express my heartfelt thanks to supervisors for guiding this work so diligently and delightfully. It has indeed been a privilege to carry out this research work under their guidance.

I am grateful to the Director, Thapar University, Patiala for providing me university's resources and facilities necessary to carry out this research work. I express my gratitude to the Doctoral Committee comprising Professor (Dr.) R. K. Sharma and Dr. Deepak Garg for monitoring the progress and providing valuable suggestions for improvements of my Ph.D. research work from time to time. I am also thankful to the entire faculty and staff of SMCA for healthy discussions and suggestions during the progress report.

I am thankful to my colleagues from different organizations for supporting me during the study time frame. Also, Special thanks are due my brother Dr. Arun Kumar for his valuable suggestions on research paper publications and methodologies.

My hearty thanks to all the anonymous referees of several international research journals in the domain of software engineering, quality, neural and fuzzy computing for their constructive comments along with new ideas. I wish to thank many acclaimed researchers and reviewers of our publications across the world for valuable ideas and suggestions. Thanks to all researchers and paper authors, from where we got reference for our research and analysis of their proposed methodologies.

Finally I would like to dedicate this thesis to my family. I thank my parents, who gave me the courage to get my education, supported me in all achievements throughout

my life. I am grateful to my other family members for their continuous encouragement and motivation. I must acknowledge the constant cooperation, hard work and moral support of my loving wife, Mrs. Jyoti Sharma along with our lovely children Vedant Gaur and Vedika Gaur throughout this research work, without which it could have not been possible to achieve this target so contentedly. Above all, I pay my reverence to the almighty God.

(Vijai Kumar)

Place:

Date:

List of Publications by the Author

Papers in International Journals (Published)

1. **Kumar, V**., Sharma, A., Kumar, R., Grover, P. S., 2012. Quality Aspects for Component-based Systems: A Metrics Based Approach, **Software: Practices and Experience, John Wiley & Sons,** December, Vol. 42, Issue 12, pp: 1531-1548. ***(SCI Indexed)***

2. **Kumar, V**., Sharma, A., Kumar, R., 2013. Applying Soft Computing Approaches to Predict Defect Density in Software Product Releases: An Empirical Study. **COMPUTING AND INFORMATICS, North America,** Vol. 32, Issue. 1, pp: 203-224. ***(SCI Indexed)***

3. **Kumar, V**., Kumar, R., Sharma, A., 2013. Applying Neuro-fuzzy Approach to Build the Reusability Assessment Framework Across Software Component Releases - An Empirical Evaluation, **International Journal of Computer Applications (IJCA),** May, Vol. 70, Issue 15, pp: 41-47.

4. **Kumar, V**., Kumar, R. and Sharma, A., 2014. Maintainability Prediction from Project Metrics Data Analysis Using Artificial Neural Network: An Interdisciplinary Study**, Middle-East Journal of Scientific Research,** Vol. 19, Issue 10, pp: 1412-1420. *(Scopus Indexed)*

List of Figures

List of Tables

Abbreviations

ANFIS	Adaptive Neuro Fuzzy Systems
ANN	Artificial Neural Network
API	Application Program Interface
BPNN	Back-Propagation Neural Network
CBD	Component-Based Development
CBS	Component-Based System
CBSD	Component-Based Software Development
CBSE	Component-Based Software Engineering
COCOMO	Constructive Cost Model
CR	Component Reusability
CRL	Component Reusability Level
COTS	Commercial of the Shelf
DD	Defect Density
DIT	Depth of Inheritance Tree
FBBPNN	Feed Forward Back-Propagation Neural Network
FI	Failure Intensity
FIS	Fuzzy Inference System
GA	Genetic Algorithm
GRNN	General Regression Neural Network
IBM	International Business Machines
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization

KLOC	Thousands of Lines of Code
LOC	Lines of Code
MF	Membership Function
MIS	Management Information System
MRE	Mean Relative Error
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
NASA	National Aeronautics and Space Administration
NATO	North Atlantic Treaty Organization
OO	Object Oriented
OOP	Object Oriented Program
PCA	Principal Component Analysis
PREDD	Pre Release Defects
RBF	Radial Basis Function
REBOOT	Reuse Based on Object-Oriented Techniques
RMSE	Root Mean Square Error
ROR	Ruby On Rails
SC	Soft Computing
SDLC	Software Development Cycle
SPI	Software Process Improvements
SQ	Software Quality
SQA	Software Quality Assurance
TLOC	Total Lines of Code

TMF	Triangular Membership function
UML	Unified Modeling Language

Contents

Chapter 1

Software Metrics and Soft Computing - An Introduction

1.1 Introduction

IEEE Standard 610.12 (IEEE, 1993) has defined the software engineering as an application of organized, closely controlled, quantitative methodology for the design, development, maintenance and operation of a software system. In simple words, software engineering is a methodology which includes and integrates the methods, processes and tools for software development work. The different process models in software engineering field have been developed to control software development process. Each model has own weakness and strength but also has some common phases in the modeling of the software process.

Software engineering term was first used in North Atlantic Treaty Organization (NATO) software engineering conference in 1968. The software engineering discipline was formed and comes into realization in industries during the period of late 1960s. Before that software engineering processes were not formally part of the software

development processes and there was not a good control on the processes during the software development. Over the period of time, the researchers have analyzed different attributes and included these in the process of software development, which make the process easier to manage. Software metrics are defined to measure the various software attributes during the software development. There has been a lot of contribution from research world for measurement of software attributes which forms the metrics. After continuous research, various software metrics and methods have been proposed which makes the software development process more controlled to manage and also improve the quality and reliability of the developed software.

1.1.1 Software Engineering Development Approaches

Software engineering development approaches are mainly divided into three categories: structural or conventional software engineering, object oriented software engineering and Component-based Software Engineering (CBSE). Later, other software engineering paradigms; aspect oriented, agent oriented, etc. have also been proposed but the three main methodologies are frequently used approaches in industries.

1.1.1.1 Structural Software Engineering Approach

Conventional or structure based software engineering includes a hierarchical set of individual modules. These modules are in abstract form at top level and at lower level, these are detailed out. The complete system is divided into the modules which logically consist of different set of functionality. In structural approach, the coupling and cohesion metrics were defined by Larry Constantine in 1968, who was original developer of structural concept. Later, these concepts are used in other software engineering approaches as well. It is observed that for better design there should be less coupling and high cohesion, which makes the system maintainable and easy to enhance. Low coupling is considered to be a factor for well structured design. If a structured system is having low coupling and high cohesion, the structural design will have high maintainability and readability.

1.1.1.2 Object Oriented Software Engineering Approach

Earlier, organizations were following structure based software development approach and it was victorious for small and less complex applications. Then Object Oriented (OO) approach was introduced in 1960s, it is mainly based on polymorphism, data encapsulation and inheritance concepts. Inheritance is the property of object oriented approach in which attributes and operational profiles of subclass and object are inherited from a class. Polymorphism is the concept where different operations can have the same name and existence in system. Later in 1990s, object oriented software engineering becomes more popular among the software industries. Although, different concepts have been used in object oriented approaches but the fundamental activities of the programming still remain same as conventional programming. Industries have adopted the object oriented approach at a very fast pace (Booch, 1994) considering the benefits of encapsulation, modularity, polymorphism and inheritance (Jacobson *et al*., 1992; Rumbaugh *et al*., 1991). However this approach has some drawbacks also. Object oriented approach can support information hiding to class level only. Also, confidentiality and integrity are known problems in object oriented approach.

1.1.1.3 Component-Based Software Engineering Approach

Component-based software engineering enforces reusability approach to design and implement less coupled components which are independent in the system. In other words, component-based software engineering is a methodology where the software product is built from distinguished and independent boxes of functionality, theses boxes are called components. According to Sparling (2000), the component is language independent package offering unique software services and encapsulated container which has clearly defined and published interface to access it. Software industries have some common domain, functionality and trends across the software projects development. The common domain projects must have a good percentage of common and unconnected functionalities to build. Hence, instead of developing again and again, the common and identical functionality can be built into a single component, which should provide a defined interface to access it from application layer. This fundamental requirement results to the formulation of Component-based System Development (CBSD). The reusability is

the main characteristic for CBSD and plays an important role in system quality and reliability.

1.2 Software Metrics

When you are not able to measure an entity, you cannot control and quantify the process which uses this entity. Considering this necessity, metrics have been introduced in engineering and science fields. Similar to other engineering disciplines, software engineering also has metrics. In general term, metrics are used for measurement of a software engineering attribute. Software metrics are used for quantitative assessment of the software design and development process. Software metrics provide the current status of an attribute of the software process and give some insight information to analyze it.

The objective of software metrics is dual. First, the metrics are used to define the development schedule and make necessary adjustments during the development which helps to mitigate the identified risk. Second, the project metrics are combined to access the product quality and using these metrics, wherever needed, changes can be incorporated in the approach being used for software quality improvement process. It is difficult to give the details of all the metrics used in software development but most of them are interconnected to achieve the desired goal of better software processes. For example, defect metrics are calculated and quality factors are derived from defect rate, then actions are taken to improve quality. In turn, this metrics assessment may lead to rework the schedule and overall project cost. Some of the important direct measure of the software process model includes faults count, code size, used memory size, execution speed etc. The indirect or derived measures are quality, maintainability, complexity, reliability and many more. The metrics are used in entire software development life cycle to define the schedule estimation, efforts estimation, quality assessment and control of the project as project proceeds.

The metrics are included in the process to isolate the areas of the problem during the development so that the early steps can be taken if something is wrong or expected to be wrong, based on the metrics values. If, we do not have a quantitative analysis based on the metrics, the decision can be a biased evaluation. Using the measurement, ongoing software process trends can be visualized; it can be good or a bad trend. Taking the trends

inputs, the process improvements can be defined in Software Development Life Cycle (SDLC). Software metrics define the project and process indicators. Project indicators facilitate process manager to analyze, assess the condition of the current and future state of the project and also risk analysis. The process and project indicators help to discover the problem areas before it create any crisis and adjust the process accordingly to improve the quality and planning. There are various software metrics which can be used for project and process indicators. In simple software process model, process and product measures are collected and normalized using some standard scale. In calculation of the measurements, the collected metrics and combination of these metrics are used to access the current state and trends. The defined metrics and collected data sets can be used as standard for the similar type of projects within the organization.

Park *et al.* (1996) defines the four major reasons to use the metrics; to distinguish, to forecast, to enhance and to assess. The main intent of the metrics is to provide some measurements which can lead to software process improvements. There are various methods proposed about the application of software metrics in software development and design processes which required extensive study of the metrics. Barry Boehm provides the important work to predict the resource requirements for small to large projects size which is called as Constructive Cost Model (Boehm, 1981). As per Goodman (1993), including the metrics at each step in the software process and product development provides significant information timely to the management so that they can use it to improve the processes and product quality. Sedigh-Ali *et al.* (2001) define the importance of the metrics as usages across the SDLC for guiding the quantitative decision and access the quality assurance process for the financial feasibility.

Broadly, software metrics can be categorized into product and process metrics. Product metrics are gathered and analyzed at any point of software development process from the requirement specification to the field operation. Product metrics includes complexity, size, documentation size etc. Process metrics takes care of software development process attribute's analysis and estimation. It includes development schedule, cost, staff experience etc. Software metrics are planned to compute quantitatively the software quality attributes, planning, scheduling, effort estimation resource distribution, product evaluation. Three categories are defined for software

metrics, project metrics; process metrics and product metrics (Pressman, 2001). Although out of these categories, several metrics have been proposed by research community. The following metrics have been used and proposed in the current research work.

- Software defects and defect density
- Software quality
- Software reliability
- Software reusability
- Software maintainability
- Maintenance severity

1.2.1 Software Metrics and Prediction

The prediction of entities is crucial in software metrics analysis and design for software development planning (Furulund and Molokken, 2007). The prediction is defined as an estimation of a particular entity (Jorgensen and Shepperd, 2007). Voas (2000) defines the predictability as an evaluation and forecasting of a future event. Some of the important metrics which needs prediction in software process are defects, quality, reliability, reusability, maintainability etc. There are various prediction methods proposed to predict the software metrics attributes. Software defects prediction is an important activity as it provides a good indicator to predict the software quality, reliability, reusability, maintainability etc. Various conventional and unconventional techniques have been used for metrics prediction. In unconventional techniques, every model is based on the independent input factors, based on that it predicts the derived dependent output. Basic conventional techniques are correlation methods, accuracy, precision, classification, ranking and other statistical regression models. In last two decades, the researchers have used unconventional techniques such as fuzzy logic, artificial neural network, adaptive fuzzy inference system, genetic algorithm, probabilistic reasoning etc.

1.2.2 Software Quality

The correct schedule, cost and good level of user satisfaction threshold are the eventual requirement of a good quality software development process. Software system is considered with good quality if it behaves as per defined requirements and its availability

without any failure. It is experienced that the software organizations invest 80% time of the resources to improve and achieve the software quality to a defined level. Although the software quality is always important for all type of development processes but in component based development, the developers have to trust on component developers. Hence the quality of components will impact over system quality which is being developed using these components. As shown in Fig. 1.1, McCall *et al.* (1977) describes the basic quality metrics such as *correctness, reliability, efficiency, integrity, maintainability, flexibility, testability, portability, reusability* and *interoperability.*

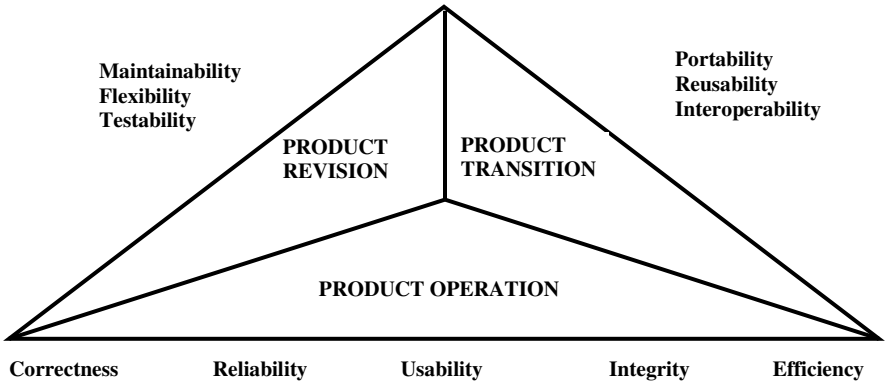


Fig. 1.1: McCall's Software Quality Metrics

1.2.3 Software Defects and Defect Density

Defects are the software errors which can occur after a specified period of time during the system testing or field operation. The defects can appear before or after customer release. Defect density metric is simply the ratio of the total defect to the entity size as following:

$$Defect\ Density = \frac{Total\ Number\ of\ Defects}{Software\ Size} \qquad (1.1)$$

According to IEEE standard 982.2 (1988) and Pfleeger (1992), defect is an abnormality in any intermediate or final software product resulting from an error or fault. It can be an erroneously test data set or incorrect data entry in user documentation. Defect

may cause failure, which is an incapability to perform the desired function of the system software, or failed to act upon functionality in-between the specified limits as defined during the requirements. In following chapters, defects and faults will used interchangeably. Defects or faults are the software errors which can occur after a specified period of time during the system testing or field operation.

In some situations, a defect may not be uncovered for a long period of time. When a particular code or logic of software is executed where the defect exists, it may be uncovered during the execution leg of that particular software module depending upon the special condition, it can cause single or multiple failure at any point of time (Kan, 2003). Hence, the prediction of defects is an important process to access and enhance the software reliability and quality.

1.2.4 Software Reliability

Software reliability engineering is a collection of techniques for the development and enhancement of software system for which the reliability can be quantified and evaluated. Software reliability is not a direct measure, it has to be derived and predicted using a combination of metrics. In simple terms, reliability is measured to quantify that how much the system is close to fulfill the defined requirements. Reliability is related to faults and failure. A failure is an unexpected behavior of software during the operation at any point of time. The unexpected behavior can occur due to an anomaly or bug in software which may not be uncovered during the system testing or the earlier field operation. Such anomalies or bugs are called faults or defect in the system which have been there in the system but passive before they results to failure. There is no straightforward definition about the aspects which are related to reliability. During more than 40 years, a good amount of research work has been carried out to estimate the reliability of software systems. Different metrics have been proposed with combination of faults, time and failure. The following basic metrics are used for reliability:

- *Mean Time To Failure* (MTTF): It is defined as the probable duration between the two failures in sequence.
- *Failure Intensity* (Fl): It is the count of failures which are expected per unit time.
- *Defect Density* (DD): It is measured as number of faults per thousand lines of code.

- *Test Coverage Measures*: It is the measure that how much portion of the software has been executed and tested by the given test suite. There are two types of coverage; statement coverage and branch coverage

1.2.5 Software Reuse and Reusability

Software reuse is being used in software development process to reduce the cost and time of the organization. It is a crucial need to adopt software reuse as the complexity and the size of software are increasing day by day during the software development. As per Basili and Rombach (1988), software reuse is not only reuse of software modules or components but use of everything i.e. processes, methodologies, knowledge etc. Software reuse is a systemic approach where any organization establishes the existing defined operating procedures to indicate, generate, quantify and adjust the facts to use in the software development activities (Mili et al., 1995). Reusability is a measurement which defines that how much a component can be reused to reduce the schedule and cost by taking the existing code and interfaces. Majorly, the integration work will be required to combine these components in software product development (Wang, 2002).

1.2.6 Maintainability

Software maintenance is a set of defined activities which need to be performed as per system's operational and business requirements. Generally, the environment or platform changes over the period of time after the field deployment of system. So the system maintenance is started as soon as the first release of the developed software is delivered to customer or deployed in the field. There will be either bug fixing or some new requirements which need the maintenance and enhancement of the system. Also if the system is operational for long time but it may need to enhancement or maintenance to cater the current new requirements. For example change of the underlying platform from Windows to Linux OS involves the existing system maintenance and porting over the new platform.

Software maintainability is the ease to which degree software can be changed or enhanced (IEEE, 1993). The maintainability is the degree of measurement which defines that how much easy to evolve or maintain the software module or component. For the

measurement, we need to identify the set of factors which defines the degree of easiness and comfortability to enhance the software. In case of component based systems, the component has no meaning if it does not have good maintainability. The main aim of CBSD is to reuse the existing components in the software development process to reduce the effort and time, so maintainability has been considered an important factor for quality in CBSD. Also, better maintainability defines the life of any component which can be reused for a long time.. The current trend is high percentage of maintenance work in software industries which indicates the importance of maintainable software. It is very difficult to measure the maintainability directly; however, it is generally measured combining various factors on which it depends.

1.3 Introduction to Soft Computing Approaches

Soft computing is different from traditional computing because soft computing is used in engineering to solve the uncertainty of problems for which there is no easy and direct measurement possible. It is defined as tolerant of vagueness, approximation, partial truth and uncertainty. The fundamental idea of soft computing is inherited from human brain and its functionality. By implementing the concept of human brain functionality, the aim of soft computing is to provide the solutions with more robustness, understandability and low cost. There are three main constitutes of soft computing; fuzzy logic, artificial neural network and neuro-fuzzy. Zadeh (1965) introduced fuzzy logic theory first time to solve the complex and decision making problems. Neural network and other soft computing techniques are introduced at later point of time. Other soft computing techniques are genetic algorithm, ant colony optimization, swarm optimization, bayesian network, chaos theory and perception. In early 1990s, soft computing has been included as a formal part of computer science (Zadeh, 1994).

1.3.1 Artificial Neural Network

Artificial Neural Network (ANN) is inspired from the biological nervous system. ANN is characterized by its architecture, its learning algorithm, and its activation functions. A set of input and output data is used to train the network. As shown in figure 1.2, ANN is a group of interconnected artificial neurons which formulates the

information processing based on the defined mathematical function. ANN has adaptive nature, it adjust its internal structure based on the internal and external information processing flows. In mathematical modeling of artificial neuron the synapses effects are represented using weights of connection. The weights can be adjusted and work as modulator for input signal associated with it. The transfer function is used to formulate the neuron's nonlinear feature. As shown in figure 1.3, the neuron's output is calculated by taking sum of weighted signals. The learning characteristic of neuron is simulated by changing the weights according to the selected algorithm for learning. The applications of ANN can be used for following type of problem but not limited to them:

- Prediction of an entity based on input values in a trained ANN.
- Classification of input values
- Data association and analysis, it can find the particular data which contain errors.
- Input data analysis for establishing the group relationship

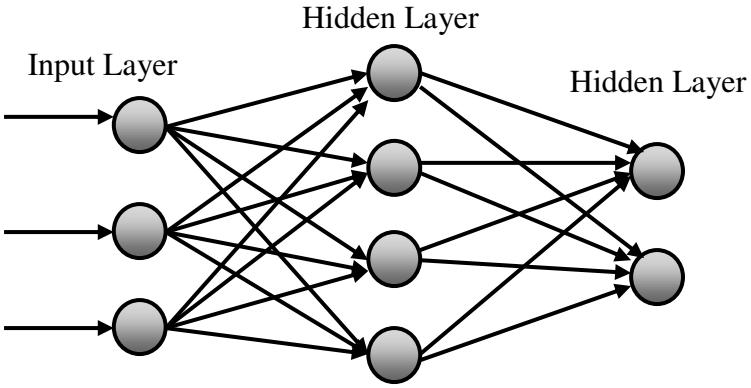


Fig. 1.2: Architecture of a Multilayered Artificial Neural Network

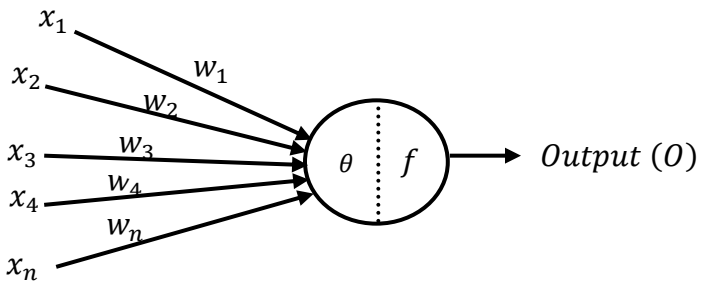


Fig. 1.3: Architecture of an Artificial Neuron

In ANN, the learning algorithm can be supervised and unsupervised. In supervised algorithm, an input vector is given with the defined set of output or response. In turn, the discrepancies in expected and actual responses for each node are discovered. Then the errors are used to find the weight changes in network as per learning rule. The supervised algorithm is based on back propagation algorithm, perception rule and delta rule. In unsupervised algorithm, the training of output unit is done to respond the clustered output pattern. Here, the system is expected to find the specific feature of inputs statistically, the system in case of supervised algorithm needs to develop own input pattern representation.

In ANN architecture, there are different learning algorithms and its activation functions. The learning capability can be designed as adjustment in weight connection values that infer to collection of information which can be later recalled, simulating the human brain behavior. Before training of the ANN, the default weights are set to some random values but within the specified range. Two types of neural network learning are proposed, supervised and unsupervised.

Supervised learning is frequently used learning algorithm in neural networks. It requires many samples to serve as exemplars. Each sample of this training set contains input values with corresponding desired output values (or target values). Then the network will attempt to compute the desired output from the set of given inputs of each sample by minimizing the error of the model output to the desired output. It attempts to do this by continuously adjusting the weights of its connection through an iterative learning process called training (Beizer, 1990; Mayrhauser *et al*., 1995). The supervised learning approach is being used in the present study.

Unsupervised learning is sometimes called self-supervised learning and requires no explicit output values for training. Each of the sample inputs to the network is assumed to belong to a distinct class. Thus, the process of training consists of allowing the network to uncover these classes.

There are two main advantages when using estimation by artificial neural networks. First, it allows the learning from previous situations and outcomes. Second, it can model a complex set of relationships between the dependent variable and the independent variables. However, there are some shortcomings of ANN approach also.

Neural networks approach may be considered as 'black-box'. Consequently, it is not easy to understand and to explain its process to the users. Moreover, there is no guidelines for the construction of the neural network topologies, number of layers, number of units per layer, initial weights etc.

1.3.1.1 Neural Network Architecture

Neural network architecture (or network topology) refers to the types of interconnections between neurons. A network is said to be fully connected if the output from a neuron is connected to every other neuron in the next layer. A network with connections that passes outputs in a single direction only to neurons on the next layer is called a feed forward network. A feedback network allows its outputs to be inputs to preceding layers. Networks that work with closed loops are known as recurrent networks. Feed forward networks are faster than feedback networks as they require only a single pass only to obtain the solution. The former networks are commonly used for prediction problems (Mukherjee and Deshpande, 1995). An elementary neuron with inputs is weighted with an adjustment. The sum of the weighted inputs and the bias forms the input to the transfer function. Neurons may use any differentiable transfer function to generate their output.

1.3.1.2 Learning Algorithms

A prescribed set of well-defined rules for the solution of a learning problem is called learning algorithm. Several learning algorithms exist for neural networks learning. In this study, supervised algorithm of Feed forward back-propagation neural networks (FFBPNNs) is used. Back propagation is the generalization of the Widrow-Hoff learning rule to multiple-layer networks and nonlinear differentiable transfer functions. Input vectors and the corresponding target vectors are used to train a network until it can approximate a function and associate input vectors with specific output vectors.

Properly trained back propagation networks tend to give reasonable answers when presented with inputs that they have never encountered. Typically, a new input leads to an output similar to the correct output for input vectors used in training, that are similar to the new input being presented. This generalization property makes it possible to train a

network on a representative set of input-target pairs and get good results without training the network on all possible input-output pairs.

In most of the ANN approaches, the network is trained by using *trainlm* and *trainbr* functions separately of feed forward back propagation algorithm and then result is compared to get the best output. *trainlm* is a network training function which is based on Levenberg-Marquardt optimization. *trainlm* is often the fastest back propagation algorithm available in MATLAB, although it does require more memory than other algorithms. *trainbr* is also a network training function that updates the weight and bias values according to Levenberg-Marquardt optimization. It minimizes combination of squared errors and weights, and then determines the correct combination so as to produce a network that generalizes. The function *tansig* is used as linear transfer function.

13.2 Fuzzy Logic

Zadeh (1965) has conceptualized the fuzzy logic theory to represent linguistics imprecision. The fuzzy logic implements and articulates the human knowledge processing in natural way. Fuzzy logic is defined by the theory of fuzzy sets. A fuzzy set is defined as a set which has clearly defined crisp boundaries. The partial degree of membership is also possible in fuzzy sets. Membership function is used to define the mapping of each input space to membership value between 0 to 1. The defined input space is named as universe of discourse. There are various membership function, the simplest membership functions are trapezoidal and triangular. The fuzzy logic reasoning can also be considered as standard boolean logic superset.

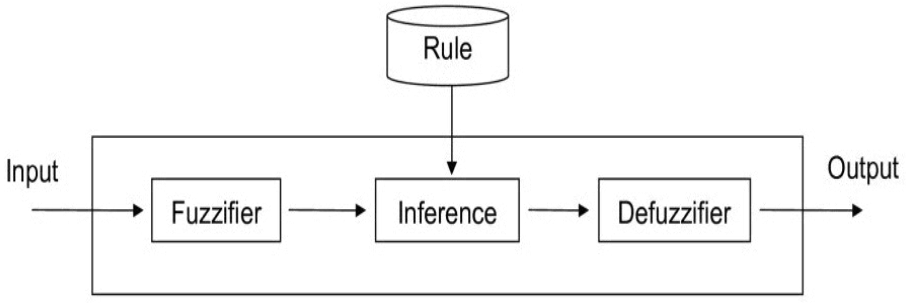


Fig. 1.4: Fuzzy Inference System Process

If-then rules are used to characterize fuzzy rule base. The collection of all the fuzzy rules creates the inference engine for the fuzzy system. Basically, these if-then rules simulate the human reasoning and are able to make the decisions in case of uncertainty and imprecision. If-then rules are nothing but the form of "if the value of x is a then value of y is b". In this case a and b are the values in universe of discourse range of x and y which belongs to X and Y. The part before "then" is called antecedent and later is consequent. As shown in figure 1.4, the interpretation of the if-then rules needs the fuzzification process of inputs and defuzzification of output to understand again.

Fuzzy logic provides a technique to deal with imprecision and information granularity (Sivanandam, 2007). Fuzzy logic offers a particularly convenient way to generate a mapping between input and output spaces by using natural expressions (Zadeh, 2002). In direct contrast to neural networks, which take training data and generate opaque models, fuzzy logic is based on if-then rules, which are designed by considering the opinion of experts from that domain. It has been found that the most accurate prediction models are based on analogy and experts opinion. Expert-based estimation was also found to be better than all regression-based models (Musilek *et al.,* 2000). Hence, the use of fuzzy logic in prediction is desirable since expert knowledge can be incorporated into the fuzzy prediction models.

Major advantage of this approach is that it is less dependent on historical data. Fuzzy logic models can be constructed without any data or with little data (MacDonell *et al.*, 1999 and Ryder, 1998). This makes fuzzy logic superior over data-driven model building approaches such as neural network, regression and case-based reasoning. In addition, fuzzy logic models can adapt to new environment when data become available (Sailu *et al*., 2004).

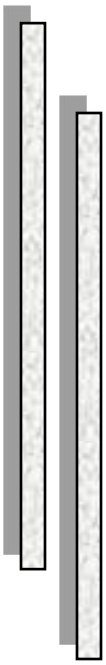
1.3.3 Adaptive Neuro-Fuzzy Inference System

In interdisciplinary era, the recent trend is to combine the different strategies by taking complementary advantage of each strategy and create a more efficient robust system. Following this, the various hybrid soft computing techniques have been introduced such as neuro-fuzzy, evolutionary-neural, evolutionary-fuzzy and evolutionary-neuro-fuzzy systems. The hybrid techniques perform better in case of

solving the specific type of real world problems but yield a complex assessment process because of involvements of various disciplines. Neuro-fuzzy is the most commonly used hybrid soft computing technique. Neuro-fuzzy model is implemented by taking the inputs to fuzzy inference system and then fed to ANN learning algorithm for training and testing. This technique uses the logical decision power of fuzzy logic and adaptability advantage from ANN. This combination increases the predicting capability in case of data analysis problem.

1.4 Conclusion

There is always a need to predict the software metrics attributes in software engineering quality assurance for software engineering process improvements. There are evidences when we need to predict various factors of software metrics such as defects, reliability, quality, maintainability, reusability and other product or process metrics which helps in software process improvements. It is easy to quantify the metrics if the metric value can be measured directly, but if it needs to be derived using the combination of other independent and influencing factors, then soft computing techniques can play an important role to increase the accuracy and efficiency. The principal constitutes of soft computing techniques indicates that we can apply fuzzy logic, artificial neural network and neuro-fuzzy as a standalone for the applied and interdisciplinary research but also the combination of these concepts can be used to get the better results.



Chapter 2

Literature Review and Related Work

2.1 Introduction

Software development processes were not so complex in early days, but in last two decades, the highly complex, large and mission critical system are being developed. In development of such highly complex system, there is a need of well defined set of software processes which can quantitatively control the overall software development. The conventional and unconventional techniques have been designed to quantify the development process. The conventional techniques include the mathematical and statistical regression analysis. Unconventional analysis includes mainly the soft computing approaches such as fuzzy logic, artificial neural network, neuro-fuzzy, genetic algorithm and others.

The successful applications of soft computing and also the rapid growth of interdisciplinary studies suggest that the impact and necessity of soft computing will be experienced increasingly in coming years. The researchers' communities have used soft computing approaches for many applied research areas like Mathematics, Physics,

---

The major findings of this Chapter have been published in Software: Practices and Experience, John Wiley & Sons, December, 2012, Vol. 42, Issue 12, pp: 1531-1548. (*SCI Indexed)*

Statistics and other areas as well but there is huge scope for interdisciplinary research in software engineering using soft computing approaches. Soft computing is likely to play an important role especially in science and engineering, but eventually its influence may extend much farther in domains like computer applications and others. In many ways, soft computing represents a significant paradigm shift in the aims of computing which reflects the fact that the human mind, unlike present day computers, possesses a remarkable ability to store and process information. The information in mind can be pervasively imprecise, uncertain and lacking in categorization. The similar problem exists in software engineering to represent and classify the information. It has been proved that soft computing techniques are now established and are more efficient in some areas of software engineering research. It is recently explored by the software engineering researchers and they validated the soft computing techniques which are useful for software industries.

There has been a lot of analysis and research to quantify the software metrics and development processes. The statistical and unconventional methods are proposed to measure the software attributes, the detailed analysis of these methods is presented in next sections of this chapter. The researchers have also started exploring the soft computing for software metrics and process design. The common conclusion of all the proposed approaches is that a single defined and presented soft computing approach may not work in all condition but these are better than conventional methods to solve the specific software metrics problems. Still, there is a need of more efficient soft computing approaches which can predict the future occurrence of an entity. Soft computing approaches are well known techniques to solve the complex prediction and measurement issues with good level of accuracy and efficiency. The presented soft computing approach in further chapters are applied across the mixed domain of software engineering i.e., structural, object oriented and component based systems. It is found (Singh et al. 2011; Aggarwal et al.2005a; Aggarwal et al. 2006; Pedrycz et al. 2004; Huang 2005; Sandhu 2010; Yuan and Zhang 2011; So et al. 2002) that there is a need to develop metrics to include the many more advantages of soft computing methodologies for:

- Metrics which are more efficient in prediction of an entity.

- Metrics which can work effectively with partial or small set of data. Should have been tested with the large project data successfully.
- Metrics which can adopt the changes of the environment/domain.
- A set of small metrics which can yield to a managed process and lead towards the software process improvements.
- The model which can provide the better indication of the quality of product and process.
- The metrics model which can interpret and process the common language variables efficiently.
- A more practical soft computing approach which can be adopted efficiently by software industries.

For the various software metrics, different soft computing approaches are designed but we have analyzed the principal constitutes of soft computing such as fuzzy logic, artificial neural network and neuro-fuzzy for the software defect, reliability, quality, reusability and maintainability and some related metrics. This chapter contains the analysis and literature review of the existing fuzzy logic, ANN, neuro-fuzzy approaches and methodologies which have been used for the design of the software defect density, reliability, quality, reusability and maintainability and other related metrics.

Following sections are divided into two major categories. Firstly, conventional approaches have been explored for various metrics such as software defect density, reliability, quality, reusability and maintainability. Next, soft computing approaches are reviewed for the set of above metrics, and the approaches are logically divided into three sections: fuzzy logic, ANN, neuro-fuzzy.

2.2 Conventional Approaches for Metrics Analysis and Design

Yasunari *et al*. (1995) provided the analysis of the relationship between the review, productivity and the software quality. They used real data from 36 projects to study the behavior. Khairuddin and Elizabeth (1996) proposed a maintainability model, and included factors, namely, modularity, readability, programming language, standardization, level of validation and testing, complexity, and traceability for evaluating the maintainability of software systems.

Gui and Paul (2000) proposed coupling metrics for assessing and ranking the reusability of java components, which are retrieved from the internet by a search engine. An empirical comparison of the proposed metrics with several other existing metrics is also done, which concludes that the proposed metrics are superior with better results in ranking the components according to their reusability. Dumke and Schmietendorf (2000) proposed a set of reusability metrics for javaBeans components. The metrics are adapted from structured and object-oriented design context. This work considers the source code to measure the metrics, and therefore cannot be applicable for black-box components.

Reussner (2003) presented a framework for reliability prediction in component based systems. The approach is developed using the analytical and empirical techniques for the component type software architectures. The method is evaluated with an online banking system. Gill (2003) suggested some guidelines for high reusability for software components. These guidelines include conducting reuse assessment, performing cost-benefit analysis for reuse, adoption of standards for software components, selecting pilot projects for deployment of reuse, and finally identifying the reuse metrics. Washizaki *et al.* (2003) proposed a metric suite for measuring reusability of black-box components. These metrics are based on the limited static information that can be obtained from the black box components. In the presented study, quality factors namely, understandability, adaptability, and portability affect reusability of a component. Six metrics are proposed to measure these factors. Another interesting work for measuring software maintenance project effort estimation has been carried out by Ahn *et al.* (2003). They proposed a software maintenance project effort estimation model, which is based on the function point measure and 10 maintenance productivity factors. The regression analysis is performed by taking the data from 26 maintenance projects.

An independent validation performed by Goulao and Abreu (2004) indicated that the metrics are unreliable for components with a number of features on their interfaces. Boxal and Araban (2004) considered interfaces of the components to measure the reusability. The work assumed that understandability affects the level of reuse. Understandability of a component can be made through its interface properties. Ardimento *et al.* (2004) made the assessment that if a component is difficult to

understand, then it will be difficult to maintain as well. The authors also advised the trial usage of a component before adopting it for application.

Nachiappan and Thomas (2005) analyzed the general regression models, and developed a code change based relative measure model for prediction of defect density. The presented method is able to predict the faulted files with good level of accuracy. Rotaru *et al.* (2005) considered adaptability, composability, and complexity of a component to describe its reusability. Composability and adaptability assess the ability of a component, which is to be integrated as a part of a software system. Composability is the ease in combining a component with others, while adaptability is the ability of a component to accommodate changes in the environment.

Kajko-Mattsson *et al.* (2006) discussed several problems related with the maintenance of the software systems. They proposed two maintainability models separately for product and process. The research work claimed that the commonly defined model of maintainability can substantially contribute to the overall success of the software. Misra and Kilic (2006) emphasized that a new software complexity measure should also be satisfied by measurement theory criteria. However, most of the developers of new complexity measures do not care about measurement theory. In their paper (Misra and Kilic, 2006), they evaluated measurement theory, and point out, why measurement theory is not well applicable for evaluating complexity measure.

Dapeng and Shaochun (2007) developed metrics to define the coupling which measure the classes' relationship. The method has also been empirically validated on open source projects. Meine and Miguel (2007) studied the relationship of dependability with other metrics. They used a very large collection of small projects to conduct the experiment. The conclusion is drawn that there is a high level of positive relationship between the size and halstead volume metrics. Also, they found a good level of correlation in size and complexity which can help for quality analysis of a project.

Yingjie *et al.* (2008) developed estimator model for classification of software quality. The authors also proposed a unique method for source code size and complexity reduction for performance improvements. Wadhwa *et al.* (2008) evaluated maintainability of small-scale web applications built on J2EE, .NET and Ruby On Rails (ROR). The maintainability criteria considered modifiability, testability, understandability and

portability. They found that the ROR implementation is better on modifiability, testability, and understandability, while J2EE implementation is the most portable. The results led to conclusion on the maintainability of small web applications with respect to underlying architecture and development environments which three platforms provide. The results are expected to vary for medium and large-size web application. Raimund Moser *et al*. (2008) conducted a comparative study of efficiency for prediction on two different metrics set. They considered the metrics from process and product, and analyzed the metric to find the defected files. Ayse *et al.* (2008) developed the classifier based defect prediction model. The empirical validation shows that ensemble of classifier can be used to enhance the ability to predict the defects. Mauricio and Hisham (2008) indicated some cases when a set of metrics can provide a good level of fault detection at early stage of SDLC, and work as an input for the remaining phases to reduce the errors.

Georgios (2009) proposed a framework to predict the defects and quality of complex software by narrowing down the set of independent metrics to five. The author concluded that a small number of metrics set can also have good correlation with the defects. Gao *et al.* (2011) selected a set of metrics for defect prediction based feature selection technique. The presented model is able to perform with good accuracy. The model is validated over the defect data of the multiple release of the large software system

Tahir and MacDonell (2012) analyzed the aspects of using dynamic metrics for evaluation of software product quality. They conducted a systematic review and survey of dynamic metrics application for metrics evaluation. The conclusion is drawn that mainly the research community have focused to address dynamic metrics for maintainability and complexity, and now there is a great attention on dynamic metrics in metrics design and analysis. Wang *et al.* (2012) presented the test coverage and failure data based reliability growth model. They developed test coverage function based on beta and two mean value functions. These are validated against the reliability data sets from two projects. Wu and Ding (2012) combined the different reliability models using the static and dynamic weight, and presented a hybrid software reliability model.

2.3 Soft Computing Approaches in Metrics Design

Although, various models and methods have been proposed by researchers using statistical and empirical methods, but there is a scope of using soft computing approaches to explore the software metrics. In turn, the software product or process can be measured and analyzed with better accuracy and less efforts. In literature survey, it is observed that researchers used the soft computing approaches for some of the software engineering and other related research problems. They are able to discover better or enhanced results by using these approaches. Also, soft computing models and approaches can be used to validate the empirical methods to categorize the subset of measures. The soft computing techniques can provide the better indicators in the software development process. Kumar *et al.* (2012) conducted a survey on quality aspects for the component bases systems. The authors focused on the specific parameters for the proposed conventional approaches. For the selected literature, they analyzed the existing models and metrics of software quality considering complexity, maintainability, reusability, dependability as the key factors for quality. The conclusion is drawn that the unconventional approaches such as soft computing can provide good results in software quality factors evaluation.

2.3.1 Fuzzy Logic Approaches

Andrew and Stephen (1999) indicated that fuzzy logic is better method for modeling of software metrics. The authors enforced that the different models of fuzzy logic can be used in software development cycle. The modeling can be done using the available information metrics at different stages of SDLC. MacDonell *et al.* (1999) discussed the fuzzy logic modeling benefits, especially for the development efforts estimation. They proposed fuzzy model and validate against the real project data set. It performed reasonably well compared to the regression-based models.

So *et al.* (2002) indicated that the inspection data gathered during the development can be used as an indicator to provide some insight information for software process improvements. They introduced a fuzzy logic analysis of the inspection data, and predicted the fault-prone modules. They defined a fuzzy membership function and fuzzy rules are generated to analyze to classify fault-prone module. The empirical evaluation is done using inspection data gathered from launch interceptor program, and regression

analysis indicated that proposed fuzzy model can be used to find faulted modules and reliability of software system. The authors claim the advantages of this approach because the inspection data can be organized and analyzed in the form of fuzzy and the model provides a mechanism for such analysis. The rules can be generated with less empirical data and the performance can be tuned when more empirical data available. Practically, in industries, limited human resources are available to work on development with tight schedule. Hence, their model not only reduce the extra efforts but also give good insight information to the project manager for effective resource allocation on the basis of the prediction of the defect prone modules and reliability of the module or system.

Liu *et al.* (2003) developed the framework for early warning in software development life cycle. The proposed system can find the risk at early stage of SDLC using fuzzy logic. The system uses the quantified approach of the process, product and organization metrics. Direct measurement of risk, using the combination of these metrics and fuzzy logic, risk assessment cannot be done. Hence, the developed framework can help for risk analysis during the project development.

Saliu *et al*. (2004) proposed the ANFIS approach for the efforts prediction using Constructive Cost Model (COCOMO) model. The system is able to adapt to the different environment in case of the new data set is provided. The framework solves the issues in existing models such as absence of less capability of historical data analysis, expert decision power and performance tuning based on the nature of the data. The fuzzified inputs are considered as organic mode, semidetached mode, embedded mode, size and 15 cost drivers form COCOMO. The output is measured as predicted efforts.

Aggarwal *et al.* (2005a) used a fuzzy-based approach to measure the software maintainability by considering the average number of live variables, average life span of variables, average cyclomatic complexity and the comments ratio. The fuzzified inputs and outputs are used in experimental setup. The combination of total 81 rules used for making the fuzzy inference system to predict the maintainability. To validate the proposed model, they considered 10 software projects of procedure oriented methodology. The selection criteria of the ten projects are based on the defined set of input availability in data set of the projects. By introducing some errors in these projects, the time is measure for maintenance to correct the system. Then the maintainability is

measured with the fuzzy model and data sets. They found a strong correlation between maintainability and maintenance time.

Pizzi and Pedrycz (2006) demonstrated the efficacy of fuzzy integration in combining classification outcomes from multiple classifiers in the discrimination of complexity and maintainability of software objects. The fuzzy integration method produced a 15% improvement in predicting software complexity compared with the best individual classifier using linear discriminate analysis (0.86 vs 0.75) and 29% improvement in predicting software maintainability (0.75 vs 0.58). These results are achieved using less than a quarter of the original software metrics. Dick and Sadia (2006) demonstrated the fuzzy clustering analysis of the quality data gathered from open-source. The data set of object oriented quality metrics and defects used for fuzzy clustering. The developed approach of fuzzy clustering is able to identify the modules, which are having the higher defect density. They calculated the correlation between the large data set attributes to find the positive or negative correlation between the out and the different attributes. The principal component analysis is also used on the Mozilla project data set, which is used for the validation of the presented model. Burgin *et al.* (2006) analyzed about the fuzziness in software engineering process and metrics. As per the analysis, the software process can be quantified using the metrics, and metrics values can be fuzzified for assessment.

Grover *et al.* (2007) extended the ISO 9126 model to add one subcharacteristic, trackability under maintainability, and proposed a fuzzy logic based approach to predict the maintainability of the CBS. Authors considered interaction complexity, reusability, testability, understandability and trackability as inputs to predict the maintainability. Inputs are designed by fuzzy sets as low, medium and high, while maintainability as very low, low, medium, high and very high. A total of 243 rules are provided to the fuzzy inference engine to get the output. The value of maintainability is measured by using a defuzzification process. The proposed model is applied on a classroom-based project to evaluate its maintainability, and the results are encouraging.

Chang *et al.* (2008) proposed fuzzy logic based algorithm and model to evaluate the software quality. They developed a fuzzy based analytical hierarchical process to solve the uncertainty issues in service evaluation by representing them in form of fuzzy

functions. The authors claim that fuzzy based analytical hierarchical process can evaluate the software quality and provide important guidelines to project managers, but also it can be a helping tool for developers to evaluate the software quality. The applied fuzzy logic concept in quality evaluation will be able to analyze the limitation and positive attributes of the software product. The proposed approach inherited from ISO-9126-1 model, and to evaluate the software quality, the software metrics attributes, functionality, reliability, usability, efficiency, maintainability and portability are considered as independent and deciding factors for quality.

Sharma *et al.* (2009) used fuzzy logic for prediction of maintainability of component based systems. This paper discussed the maintainability related issues in case of component based system development, and the authors implemented the fuzzy inference system to predict the maintainability as an important quality factor. The influencing factors of maintainability are identified and grouped into different fuzzy sets. Although, the approach has not been validated on large and complex component based system but authors have successfully validated the presented approach using the classroom projects' data. The approach is validated using analytical hierarchy process with two case studies, which are based on class room projects. Khatatneh and Mustafa (2009) developed a new fuzzy expert system to predict the software failures and defects. This model focused on a particular dataset behavior in predicting failures. Focusing on a particular data set, the experiment is performed to develop an more effective and accurate model.

Jatin and Gaur (2012) highlighted the importance of reusability assessment for component based system development. They used several quality factors and fuzzy inference system to estimate the reusability of components. However, they did not validate the proposed approach using real project data. The authors identified customizability, configurability, compatibility, interface complexity and portability as influencing factor for reusability, and can be used as independent attribute to assess the reusability.

2.3.2 Artificial Neural Network Approaches

ANN is characterized by its architecture, learning algorithm and activation functions. A set of data in the form of input and output is used for training the network. Once the network is trained, it sets a relationship between input and corresponding output. This network can be used to predict the output for any input set of data. The researchers considered different factors as inputs for training the network to predict the dependent output attribute. The following sections describe the work in this direction.

Karunanithi *et al.* (1992) used ANN for reliability prediction. The feed-forward neural network used for predicting the faults and taking the execution time as independent variable. They analyzed the variation effects of training parameters and data sets, and concluded that depend upon the data set behavior, different types of models can be built.

Hong and Kim (1997) presented an empirical investigation to analyze the relation between the different metrics to the change request counts of the function blocks. The authors analyzed the degree of relation between the identified metrics and faulted block which encountered during the software development phase. Based on the proposed metrics, the neural network is enough capable to differentiate the function blocks which are having faults. The developed model is applied for empirical investigation to find the faulted function blocks in release development of Automated Teller Machine (ATM) switching system. Generally, most of regression models have been developed to classify the fault-prone modules. The faults count in the faulted module has to be measured. Hence, the author applied the neural network to find the count of defects. They used 0.4 millions LOC from a large telecom software, and the data set is divided into the training and test data sets.

Kumar *et al.* (1998) predicted the faulted modules and number of errors at early stage of software development life cycle. Principal component analysis technique is used with the two types of neural network training rules. They presented various neural network variations and techniques for software quality evaluation. In regression and discriminant analysis, the assumption is made that independent variables used in analysis are non-collinear, hence PCA is used to achieve this target in the proposed model. The neural network models are good to use when there is long set of data needs to be

analyzed to reach at a conclusion. The conclusion is drawn that neural network has the capability for a good quality management tool in software development to uncover the basic problems of attributes' prediction.

Khoshgoftaar *et al.* (2000) defined the testability in connection with the test case failure due the program fault. They conducted a case study to predict the testability of each module of real-time avionics software. They defined the goal for the study as to predict an attribute, which is having values between 0 and 1 and highly skewed distribution approaching to zero. According to authors, to find the testability on large and complex software system needs a lots of computation, hence neural network can be better techniques for making the model for such type of prediction from large data sets. The neural networks are very good for modeling and establishing the relationship in case of nonlinearities.

The ANN based approach is considered to be very helpful in estimating the maintainability of a system. ANN is inspired from the biological nervous system, therefore it process the information similar to the human brain processing. Singh *et al.* (2004) considered readability of source code, documentation quality, understandability of software, and average cyclomatic complexity as input variable to measure the maintainability of the software systems by using the back propagation algorithm. The results are every encouraging with 91.42 % prediction quality. Kanmani *et al.* (2004) applied the regression neural network to predict the fault ratio as a quality attribute. The analysis is done using fault ratio as dependent factor. They used four attributes for General Regression Neural Network (GRNN) such as coupling, cohesion, size and inheritance. All possible combinations of these four metrics are considered. The authors concluded that object oriented metrics can be normalized in between 0 and 1 for better results, and used for fault ratio prediction. To make the model simple, they tried to reduce the input vector size. They found that three factors size, coupling and inheritance give the best fault ratio prediction.

Su *et. al.* (2005) presented the artificial neural network based approach to estimate the reliability, and designed a model for reliability estimation. The ANN model is used for modeling of the dynamic data set, and the trend analysis is also provided in the study. They have taken the four projects data set to validate the proposed model. It is concluded

that the ANN model performs well for reliability prediction. They compared the results of the developed model with conventional method, and proved that ANN provides better predictability. Quah and Thwin (2005) presented a model for maintenance efforts prediction using two types of neural networks. They selected object oriented metrics for software quality prediction. The outcome of study shows that the ANN models are useful for software quality prediction. To show usefulness of presented model, the results of the ANN model are compared to the multivariate regression model. They also explained the useful of neural networks in software quality prediction.

Bhattacherjee (2006) presented ANN technique with different architectural attributes such as five neurons, one hidden layers, seventeen hidden nodes and one predicted output. In the presented model, they used different program and personnel attributes' factor for the modeling of software development time. The validation is done to predict the development time using different project data sets. In 91% cases, the errors were within the 25% for the training data set. In 70% cases it is within the 20%. They demonstrated the suitability of ANN for such type of software attributes prediction. Aggarwal *et al.* (2006) identified principal components of eight object-oriented metrics as independent variables to estimate the maintenance as a quality attribute for object-oriented systems. Back propagation algorithm is used for training the network. The object oriented metrics includes, lack of cohesion, number of children, depth of inheritance, weighted methods per class, response for a class, data abstraction coupling, message passing coupling and number of methods per class. The data are taken from commercial product of user interface system and quality evaluation system. The final aim of the presented model is to predict the quality of software on the basis of the count of the lines changed in each class. The presented model shows the good accuracy with 0.265 mean absolute relative errors.

Shukla and Mishra (2008) used 14 factors as cost drivers for their study, and trained the network by considering different numbers of hidden layers and neurons. The results indicated that the neural network is able to predict the maintenance effort. The data sets are divided into two parts, 60% of the data vectors used for training purpose, and rest used for the testing and validation.

Lo (2009) proposed different variants of software reliability growth models with improvements in conventional models. They described the model to predict the faults, and also proposed a fault detection process. The remaining errors in software are responsible for failure in the system and reduce the quality and reliability index. Hence, to analyze the history of failures in the system, they used back-propagation neural network. Neural network based reusability measurement approach was presented by Sharma *et al.*, (2009). They considered four influencing factors which have more impact on reusability of component based system. The four used factors are understandability, portability, customizability and interface complexity. To train the neural network, the data sets of the defined four independent factors are used, and the reusability is predicted in output. The authors used various combinations of neurons' count and hidden layers in the neural network simulation.

Pomorova and Hovorushchenko (2011) indicated that neural network methodologies are the new perspective for software quality evaluation. They used artificial neural network for analysis and prediction of design stage metrics such as complexity and quality characteristics. For testing, total of 324 input vectors used, and the analysis demonstrated that in case of the specific data set, the different algorithm does not matter in terms of mean square error.

2.3.3 Neuro-fuzzy Approaches

Baisch *et al.* (1995) proposed a model for improvements in software productivity on the basis of quality. The changes in modules can be predicted using neuro-fuzzy inference system. They used the history data to build the neuro-fuzzy model. The authors suggested that the neuro-fuzzy model can be designed using the data set from the similar type of past project. The fuzzy inference system rules should be understandable, and interpretation is necessary to avoid the accidental combination. The authors also claimed that if the maintenance data per module exists and gathered, the proposed approach can predict the maintainability on the basis of the metrics' data and module.

Pedrycz *et al.* (2004) analyzed the software quality data using the neuro-fuzzy approach. They discussed that how specific patterned data can be used in neuro-fuzzy processing, and take advantage of neuro computing for decision making. They indicated

that how self organizing and adaptable system helps in software system data analysis and visualization. The experimental part is carried out on Management Information System (MIS) data sets, which are available in software quality literature. The data set contain the complexity measure and the number of changes in the module. They discussed the two aspects of soft computing, supervised neural network and neuro-fuzzy system, and shown that how these two aspects solve the quality data classification problem in software engineering field.

Huang (2005) proposed a soft computing framework, neuro-fuzzy inference system to handle contributing factor dependencies for effort prediction. As per analysis by author, accuracy in efforts estimation is a major issue in project management activities. The proposed framework is independent of the underlying algorithmic model. The author integrated neural network, fuzzy inference system and conventional algorithmic model under one umbrella. The author claims that the framework provides robustness for unclear and vague inputs. Also, the framework can be integrated to other models with the learning capability of ANN and fuzzy expert knowledge. The validation is done using industry projects, and obtained good results for software cost prediction.

Sandhu *et al.* (2008) compared the fuzzy, neuro-fuzzy and fuzzy-GA approaches for reusability evaluation for components. The halstead software science indicator for volume indication, complexity, coupling and reuse frequency are used as independent input factors. The reusability of components is estimated using the input vectors combination of these factors with proposed model. Although, neuro-fuzzy approach performed well, but fuzzy-GA approach provided the best accuracy as compare to fuzzy and neuro-fuzzy approaches. They concluded that in the data set used for validation, fuzzy-GA produced the best results because it took advantage of robust decision making but no learning capability and data optimization of GA. The analysis indicates that fuzzy-GA has not been explored much for the problems which involve large data analysis and prediction of an entity, especially in prediction of software attributes.

Ardil and Sandhu (2010) used branch count, cyclomatic complexity, design complexity, essential complexity and number of lines as inputs. The mamdani type Fuzzy Inference System (FIS) and Adaptive Neuro Fuzzy Inference System (ANFIS) are implemented to predict the maintenance severity of software application. Neuro-fuzzy

based implementation is found to be better than FIS. The authors used public domain defect dataset of National Aeronautics and Space Administration (NASA) for the experimentation. They concluded that compared to other machine learning techniques, simple logistic methods and trees performs better and with accuracy of 65% in case of fault prediction. Beldjehem (2010) proposed a model to understand and predict the software quality. The neuro-fyzzy system is developed to quantify the impact of inheritance on evolution of class library. Also, they studied if inheritance can be used as an indicator for interface stability of class in terms of class version changes. The case study is conducted to understand the compatibility of the class versions so that these are maintainable and have backward compatibility. The author concluded that looking at the large software development issues, it is necessary to combine soft computing and software engineering, to design the intelligent and practical cost, efforts and quality prediction models. Kaur *et al.* (2010) considered eight object-oriented metrics, which were used by Aggarwal *et al.* (2006). To predict overall maintenance efforts, various soft-computing techniques and algorithms are implemented such as Feed Forward Back propagation Neural Networks (FFBPNN), Radial Basis Function (RBF), FIS and ANFIS. The results indicated that ANFIS gives the best prediction with minimum Mean Relative Error (MRE).

Yuan and Zhang (2011) described the method to find the reliability using neuro-fuzzy technique. The presented ANFIS model used the defect counts for prediction of the reliability of software system. They used expansion data for reliability prediction modeling. Using their model, the reliability prediction is acceptable with fewer errors. Singh *et al.* (2011) developed a neuro-fuzzy framework to access the reusability of software system. They used four independent factors, interface complexity, changeability, documentation quality and understandability. The neuro-fuzzy model is able to predict the reusability based on the four input factors with a better level of accuracy compare to other techniques and models. The research work concluded that the model gives the better results by using the learning ability and decision capability of soft computing. Also, they indicated that the similar model can be developed for prediction of reliability, quality, cost and effort.

2.3.4 Other Unconventional Hybrid Approaches

Reformat *et al.* (2003) proposed the computational intelligence based software quality analysis. They used genetic decision trees to classify the software objects for the quality assessments. The genetic classifiers of computational intelligence are used as software quality filters for software modules. Applying these genetic classifiers, software quality assurance team can predict the quality of the software modules, and revision can be done for the low quality parts. Madsen *et al.* (2005) explored the possibility of evolutionary computing usages in reliability engineering problems. They indicated that soft computing techniques can be used for software fault analysis and prediction, optimization of reliability process in the process of reliability analysis in software projects.

Balikuddembe *et al.* (2009) analyzed temporal probabilistic reasoning for software project management and process improvements. The authors proposed the probabilistic reasoning model for project planning by using the adaptive software engineering framework. Dahiya *et al.* (2011) used the genetic algorithm for designing of software maintainability metrics. The fuzzy logic based approach is used to establish the relationship between the independent and dependent attributes. Then, the genetic algorithm with condition number is applied to improve the stability of the fuzzy approach, and the fuzzy parameters are tuned. An interesting research work, gene express programming is used by Zhang and Xio (2012) for software reliability modeling. In experimentation, the population is divided into different blocks as per generation fitness. The new approach for reliability model is built for software failure using the block strategy gene express programming. Yang *et al.* (2012) presented a study for severity prediction of defect reports using feature selection technique. The author used three main feature selection methods such as correlation coefficient, chi-square and information gain. The model is validated with experiments on Mozilla and Eclipse components.

2.4 Some Identified Gaps in Research and Literature

The prediction can be done using conventional methods or soft computing methods *viz.*, fuzzy logic, ANN, neuro-fuzzy, genetic algorithm etc. The researchers have analyzed that these techniques are capable of approximating general non-linear

relationships to high degree of accuracy (Haykins, 1999) (Wang *et al*. 2005) (Shing and Jang, 1993). Kumar *et al.* (2012) conducted a survey on quality aspects for the component bases systems. The authors focused on the specific parameters for the proposed conventional approaches. For the selected literature, they analyzed the existing models and metrics of software quality considering complexity, maintainability, reusability, dependability as the key factors for quality. The conclusion is drawn that the unconventional approaches such as soft computing can provide good results in software quality factors evaluation.

In literature (So et al. 2002; Aggarwal et al.2005a; Aggarwal et al. 2006; Pedrycz et al. 2004; Huang 2005; Sandhu 2010; Yuan and Zhang 2011; Singh et al. 2011), there are evidences where soft computing performs far better than the basic conventional methods. Soft computing techniques are well proven practical techniques that simulate human behavior and easy to use. Adaptability and robustness are the most powerful parameters of these methodologies. However, we cannot make any conclusion but there is a big floor available for the nonconventional techniques, where we can design more robust and validated models in software metrics research. There is a need of more exploration of the metrics-based approaches for quality attributes, which can cater the need of more efficient and practical approaches.

The statistical and unconventional methods are proposed to measure the software attributes, the detailed analysis of these methods is presented in next sections of this chapter. The researchers have also started exploring the soft computing for software metrics and process design. The common conclusion of all the proposed approaches is that a single defined and presented soft computing approach may not work in all condition but these are better than conventional methods to solve the specific software metrics problems. Still, there is a need of more efficient soft computing approaches which can predict the future occurrence of an entity.

Soft computing approaches are well known techniques to solve the complex prediction and measurement issues with good level of accuracy and efficiency. The presented soft computing approach in further chapters are applied across the mixed domain of software engineering i.e., structural, object oriented and component based systems. For the various software metrics, different soft computing approaches are

designed but soft computing applications in the field of defect prediction, reliability modeling, reusability and maintainability has not been completely explored. Therefore, we have analyzed the principal constitutes of soft computing such as fuzzy logic, artificial neural network and neuro-fuzzy for the software defect, reliability, quality, reusability and maintainability.

2.5 Conclusion

The statistical and conventional methods have been used for measurements and prediction from older times. Soft computing is a relatively new paradigm in engineering methodologies, but now widely used in all the disciplines of engineering and science. The soft computing techniques have been explored better in other engineering fields than the software engineering area. A detailed literature survey is conducted for the quality factors such as defects, reusability, maintainability, maintenance severity and reliability. Considering the importance of these metrics in software quality evaluation, these metrics are considered for review. The analysis and review demonstrate that soft-computing based techniques are being used widely by researchers to solve the software engineering aspects, but it is very difficult to conclude that one approach is better than another. In addition, to solve the specific problem, other hybrid techniques such as neuro-fuzzy may also be found suitable with better prediction for different aspects. Also, it is unfair to compare the conventional and soft computing techniques because they have different fundamental characteristics to solve the problem. Soft computing techniques rely on the underlying algorithm. Also, it depends on the method or model which may be existing well-proven algorithm or technique, and in some cases, the new algorithm may be included to conceptualize the model. The adaptability is the ease to adjust in a new environment based on the adaptive nature of the approach. It is very important to determine the type of data used to validate the system. To increase the applicability, the system should have the input data, which are easy to collect, and preferably fewer data sets. A method which is not practical and easy to use may not be a good model. There are many models that have good accuracy, but in real-life situations, may not be useful because of the involvement of complex procedures. The model should have the ability to extend the method for improvements due to environmental changes.

The results of the analysis show that the research community has put many efforts to meet the challenges in metrics-based quality description software systems. There are good footprints of the empirical and soft computing (unconventional) techniques to explore the software engineering and metrics problems. It is observed that there are various validated and mathematically proven conventional approaches. The mystery of the successful techniques is underneath the underlying algorithm or model used in the formulation and evaluation of the methods. Although, the conventional algorithmic and statistical methods have been widely used and proven with accuracy, but due of the lack of adaptability and forecasting power of these techniques, the research community is moving towards the unconventional techniques such as fuzzy logic, artificial neural networks, neuro-fuzzy, genetic algorithm, probabilistic reasoning.

In literature analysis, it has been observed that few areas of software metrics are being explored by using the soft computing techniques but still there is a good scope of soft computing techniques application in metrics formulation and validation. The more efficient analysis can provide the better indicators for the software industries which can serve the new foundation in software processes measurements and improvements.

Form the literature analysis, it can be inferred that there are plenty of soft computing approaches proposed but there is a need of more analysis of soft computing approaches for prediction and measurement of software attributes such as defects, reusability, maintainability, reliability, quality etc. The issues in the proposed approaches which need to be addressed are simplicity, cost effectiveness, validation, efficiency, applicability and practicality so that the industries can adopt these approaches in the software development process. The researchers have extensively used soft-computing techniques and related algorithms to propose various software metrics. They have also compared the results with other techniques, but to conclude that one particular soft-computing technique is better than another may not be appropriate. The conclusion of better technique may be true for one characteristic by considering different sets of inputs and datasets, but may not be true for the same with different inputs. The above exhaustive analysis of literature led to the opinion that soft computing methods can be a useful and powerful tool for software industries to predict and estimate a software engineering attribute.

Chapter 3

Software Defects Prediction

3.1 Introduction

A software system needs changes or enhancements due to business requirement and technological enhancements. If we are able to predict the defect prone area of module in the system or subsystem, it can have the impact on the schedules, cost and the customer satisfaction. In literature, various investigations have been proposed to show relationship between the direct measures and defects. The prediction of entities is crucial in software metrics analysis and design for better control on software development. Defect density is an attribute of software quality, which gives the measure of quality and reliability for software product. It can be defined as the number of defects in defined area such as a files, Thousands of Lines of Code (KLOC) and module. Defect density can be measured as a sum of defects or faults in a file or module. The measurement of defects helps us to find the highly defected modules from a bunch of modules in the system. In a complex and sizable software base, 80% of the defects lies in 20% of the files. It indicates that there should be more focus on the infected 20% area and put more efficient

The major findings of this Chapter have been published in *COMPUTING AND INFORMATICS,* North America, Vol. 32, Issue 1, 2013, pp: 203-224. (*SCI Indexed)*

resource to work on it rather than the equal focus and resource distribution on the entire software base.

The prediction of defects during the development of the software system is crucial to produce the system with more reliability and quality. In last three decades, soft computing techniques have been vastly used in basic sciences and engineering disciplines. ANN, FIS and ANFIS can be used for universal approximations to solve uncertainty issues. As a part of software development process, various attributes need to be calculated during the SDLC. Apart from the normal calculation, it is a good practice to find the possibility of occurrence of good or bad incident, to take the preventive measure beforehand. After applying the soft computing approaches in other engineering disciplines, researchers are now trying to use these techniques in software engineering disciplines as well. In literature, it is shown that there are many software attributes, which can be predicted in advance by using the soft computing techniques. Few of the main attributes are defects, effort estimation, software reusability, software maintainability, software quality etc. The prediction can be done using conventional methods or soft computing methods *viz.*, fuzzy logic, ANN, neuro-fuzzy, genetic algorithm etc. The researchers have analyzed that these techniques are capable of approximating general non-linear relationships to high degree of accuracy (Haykins, 1999) (Wang *et al*. 2005) (Shing and Jang, 1993). The combination of ANNs and FIS leads to better results as they complement each other (Sivanadam *et al.* 2006). Khoshgotaar *et al.* (1997) and Tagi *et al.* (2002) have used ANNs and fuzzy systems to estimate the quality of a complex telecommunication system as well as for cost estimation, which indicates ANN and fuzzy to be a good tool in empirical studies for software engineering. If, we have the probability of incident in a project, it can be used as a precaution measure for the other similar domain projects. Also the occurrence of incident can be useful in subsequent releases of a software project or across the other project's releases. There are different project metrics entities, which can be predicted using FIS and ANN techniques.

This chapter proposes FIS and ANN based metrics to predict the defects in software system. These two approaches are formulated and applied to predict the existence of the defects in files of software project. Both the approaches have also been validated against the data sets of two commercial software products. Root Mean Square

Error (RMSE) has been used as the validation criterion. The analysis of the study shows that artificial neural network provides better results as compare to fuzzy inference system but applicability of best approach depends on the data availability and the quantum of data.

3.2 Defect Prediction and Defect Density

Defects are the software bugs, which are counted and calculated over a specified period of time during the software development. The defects counts can be collected during the testing and integration phase and may be uncovered before customer release or after customer release. The collection interval can be anything during the software development such as:

- All defects after the module creation
- Defects after regression testing starts
- Defects after code freezes
- Defects after a particular feature enhancement
- Defects after software product-release delivered to customer

The count of defects after software product release deployment in field.

Defect density is simply the ratio of the total defect to the entity size.

$$Defect\ Density = \frac{Total\ Number\ of\ Defects}{Software\ Size} \qquad (3.1)$$

According to IEEE Standard 982.2 (1988) and Pfleeger (1992), defect is an anomaly in any intermediate or final software product resulting from an error or fault, ranging from an incorrectly specified set of test data to an incorrect entry in user documentation. Defect may result to a failure. Failure is an inability of a functional unit of the system depending on the software condition to perform its required function, or to perform the function within required limits.

Several discussions have been made to define the standards for the entity or program size (Pressman, 2005; Albrecth and Gaffney, 1983; Boehm, 1981; Rosenberg and Microsyst, 1997). There are two standard size metrics to define the program size;

KLOC and function count. The proposed research is focused on the simplicity and practicality because there is a need of practical approach, which can be adapted by industries easily with less effort (Kumar, 2012). According to software engineering ethics, the domain of the projects does not matter much to adapt software engineering standards (El-Emam, 2001). A defect may not be uncovered for a long period of time in some situations. It may be uncovered when a particular code or logic of software is executed where the defect exists. A single defect can cause single or multiple failures at any point of time. Therefore, it is a good practice to predict them (Kan, 2003). We have designed the defect metric to predict the defects at granularity of the file level. The defect density can be easily calculated by dividing defects with the program size i.e., KLOC or number of files. The presented defect metric (Aggarwal *et al.* 2006; Khoshgoftaar *et al.* 1997; Graves 2000; Khatatneh and Mustafa 2009; Kumar *et al.* 2012) shall be able to predict the existence or non-existence of the faults in a file or module at any point of time during the testing phase. Also, while delivering the software release to customer, it can predict the number of defects which can occur after the customer release.

3.3 Soft Computing and Related Work in Metrics Prediction

Soft computing techniques are different from conventional computing paradigms, unlike traditional computing. It is lenient of imprecision, approximation, indecision and partial truth. Hence, soft computing techniques inherit some of the important properties of human mind. The main principle of soft computing is to achieve the accuracy with low solution cost and robustness. Recently, the researchers have shown a great amount of interest towards the use of soft computing approaches to solve the uncertain problems in software engineering. To classify modules as fault prone, Khoshgotaar *et al.* (1997) used ANN with back propagation training algorithm. To measure software maintainability, Aggarwal *et al.* (2005a) have used a fuzzy modeling technique. In area of open source system study, Gyimothy *et al.* (2005) have empirically validated important metrics for fault prediction using open source system. Graves (2000) proved that software change history can also be used to predict the defect density in modules and the systems. Khatatneh *et al.* (2009) have designed a new fuzzy inference system to predict the failures in software product. Kehan Gao *et al.* (2011) were focused to select the attributes

for software quality estimation. Comparative study has also been presented to evaluate the proposed approach for attribute selection. Fenton *et al*. (1999) discussed four general approaches to predict the number of faults in software system. The researchers have also presented the approach of finding the correlation between defect density and code metrics. Khoshgoftaar *et al.* (1990) used factor analytic variables to fit regression model to a number of defect data sets. To reduce the dimensionality of many related metrics to a small type of set, they used principal component analysis. Ohlsson and Alberg (1996) presented an analysis at Ericsson where they derived the metrics by analyzing the design documents and used these metrics to predict defect-prone modules before starting test phase. Gyimothy *et al.* (2005) shown empirical results with evaluation and validation of defect prediction of Chidamber and Kamerer (1991) metrics by taking open source software base. They have used linear, logistic regression and neural network for machine learning methods for prediction model.

Statistical models have also been explored vastly in prediction of software attributes. Soft computing techniques, particularly FIS and ANN have changed the focus in recent years. FIS and ANN are being applied across various domains such as finance, medicine and other engineering areas. Khoshgoftaar *et al.* (1997) have worked on a case study of avionics software to estimate the testability of each module by applying static source code metrics and ANN technique. They were able to predict testability because ANN has capability to establish the linear relationship. Aggarwal *et al.* (2006) have predicted the maintainability from object oriented metrics using ANN technique. They have examined the application of ANN for software quality prediction and taking the input variables as object oriented metrics. Maintenance effort was used as dependent variable in their study. Principal components of eight OO metrics were used as independent variables. The produced results show a good accuracy with ANN modeling. Yuan *et al.* (2000) have used fuzzy subtractive clustering to predict the number of faults. Aggarwal *et al.* (2005a) have developed a fuzzy model for measuring software maintainability. The inputs to the model are comment ratio, average live variable, average life span and average cyclomatic complexity. The output of the model is average corrective maintenance time. They have used mamdani style fuzzy inference system. Kumar *et al.* (2012) have presented the applicability, usability, and extendibility of the

existing approaches to rank the usage of these for component based systems in software industries. The authors have concluded that soft computing approaches could work better for component bases systems (CBS) also.

3.4 Defect Prediction Techniques

We have analyzed and designed the defect prediction metrics using FIS and ANN technique. Both the techniques are evaluated and validated across two projects of hybrid domain. In general, it is not easy to estimate some attributes directly. Defect density is one of these type metrics, which can predict using the other available attributes. For example, defects are affected by various factors and there is no simple method to measure or predict them. To predict the defect, the relationship needs to be established between direct measures (which affect the defect density attribute) with each other and drive the indirect measure as defect counts.

3.4.1 Selection of Variables

To design and validate the proposed metric, we need to finalize the set of dependent factors which influence the predicted metric. Also, the data collection strategy has to be defined for empirical validation.

3.4.1.1 Input and Output Variables

The data gathered during the development and testing of the project are used to develop FIS and ANN based defect metric which, in turn may be useful in the future projects and planning. Different attributes' values were captured but to find the most correlated influencing factors, correlation method and principal component analysis method are used. The defect distribution among the files and modules leads to an interesting research points:

- To identify the files which have the defects?
- To estimate the intensity of the defects i.e., the number of defects in a module.
- To predict the defect density for efficient planning of the resources during the subsequent releases or the future projects.

Based on analysis for most correlated factors from the data set and the discussion with the software architects, we have identified the set of software metrics which plays an important role and responsible to estimate the faults in software system. After the subsequent analysis, these are narrowed down to three metrics, which will be possible member for building a new metrics. These metrics can be collected easily by the software industries. The following facts have been considered while deciding the attributes:

- The metrics, which has high impact, either positive or negative impact on the number of defects in the system.
- Simplified and generally available metrics
- The attributes, which can be easily collected during the software development process.
- A large set of software attributes were available, which are narrowed down to few, using correlation.

Based on the practical experience, the three software metrics are finalized for input and defect density as the output. Following factors have been taken as input for the implementation of metric which influence defect density:

- *PREDD (Pre Release Defects):* Defects before release in a file and aggregated for module.
- *TLOC (Total Lines of Code):* Total Lines of code in a file and aggregated for module
- *VG (Complexity Metric):* McCabe cyclomatic complexity of a file and aggregated for module.

3.4.1.2 Empirical Data Analysis

Two projects of different domain are considered for conducting experiments. These projects fall into the following categories:

- An optical telecom project P_1, which is developed using embedded domain structural methodology and object oriented paradigms.
- A large telecommunication system P_2, in which modules are built using the mixed software engineering approaches such as structural, object oriented and component based.

Project P_1 is an optical telecom project, which is used as optical communication platform across the cities for data transport. It contains object oriented based system design as well as structural design. It has been developed in 4 years timeframe. Project P_2 is a 4G telecom project which is mix of three main software engineering methodologies such as conventional, object oriented and component based application project. Therefore, we have a good mix of various domain projects. This results to a good validation scope across various domain projects.

We have gathered a large set of data for three attributes from two projects. Using the bug history tool Rational Rose, we captured the values of three attributes from two projects and extracted approximately 6086 data set from project P_1 and 1620 vector record for project P_2 after removal of duplicate and outlier vector. The data for two projects consist of the many attributes. To conduct experiment, three attributes are extracted:

- Complexity
- Total lines of code
- Pre release defects.

The numerical values are captured for each of the attribute in P_1 and P_2 projects. The attribute values were captured at file, module and package level. Other than three attributes, we have collected the total defects in a particular file of the two selected projects. The following section gives the details of the attributes and the captured data:

- File name: It is the name of file in the module which helps to gather more information if needed during the data analysis.
- Defects before release: It counts the number of faults in a file which occurred during the span of approximate six months before giving the release to customer.
- Defects after release: It counts the number of faults in a file which occurred during the span of approximate six months after giving the release to customer.
- Total defects: It is the aggregation of defects before release and defects after release of a particular file in a project.
- Complexity metrics: Cyclomatic Complexity metric is computed for each file of a particular release of project.
- Total lines of code: It is the count of lines of code in a file or module.

3.4.1.3 Validation Criteria

The accuracy is percentage of match between predicted values and expected values of faults those are found in the file or module. For the present study, we have used Root Mean Square Error (RMSE). RMSE is frequently used measure of differences between values predicted by a model or estimator and the values actually observed from the experiment being estimated (Challagulla *et al.,* 2005). It is represented as square root of the mean square error as shown in equation given below:

$$RMSE = \sqrt{\frac{(A_1 - P_1)^2 + (A_2 - P_2)^2 \dots \dots \dots \dots \dots (A_n - P_n)^2}{n}}, \qquad (3.2)$$

here, $A_1, A_2 \dots, A_n$ are the original outputs and $P_1, P_2 \dots, P_n$ are the predicted outputs. The model which has got minimum value of RMSE and the high value of accuracy is considered as best model.

3.4.1.4 Principal Component Analysis

Principal Components Analysis (PCA) is a technique used to identify patterns in data, and quantifies the data to highlight their similarities and differences. It is difficult to find the pattern out of high dimension, which cannot be found using graphical representation. PCA has been used widely as a powerful tool for analyzing data in image processing field. PCA is also used to maximize the summation of each squared entry of each factor extracted (Kothari, 1998). Other than its accuracy, PCA gives the liberty to find the pattern in data without much loss of information. We used the characteristic of PCA to quantify the data set which is used in ANN training and validation.

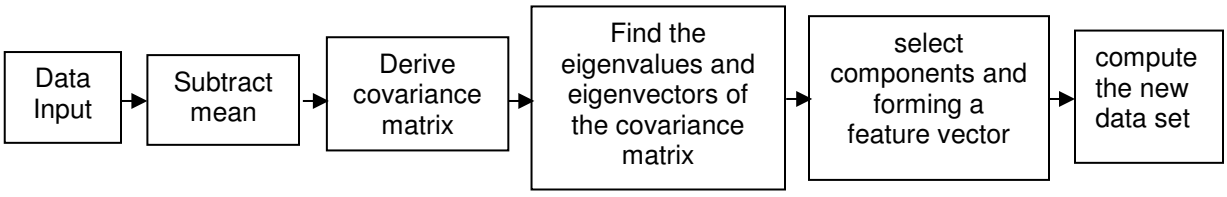


Fig. 3.1: Steps Followed for PCA

PCA is used to transform raw metrics data to set of unrelated variables. The PCA aims to derive a new variable p_i's, $i = 1, 2, \ldots, n$, which is called principal component from a provided set of variables x_j's, $j = 1, 2, \ldots, m$ (Aggarwal *et al.,* 2005b). The principal components can be calculated as per equation (3.3).

$$\left.\begin{aligned} p_1 &= b_{11}x_1 + b_{12}x_2 + \cdots + b_{1m}x_m, \\ p_2 &= b_{21}x_1 + b_{22}x_2 + \cdots + b_{2m}x_m, \\ &\bullet \\ &\bullet \\ &\bullet \\ p_n &= b_{n1}x_1 + b_{n2}x_2 + \ldots + b_{nm}x_m, \end{aligned}\right\} \quad 3.3$$

here, b_{ij} is calculated in such a way that derived principal components fulfill the following conditions:

- Principal components are uncorrelated i.e. orthogonal
- The first principal component p_1 has the highest variance; the second principal component has the next highest variance so on.

3.4.2 Fuzzy Logic Based Defects Prediction Technique

The FIS approach is designed by using the defect's influencing factors and metrics. Three basic metrics are taken as dependent factors, and defects are predicted.

3.4.2.1 Fuzzy Inference System

Zadeh (1962) has derived the concept of fuzzy logic to implement vagueness in linguistic variables. The concept of fuzzy logic implements and simulates the human knowledge as nature does it in daily life. Fuzzy logic theory is based on fuzzy sets. A fuzzy set is a set without a hard, clearly defined border. The fuzzy set may contain a fractional part of membership of an element. A membership function (MF) can be represented by a curve that defines how every point of input space, called universe of discourse, is related or mapped to a degree of membership value between 0 and 1.

Although there are various MF being used but triangular and trapezoidal are mostly used and simple MF, which are formed with straight lines. Fuzzy logic reasoning is a superset of boolean logic assumption. The interpretation of the if-then rule comprises the fuzzification of the input and applying the suitable fuzzy operators (Abraham, 2005).

Let X be a set of objects and x belongs to X. A classical set S, $S \subseteq X$, can be defined as a set of objects $x \in X$, in such a way that x can either contained or not contained in set S. A fuzzy set S in X can be defined as a set of ordered pairs

$$A = \{(x, \mu S(x)) | x \in X\}, \qquad (3.4)$$

where, $\mu S(x)$ is called MF for the fuzzy set S. A MF maps all objects of X to a value in membership function value which is called universe of discourse and lies between 0 and 1. Hence, the equation (3.4) is an obvious extension of classical definition in which the defined function can have a value between 0 and 1. Fig. 2.2 illustrates the fuzzification process.

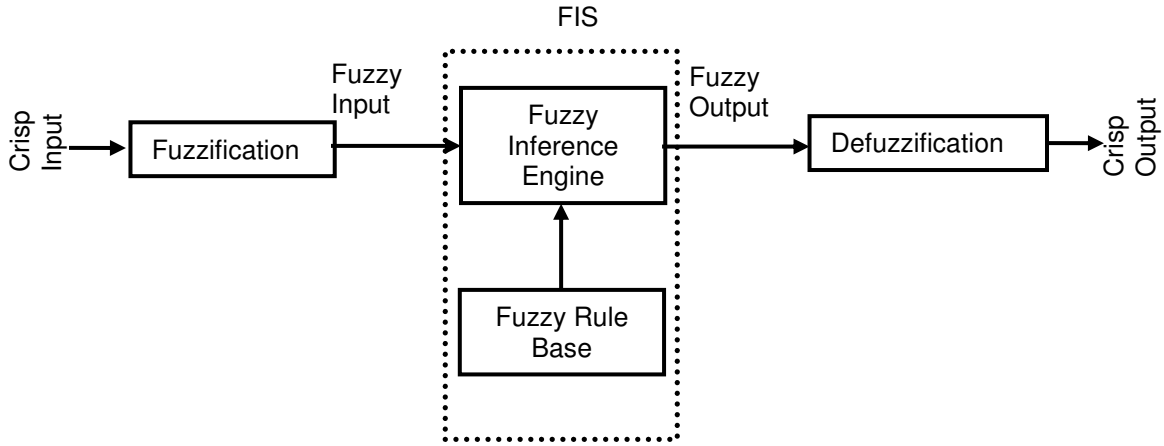


Fig. 3.2: Architecture of FIS

3.4.2.2 FIS Based Defect Prediction Approach

Fuzzy logic based approach is proposed for predicting defect density of different releases of software system using data from earlier releases. The proposed method has been validated for software system across few domains. It is validated on two domains of complex software projects. We emphasis to predict defect density by taking the direct

measures as independent attributes, which is most important metrics to estimate the system reliability and quality.

Defect density is an indicator, which depends on several other measures in the software system. We have proposed that defect density of a software system can be a measure of three most influenced factors as explained above. In the proposed approach, the fuzzy based algorithm has been designed to predict the defect density in a file or module. To find the defect density in module or software system, sum of defects in all files of a module or system has been calculated. The relation between the dependent attributes and the defect of a file can be given as follows:

Let x_n, y_n, z_n be the values of the complexity, total lines of code and pre release defects, respectively, which belongs to the nth vector of the module, then

$$D_{file} = f_m[x_n, y_n, z_n], \qquad (3.5)$$

here, f_m is the function of the x_n, y_n, z_n, which are nth value of the independent attribute in a file. D_{file} is the defect count in any nth file. In the experimental design f_m is considered as fuzzified and fuzzy rule function. The defect density of a module can be derived using equations (3.6).

$$DD_m = \sum_{file=1}^{m} [D_{file}], \qquad (3.6)$$

here, DD_m is the module's defect density. Using the above equation (3.5), defect count in a file can be calculated on the basis of three attributes values. Using equation (3.6), the defect density can be derived for a particular module. The function f_m is implemented using rules in FIS to achieve the desired goal of defect prediction.

As three attributes are direct measure, the combination of three factors can be used to predict the defect density. For complexity, let X be the input space and x_n be the value in the fuzzy boundaries [0,1]. Then equations (3.5) and (3.6) are implemented by applying fuzzy inference rules for the function f_m. The MF is used to map the actual attributes values into the fuzzy range. Similarly, for line count and pre release defects, the

triangular MF is used for fuzzification of the crisp values. The FIS engine is applied on the basis of the fuzzy rules and the defects are being predicted.

3.4.2.3 Experimental Design

The formulation given in equations (3.5) and (3.6) is implemented using the fuzzy toolbox in MATLAB 7.1. The proposed fuzzy logic based model is applied on all three factors as inputs and provides a crisp defined value of defect density using the decision logic. All inputs are classified into fuzzy sets values *viz.* Low, Medium and High. Also the output defect density is classified as Low, Medium and High. All probable combinations (i.e. 27) of inputs are taken into account for completeness to design the rule base. Each rule is connected to one of the three outputs based on the expert analysis and opinions. These rules are defined to implement the f_m given in the equation (3.5). A few of the proposed decision rules are shown as:

- If complexity value of a file is High, Total Lines Of Code (TLOC) is Low and pre release defect is Low then the defect density will be Medium.
- If complexity of a file is High, TLOC is Low and number of pre release defect is Medium then the defect density will be Medium.
- If complexity of a file is High, TLOC is Low and number of pre release defect is High then the defect density will be High.

All of the defined rules are inserted into the proposed fuzzy based model and a fuzzy rule base is created. For a particular input, depending upon a defined set of inputs, a rule is executed. Using the MATLAB rule viewer, output i.e. predicted defects are observed for a particular set of inputs in fuzzy toolbox. Table 2.1 shows the values of different parameters set for input space, predicted output used for fuzzification process.

Fuzzification and mapping of input space into predicted output are shown in Fig 3.3. MF for complexity attribute is indicated in Fig. 3.4. Rules for defuzzification process are shown in Fig. 3.5. In Fig. 3.3, complexity, TLOC and pre release defects are given as input to mamdani type FIS, and the out is produced as defect density. The value of defect density can be predicted by taking all three dependent values for input space. Fig.3.4 describes the MF used in the process of fuzzification and de-fuzzification. Fig. 3.5 is a snapshot of the rules from MATLAB's FIS toolbox rules viewer system.

Table 3.1: Parameter Values for System and Output

System	Name='DD' Type='mamdani' Version=2.0 NumInputs=3 NumOutputs=1 NumRules=27 AndMethod='min' OrMethod='max' ImpMethod='min' AggMethod='max' DefuzzMethod='centroid'
Output	Name='output' Range=[0 1] NumMFs=3 MF1='L-DD':'trimf',[0 0.2 0.4] MF2='M-DD':'trimf',[0.35 0.5 0.65] MF3='H-DD':'trimf',[0.6 0.8 1]

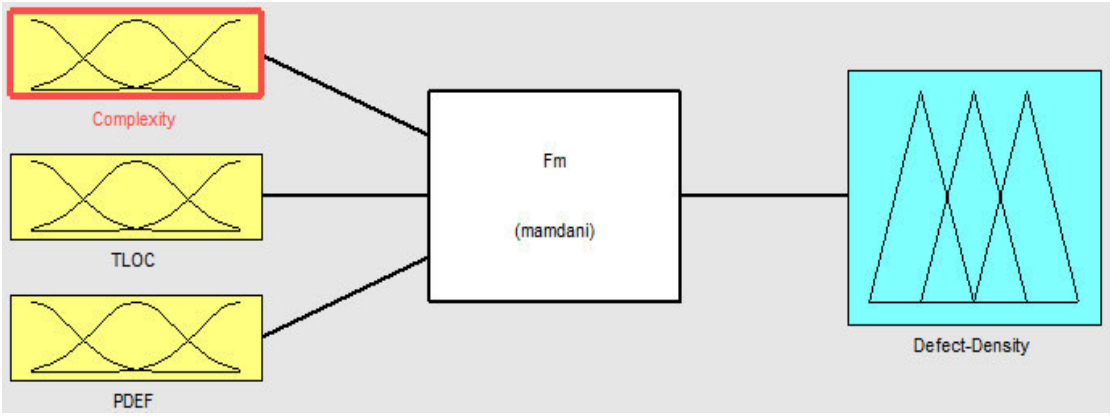


Fig. 3.3: Fuzzification of Independent Attributes Into Predicted Output

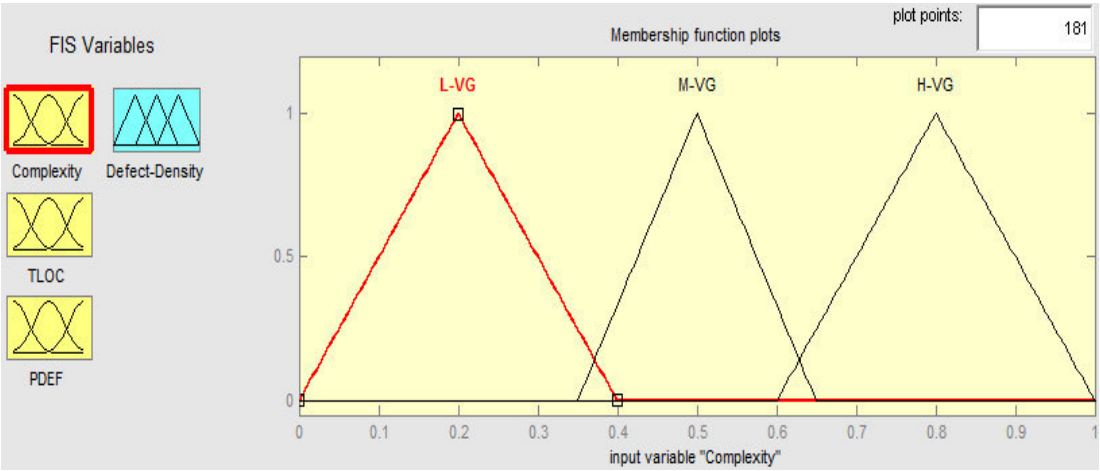


Fig. 3.4: MF for Complexity Attribute Used in FIS

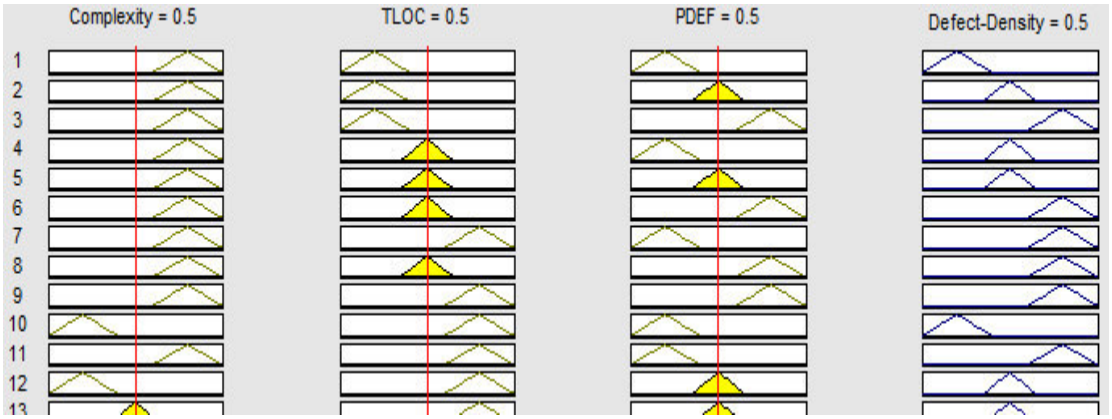


Fig. 3.5: MATLAB Rule Viewer Containing Few of the Rules

3.4.2.4 Validation of FIS Approach

The proposed approach is able to find the defect density at file and module level. As discussed above, we used data from two projects for approximate 6086 data sets of project P_1 and 1620 data sets of project P_2. The proposed approach is validated by using statistical evaluation of the predicted values vs actual data values. Proposed FIS based method is applied to predict the defect density for both projects' data sets.

The metrics for all the three input factors of defect density were collected for both the projects. These defined metric values are given as input to the proposed approach. The values of defect density, obtained from the proposed approach are validated using RMSE.

As shown in the Table 3.2, it is understood that values obtained from FIS based approach for defect density prediction have RMSE as 0.2860 and 0.3075 for project P_1 and P_2, respectively, of first set of 100 selected records out of the large data sets. The FIS validation is not carried out on the complete data set and validation on large complete data set is carried out using ANN approach, which is presented in next sections. The output of the fuzzy approach has been captured, and the root mean square is calculated error using the actual defects in the file. The comparisons of predicted defect density are made on the basis of accuracy and root mean square error.

The validation of the proposed formulation and experiment is done by taking the real project data as described in section 3.4.2.2. Proposed fuzzy based prediction metric has been applied on the dataset and the accuracy as well as RMSE are calculated using the equation (3.2).

Table 3.2: Validation Results of FIS

	RMSE		
Project	Data set:1	Data set:2	Data set:3
P_1	0.2860	0.2813	0.2943
P_2	0.3075	0.3124	0.2988

The results in Table 3.2 indicate that in best case the fuzzy logic approach is able to predict the defects with RMSE as 0.286. The FIS rules can be fine tuned to get better accuracy for a data set.

3.4.3 Artificial Neural Network Based Defect Prediction Technique

In this section, ANN approach is applied to formulate and design the defect density metric which is able to predict the defect density area in the software system.

3.4.3.1 ANN Based Defects Prediction Approach

ANN has been designed as generalizations of mathematical models of biological human brain nervous systems. Research communities have shown interest in neural networks after proposing simplified design of neurons by McCulloch and Pitts (1943).

In feed-forward network, which is being used in defect metric design, the signal travels in feed-forward direction from input to output units. In this type of neural network, there are no feedback connections present but the processing of data can broaden over multiple layers of neuron units (Abraham, 2005). Other recurrent network can contain feedback connections. Sometime the activation values of the neuron units have to pass through a recreation process so that neural network can grow to a constant and established state. After the relaxation process, these activations do not revolutionize further anymore. The simple architecture of an artificial neuron is described in the Fig. 3.6.

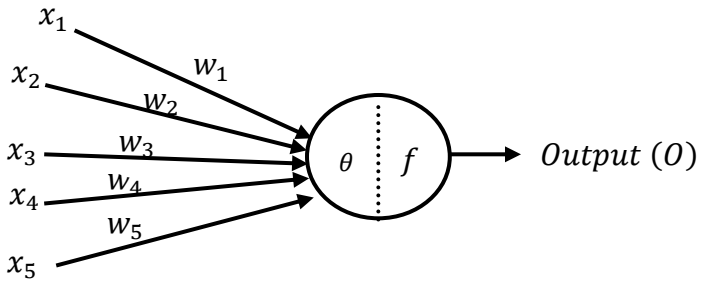


Fig. 3.6: Artificial Neuron Structure

The signal flow from input $x_1, x_2, \ldots, x_n$ is considered to be unidirectional which corresponds to consolidated output signal (O) of a neuron. The output signal O of a neuron is defined by the equation (3.7).

$$O = f(net) = f\left(\sum_{i=1}^{n} w_i\, x_i\right), \qquad (3.7)$$

where, w_i is defined as weight vector for input x_i (i= $1, 2, \dots, n$), and the function $f(net)$is defined as a transfer function. The variable net is defined as a scalar result of the input vectors and weight,

$$net = W^T X = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n \,, \qquad (3.8)$$

where, $W = (w_1, w_2 \dots, w_n)$, $X = (x_1, x_2 \dots, x_n)$, T is defined as transpose of a matrix, and the output value O is computed in simple case using the following equation

$$O = f(net) = \begin{cases} 1 & if\ W^T X \geq \theta \\ 0 & otherwise \end{cases} \qquad (3.9)$$

This category of node is called a linear threshold unit. θ is defined as the threshold level of neurons. Artificial neural network is applied on the similar data set, which has been used in fuzzy logic approach discussed earlier. The input values are normalized using min-max normalization, which does a linear conversion of original input set (Han and Kamber, 2001). If, Min_A and Max_A are minimum and maximum values of a dependent factor in data which are used for the study. Then, the formula given in equation (3.10) is used to map X value of A to X'as decimal value within the range of -1 to 1. Therefore, after normalization X' will belong to [-1, 1]. Output values are again mapped to the actual output to get the practical and understandable value of the faults.

Let us take Min_A and Max_A as the minimum and maximum values of A. Then the linear transformation will map the value v to v' in the target range -1 (min_{target}) to 1 (max_{target}). The formula given in equation (3.10) is used for linear transformation.

$$v'' = \left(Max_{target} - Min_{target}\right) * \left(\frac{v - Min_A}{Max_A - Min_A}\right) + Min_{target} \qquad (3.10)$$

Multilayer feed forward network model is used for modeling. Every join of the neuron hidden layer is connected to input nodes. Input nodes are not straightforwardly connected to neuron output nodes. In turn, it proves that ANN does not contain any shortcut neuron connection. As a characteristic property of ANN, it adjusts the weights to

adapt the actual outputs, and difference between the desired and actual output is minimized.

The metrics values have been captured during the development and maintenance phase using the IBM Rational Rose tool. Therefore, a large data set of three metrics values are captured for stated projects. After filtering and analyzing for any outlier, the data set compressed to 6086 data set entries from project P_1 and 1620 entries from project P_2. The data set of project P_1 is divided into two parts for training and testing purpose. 4869 records from P_1 data set have been used for training and 1217 records are used for validation in ANN approach. From project P_2, total numbers of 1620 records are used for testing the network which was trained using total 4869 records of project P_1. Then, ANN model is developed using training data set. MATLAB is used for simulation of the training. Then validation data set is applied on the trained ANN model to validate and check the accuracy of the developed model. Fig. 3.7 shows the network architecture which is empirically determined, and ANN architecture attribute is shown in Table 3.3.

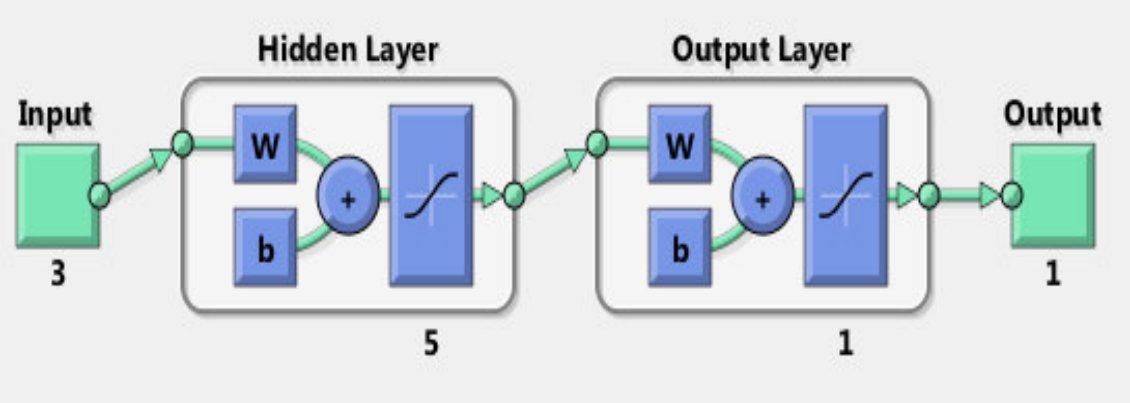


Fig. 3.7: A FFBP ANN Architecture with 5 Neurons

The neural network with architecture as shown in Table 3.3 is trained using the data set obtained from project P_1 and P_2. It is validated through feed forward back propagation with algorithm *trainlm* and *trainbr*, then simulated using MATLAB neural network toolbox. *trainlm* uses Levenberg-Marquardt optimization in neural network training function that modifies weight and bias values. *trainlm* is frequently used supervised algorithm but it needs more memory than other existing algorithms. The input

metrics to the ANN are pre release defects, complexity and lines of count, and the output is defect density of a file.

Table 3.3: Attributes of ANN Architecture

Network Type	Feed-Forward Back Propagation
Training Function	*trainlm,trainbr*
Adaptive Learning Function	*learngdm*
Performance Function	RMSE
Number of layers	02
Number of Neurons	5,10,15,20
Transfer Function	*tansig*
Layers	2
Input units	3
Output units	1

Fig 3.8 shows the architecture of the proposed ANN model. This architecture is trained using the training records of the data sets; the training performance is shown in Fig. 3.8. The three independent attributes, pre release defects, complexity and lines of count are fed into the ANN model and the defect density is predicted as a dependent derived measure. When the defect density data vectors are used, during the training with performance as shown in Fig 3.9, the model progressed in different training and regression states as shown in Fig. 3.10 and Fig. 3.11, respectively.

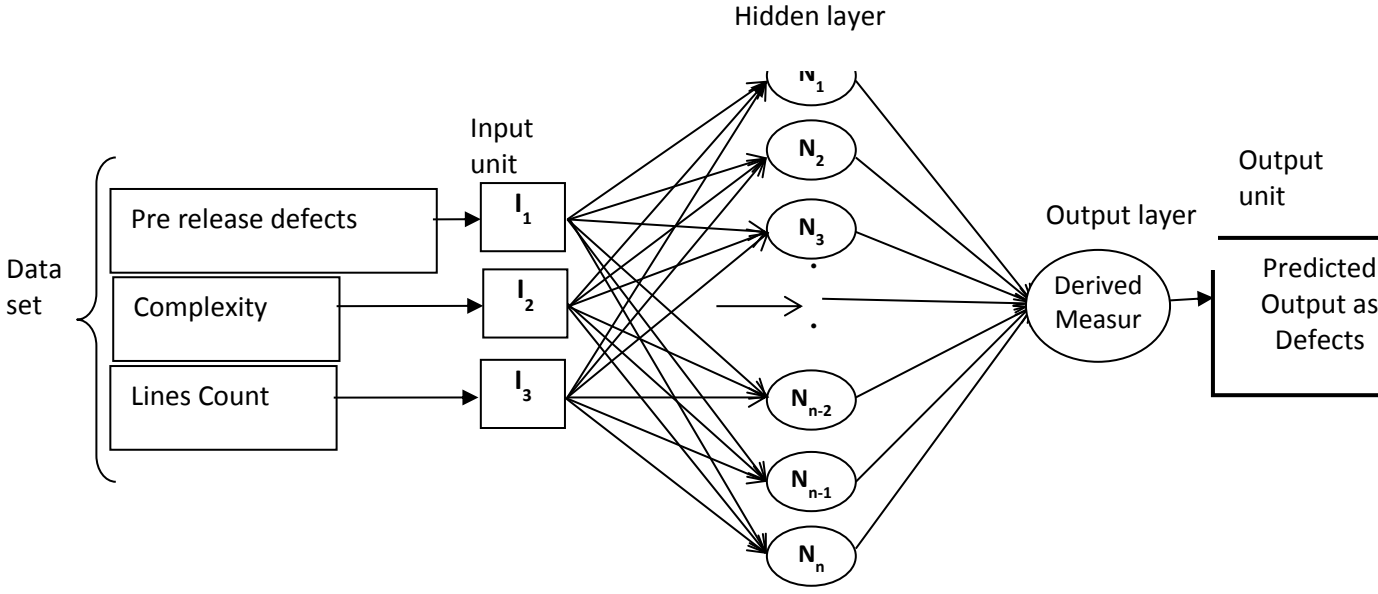


Fig. 3.8: Architecture of Proposed Approach

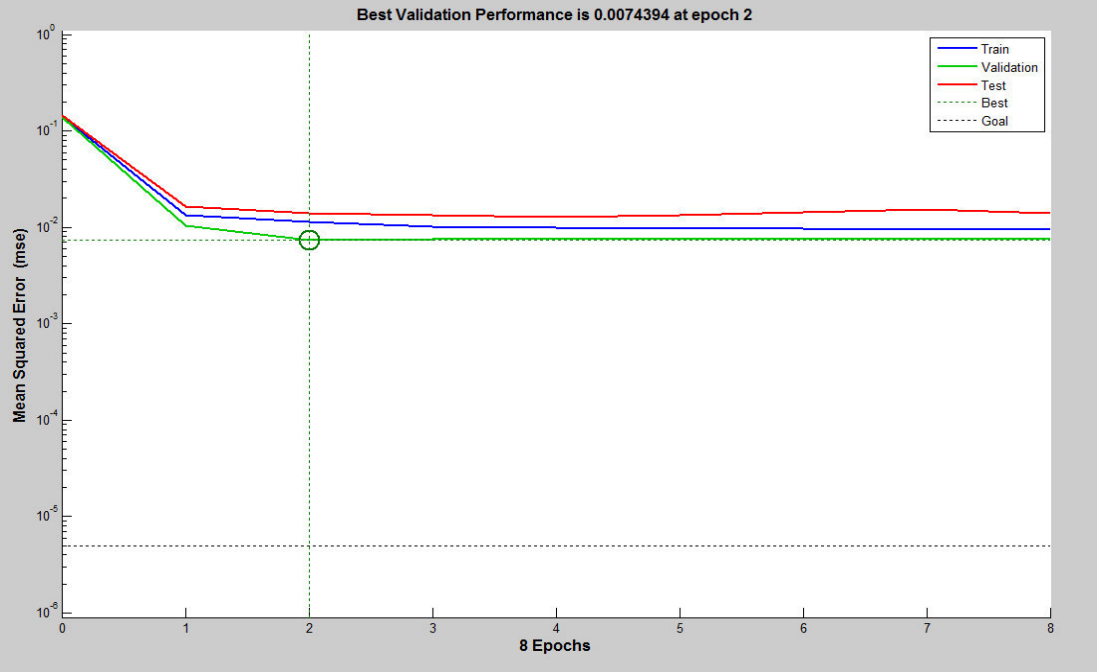


Fig. 3.9: Neural Network Training Performance Using Defect Density Data Vectors

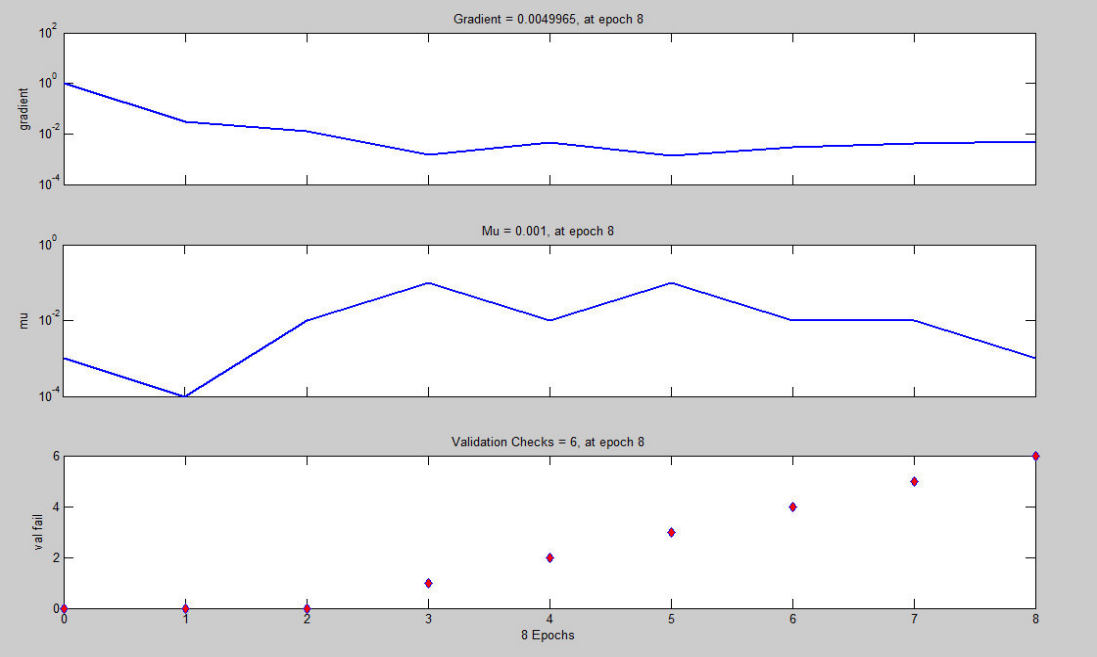


Fig. 3.10: Neural Network Training States Using Defect Density Data Vectors

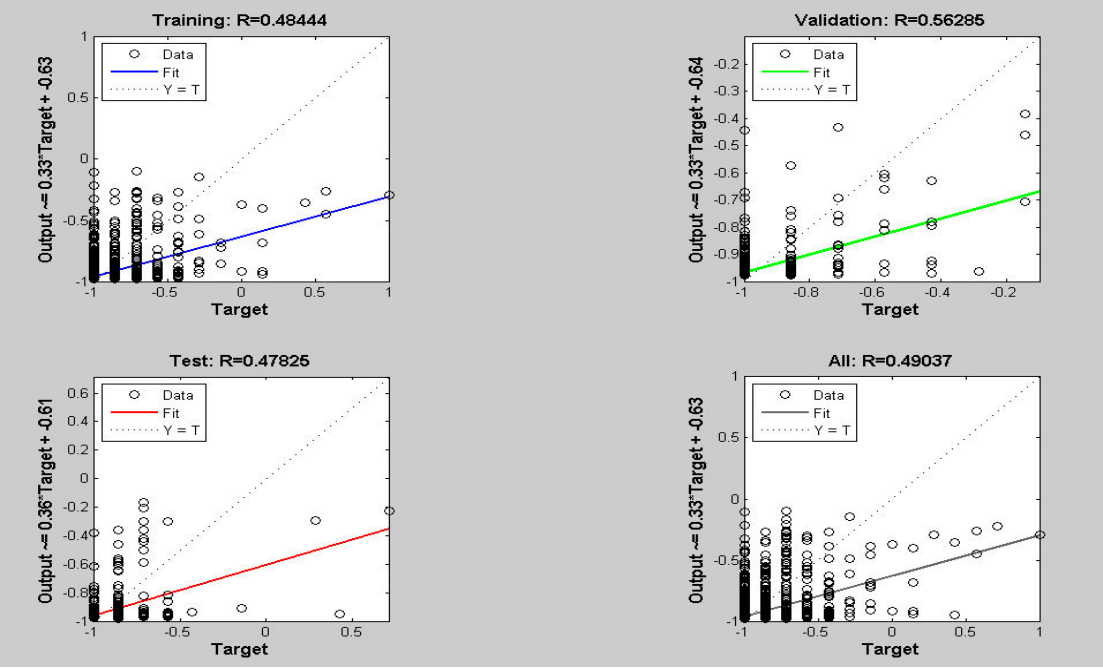


Fig. 3.11: Neural Network Training Regression Using Defect Density Data Vectors

3.4.3.2 Validation of ANN Approach

The validation of the proposed formulation and experiment is done by taking the real project data from project P_1 and P_2. Proposed ANN based defect prediction metric has been applied on the dataset, the accuracy and RMSE are calculated. Total numbers of 1217 and 1620 data records are used for validation of ANN approach and total 4869 records for ANN training. The accuracy and RMSE by proposed approach is shown in Table 3.5 and Table 3.7.

Table 3.5 and 3.7 explain the empirical evaluation of the variance of the actual defects and predicted defect density using ANN. This approach is proficient with RMSE 0.1595 for project P_1 and 0.1916 for project P_2. We captured the output of the ANN approach and found the root mean square error with the actual and predicted defects in the file.

Table 3.4: ANN Architecture Attributes for P_1 Validation

Training Data	4869 training data vectors from P_1
Testing Data	1217 test data vectors from P_1
Network Type	Feed-Forward Back Propagation
Training Function	*trainlm,trainbr*
Adaptive Learning Function	*learngdm*
Performance Function	RMSE
Number of layers	02
Number of Neurons	5,10,15,20
Transfer Function	*tansig*
Error	0.00005
Epochs	Maximum 1000

Table 3.5: Validation Results Project P_1 using ANN Approach

Project P_1 ANN validation results RMSE		
	Network Algorithm	
Network Neurons	*trainlm*	*trainbr*
5	0.1819	0.1773
10	**0.1595**	0.1673
15	0.1675	0.1756
20	0.1836	0.1782

Table 3.6: ANN Architecture Attributes for P_2 Validation

Training Data	4869 training data vectors from P_1
Testing Data	1620 test data vectors from P_2
Network Type	Feed-Forward Back Propagation
Training Function	*trainlm,trainbr*
Adaptive Learning Function	*learngdm*
Performance Function	RMSE
Number of layers	02
Number of Neurons	5,10,15,20
Transfer Function	*tansig*
Error	0.00005
Epochs	Maximum 1000

Table 3.7: Validation Results of Project P_2 Using ANN

Project P_2 ANN validation results RMSE		
	Network Algorithm	
Network Neurons	*trainlm*	*trainbr*
5	0.1930	0.1929
10	**0.1916**	0.1917
15	0.2008	0.2041
20	0.1923	0.1927

3.4.4 Comparative Analysis of FIS and ANN Approaches

Software Defects are main indicators of software quality. To design and formulate the defect metrics, software defect is considered as dependent variable of three software metrics, these are easily available during SDLC, which increases the practicality, and applicability of the proposed approaches. The FIS engines are built to decide defect density of a file, which can be aggregated for defect density at subsystem level of a product. The outcome of the proposed research on metrics design shows that software industries may use FIS and ANN approaches to predict the defect of software release files based on the data availability. This can help to maintain the better software product quality and resource management across releases.

The Table 3.8 shows the best case results of both the approaches. This research presents the investigation using FIS and ANN as well as comparative study of the two approaches.

Table 3.8: Comparisons of Validation Results Using FIS and ANN

Project P_1 and P_2 FIS and ANN results		
Approach	FIS	ANN
P_1	0.2860	**0.1595**
P_2	0.2988	0.1916

Although, ANN shows good results and stands out in case of defect density prediction, but FIS can also be useful in case of partial data or no data. Therefore, if the data set is available and metrics values have been captured during the development process, ANN can be used for better results and in case of less data availability fuzzy logic is also useful tool for prediction of defect density.

3.5 Conclusion

This chapter describes various existing prediction methods developed by researchers to predict the metrics attributes. Research investigations have been carried out to predict defect density using various statistical methods and approaches. Either soft computing has not been completely explored for defect metric or there is lack of validation and comparative analysis. Therefore, our investigation and metrics design show that soft computing provides some good indicator for software quality. The better technique is selected on the basis of comparison of the validation results. Here, the results of the practice shows that ANN stands better than FIS approach to predict the defected bundle of files. RMSE calculations yield to indicate that ANN is better in case of accuracy. However, it is worthy to mention here that FIS is also a good method if there is no project data or partial data. It is not always possible to have a data collection to train the neural network. Hence, in practicality, there are pros and cons of both the approaches. Therefore, software industries can use anyone or combination of FIS and ANN approaches based on their data collection and strategies. The study indicates that now industries should accept the usefulness of soft computing, and include this in general practice during the development and maintenance process. The combination of FIS and ANN, called neuro-fuzzy may yield to better results, hence, in future there is a need to explore neuro-fuzzy, fuzzy-genetic and other evolutionary approaches for prediction of defect density and other quality attributes.

Chapter 4

Reliability Management Process Framework for Software Releases Using Defects Prediction

4.1 Introduction

Software defects are main indicator of software reliability and play an important role to define the software product and release quality. In general scenarios, software defects tend to decrease as subsequent releases are delivered during the software development but not always. A complex and large software development generally contains hundreds of software releases during the development and maintenance cycle. In software industries, there is a strong need of a software process framework which helps to manage the defects across the releases and decide whether the release is ready to deploy in the field or not.

To produce the final system with good quality, the prediction of software reliability during the subsequent releases of the software system is crucial. Remaining defects are a

The major findings of this Chapter have been published in *COMPUTING AND INFORMATICS,* North America, Vol. 32, Issue 1, 2013, pp: 203-224. (*SCI Indexed)*

direct measure to estimate reliability of the software products under development phase and for the developed software as well. We proposed a new practical approach for reliability management of the intermediate software releases in a software product development process. The present study gives an option to predict the software reliability in-between the software development process, which can work as an input to the next product release and prompt the chance to improve the software reliability of the final product. The aim of the proposed approach is to formulate, design, evaluate and validate software reliability and quality management framework by mapping the defect density data of subsequent software product releases. The proposed approach is validated against data set of various releases of two commercial software products. The proposed model can be used for the quality management across the intermediate release and improve the quality and reliability of every next release or current release.

4.2 Quality and Defects

Software defects are the main constitute of quality and reliability. According to IEEE standard 982.2 (1988) and Pfleeger (1992), defect is an anomaly in any intermediate or final software product resulting from an error or fault, ranging from an incorrectly specified set of test data to an incorrect entry in user documentation. Failure is an inability of a functional unit of the system depending on the software to perform its required function, or to perform the function within required limits. A defect may not be uncovered for a long period of time in some situations. When a particular code or logic of software is executed where the defect exists, it may be uncovered during the execution leg of that particular software module depend upon the special condition, it can cause single or multiple failure at any point of time (Kan, 2003).

Paul *et al.* (2000) conducted reliability analysis based on defects for mission-critical software. The authors provided a quantitative analysis of the possibility that a program is not having specific types of faults. The approach to assess the quality was developed in terms of defects by taking the data from multiple projects. The research indicated that it is worthwhile to do defect-based reliability prediction for mission critical systems. The authors used regression analysis and clustering method for defect prediction. The regression data contains testing process metrics and program metrics. Bai

et al. (2008) proposed the model to estimate the remaining defect in a software system. The study is conducted on the basis of remaining software defect estimation curve. The curve is useful to show the dynamic behavior of the remaining defects as testing proceeds for the software product.

Based on the strong evidences of relationship mapping of quality and reliability to the defects, in the proposed approach, we have developed a concept to manage the quality and reliability of a product using defect prediction. In this chapter, the framework has been implemented and validated across the multiple releases of software product. We have used this concept across the multiple releases of a software product and developed a unique and new process which can be applied during the development of various software releases. The proposed approach can be automated and made an integral part of the software management process.

4.3 Related Work in Quality and Reliability Analysis Using Defects

The researchers have tried to define the trend of defects in a software system. A study shows that defects are in line with the rayleigh curve, which is directly related to the project staffing (Card, 2002; Huang *et al*., 1997). McConnell *et al*. (1996) tried to establish relationship of defect rate with development time and concluded that the software projects which are having the lesser defects also have shortest development time or schedule. As per analysis by Cai (1998), the defects after release need to be estimated as these will give the idea about the quality and reliability of customer release. Most of the previous work focused on the prediction of the initial software defects which may not be able to give actual picture of the post release defect considering only the single attribute. So, there is a need of some more input metrics combination which influence the post release defects. Zeng *et al.* (2009) analyzed the defect patterns to define the reliability criteria. They concluded that currently software industries are putting more efforts to collect the defect data for analysis and reliability design, therefore, defect data can be used to avoid the similar defects and improve the overall developed software quality.

Reliability is the key attribute to define the software quality. David (2002) concluded that estimating the defect growth is a better technique to evaluate the software

quality. Syed-Mohamad and McBride (2008) compared the defect arrival rate in open source and in-house developed software projects. In a recent and important study, Jingyue *et al.* (2012) presented a study that, in general, the data collected during the development process in industries is either inconsistent or difficult to apply in software quality assurance. Their analysis focused on revising the attributes values and also introducing some new attributes in data collection. It was found that the proposed process improvements reduce the fault density and also increase the bug fixing efficiency of the remaining defects.

Literature dictates that there are many models proposed to predict the reliability growth of software product during the last three decades (Zhang and Pham, 2006; Huang and Lin, 2005; Huang, 2006; Hu *et al.,* 2007; Jeske and Zhang, 2005). Goel and Okumoto (1979) formulated and validated the non-homogenous prose model. They tested it and tried to fit the data in the model.

In last ten years, researchers have started exploring the soft computing approaches to solve uncertainty problems in reliability modeling and prediction. Madsen *et al.* (2005) used fuzzy logic for software reliability. They studied the soft computing approach for fault analysis and optimization of reliability. They concluded that application of soft computing approaches such as fuzzy logic, evolutionary computing produce interesting results and there is a need to explore soft computing techniques for reliability prediction and engineering.

Yuan and Zhang (2011) presented the study of reliability prediction based on fuzzy neural system. The results from this approach were better than other. Jianhong *et al.* (2010) presented ANN based techniques to predict the severity of faults in a software system and the data used from function based system. Using ANN, they are able to identify the area of the major faults which need immediate attention for quality and reliability improvements, although the RMSE was higher in the predicted results. Considering the good footprints of FIS and ANN in software attributes analysis, reliability management framework is built using the soft computing approaches.

4.4 Quality and Reliability Management in Software Intermediate Releases

There are plenty of research contributions to establish the relationship between the defects with reliability and quality. Cangussu *et al.* (2004) give the concept of software release quality control based on software defects. They presented the two case studies, prediction of residual faults and the efficiency of the test phase based on the revised checkpoints taking input as predicted faults. Their prediction can help the managers for the resource distribution and also inputs for revision of the cost for maintenance and testing. Similar concept is being used in the coming section to design the reliability framework.

4.4.1 FIS and ANN Approaches to Predict the Quality and Reliability of Releases

In literature analysis, it is observed that there are different reliability growth models to define the reliability at any point of time in software development (Musa *et al.*, 1987). The focus was to develop a framework which is easy to use, more applicable and practical. Therefore, the simple model is used as indicated in Fig. 4.1. The graph shows the faults discovered at any time t where every testing cycle is finished, and the faults are available which are found during the testing.

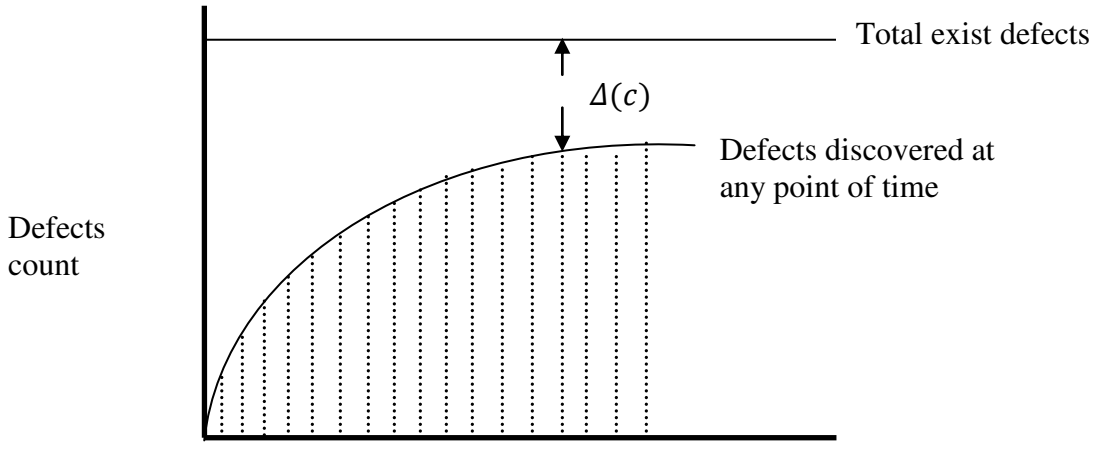


Fig. 4.1: Reliability Growth Model in Terms of Defects

Here, $\Delta(c)$ is the difference between the detected faults during the testing phase and the total defects exist in the release or product. All efforts by researchers have been made to define the reliability growth at product level. We used the G-O model given by Goel and Okumoto (1979) as a base which is the simplest and mostly used reliability growth model in industries. Goel-Okumoto (G-O) model can be defined as per equation (4.1).

$$\delta(t) = A * (1 - e^{-bt}), \qquad (4.1)$$

here, $\delta(t)$ are the total defect discovered at point of time t during the testing and development, A are the total defects existing in the module or software system, b is a constant.

For most of the reliability growth models (Stieber, 2007), mainly the factor $(1 - e^{-bt})$ differs in terms of addition or change of dependent variables. Hence, in general case, in the existing models the equation comes out as shown in equation (4.2).

$$\delta(\mathrm{t}) = A * f(t), \qquad (4.2)$$

here, $f(t)$ is the cumulative distribution function. In simple case, it is $(1 - e^{-bt})$ as per defined curve in Fig. 4.1. Therefore, when the release is given to testing and integration team, A will be 0, hence, the equation becomes as shown in equation (4.3).

$$\delta(t) = A * f(t) = 0 * f(t) = 0 \qquad (4.3)$$

At infinite (assumed to be long period in product life time), i.e., $t \to \infty$ then the function $f(t)$ can be given as

$$f(t) = (1 - \mathrm{e}^{-\mathrm{bt}}) = f(1 - \mathrm{e}^{-\infty}) = (1 - 0) = 1$$

Therefore, if we are able to predict the defect using a techniques and denote the total defects A as $p(d)$, then,

$$\delta(t) = A * f(t) = p(d) * 1 = p(d),$$

here, $p(d)$ are the total predicted defects which have not been discovered yet in testing. The value of $p(d)$ may differ from the actual defects values as shown in equation (4.4).

$$p(d) = (A \pm m), \qquad (4.4)$$

here, m is some constant value, which differ from the actual value A. Hence, after a long testing, execution and field operation, the system will reach the value of $\delta(t)$ near to $p(d)$ because during the long time system sustenance most of the faults have been discovered and fixed in the maintenance releases. It means that all faults are discovered and the reliability is also reached at maximum level.

Consider the equation (4.1) for defining the reliability at any fixed time (after 1st phase of testing) in the curve Fig. 4.1. In terms of remaining undiscovered defects in the system or module, the reliability at any time t can be obtained using equation (4.5).

$$\Delta(c) = p(d) - \delta(t), \qquad (4.5)$$

where, $\Delta(c)$ denotes the difference between the predicted faults ($p(d)$) and total discovered ($\delta(t)$) at any time t. The total number of predicted faults ($p(d)$) in equation (4.5) is a hypothetical value which cannot be exactly defined for any system or release. It has to be predicted on the basis of some independent attributes using a conventional or unconventional technique. Then, if we denote the predicted defects by the function $p(d)$, the equation (4.1) becomes as given in equation (4.6). For defining the reliability at any fixed time t in the reliability curve the equation (4.6) has been considered.

$$\delta(\mathrm{t}) = p(d) * (1 - e^{-bt}) \qquad (4.6)$$

Considering a release R_1 of project P_1, $P_{R1}(\mathrm{d}, m_1, \mathrm{t})$ is the value of predicted defects after release testing at time t, where, the faults are being discovered. Here m_1 is a particular module of the release R_1 of Project P_1.

Software industries collect the testing and design data during the development of a software release. The data gathered during the design and initial development can help a lot during the quality assurance process. Hence, in the proposed research, the metric is designed using the independent attributes values from gathered data to help in release quality control process. To design and formulate the defect metric based reliability metrics, software defect is considered as dependent variable of three software metrics. These are easily available during SDLC, which increases the practicality, and applicability of the proposed approaches. These three attributes, namely, complexity, total lines of code and defects before release (discovered at early stage during the testing and integration phase) are taken into account to decide reliability factor. The FIS engine and ANN learning algorithm are built to decide the expected count of the defects, which can be aggregated for defects at subsystem level of a product, and hence total defects as well as reliability of software release.

Let x_n, y_n, z_n be the values of the complexity, total lines of code and defects before release respectively then

$$P_{R1}(d, m_1, t) = f_m(x_n, y_n, z_n), \qquad (4.7)$$

where, f_m is the function of the x_n, y_n, z_n, which are value of the independent attribute in a file or module. $P_{R1}(\mathrm{d}, m_1, \mathrm{t})$ is the predicted defect count in the module m_1 of release $\mathrm{R_l}$ at any point of time t. The total defects in the release or system at time t can be calculated using the equation (4.8).

$$p(d) = \sum_{i=1}^{n} P_{R1}(d, m_i, t) \qquad (4.8)$$

The data were gathered from the two large commercial projects for various releases as considered in last chapter also. In case of fuzzy logic, FIS engine is built in

MATLAB software tool for defect prediction. The three attributes are given as an input for the one particular release data set. This step will predict the expected remaining defects in the release after a particular integration and testing phase. The predicted-defects are used to find the dense areas of modules where there are more defects expected after release delivery.

Therefore, if the data set is available and metrics values have been captured during the development process, FIS may be used for better results and also it can be used in case of less data availability. A fuzzy set S in X can be defined as a set of ordered pairs

$$A = \{(x, \mu S(x)) | x \in X\}, \qquad (4.9)$$

where, $\mu S(x)$ is called MF for the fuzzy set S. A MF maps all elements of X to a membership value between universe of discourse 0 and 1. Hence, the equation (4.9) is an obvious extension of classical definition in which the defined function is allowed to have a value between 0 and 1. In the experimental design $P_{R1}(d, m_1, t)$ is considered as fuzzified and fuzzy rule function. The predicted values of faults can be calculated as per the following equation (4.10).

$$P_{R1}(d, m_1, t) = \{(x, y, z, \mu S(x_{m1}, y_{m1}, z_{m1})) | x_{m1} \in X, y_{m1} \in Y, z_{m1} \in Z\} \quad (4.10)$$

Fuzzy inference toolbox of MATLAB is used to implement this relation and the results for each release are shown in next section. In case of ANN approach $P_{R1}(d, m_1, t)$ is implemented as ANN learning algorithmic function. The defect density of a module using ANN function can be derived as:

$$P_{R1}(d, m_1, t) = O = f(net) = f\left(\sum_{i=1}^{m} w_{a_i} x_{m1_i},\ w_{b_i} y_{m1_i}, w_{c_i} z_{m1_i}\right), \quad (4.11)$$

where, w_{a_i}, w_{b_i}, w_{c_i} are the weight vectors for input vectors from module $m1$ as complexity (x_{m1_i}), total lines of code (y_{m1_i}) and pre release defects (z_{m1_i}), respectively.

In the implementation of the above relation, *nntool* of MATLAB is used. The results of the *nntool* and RMSE are shown in next section.

For both the implementation, the defects count of release R_1, taking all the modules' predicted defects after the first phase of testing, will be as shown in equation (4.12).

$$p(d) = \sum_{i=1}^{n} P_{R1}(d, m_i, t), \qquad (4.12)$$

where, n is the count of total module in system and i is a particular module. The same process has to be repeated for every cycle of system integration testing. There is an inversely proportional relationship of software quality and reliability with the software faults. Software defect is a most important factor to estimate the quality and reliability (Bai *et al.*, 2008). Hence, we used the proposed FIS and ANN approaches to predict the defects as described in Chapter 3. In this chapter, the fundamental concept remains the same but the techniques are applied across the multiple releases of the software data base and built the project planning and management concept.

Reliability is an important attribute to define the software quality. Software reliability and quality are inversely proportional to the defect density in a software release. The software quality percentage for a release can be formulated as below:

$$SQ_m \, \alpha \left(\frac{1}{p(d)_{m_i}} * 100 \right), \qquad (4.13)$$

where, SQ_m is the cumulative percentage quality of a module m_i of the software system release at any point of time t and $i = 1, 2, \ldots, n$. Here, n is the count of total module in system. $p(d)_{m_i}$ $(i = 1, 2, \ldots, n)$ are the predicted defects in module m_i at time t.

Since, in the gathered data set, we did not get the defects data based on the different time interval during the testing phase, hence, the calculated RF is used to decide the maturity of current release state. Reliability factor is deciding factor for the reliability in software quality assurance. The formula to calculated RF is given in equation (4.14).

$$RF = \frac{Remaining\ faults}{Total\ expected\ faults} = \frac{(Total\ Faults - discovered\ fault)}{Total\ expected\ faults} \qquad (4.14)$$

The value of RF varies from 0 to 1. If, RF value of any software release or product is near to 0 then it is considered to be more reliable for customer delivery or field deployment. The calculated results are shown in next section, which are based on the real project data.

Here, in this chapter, the new approach has been presented to predict the reliability growth at module and release level granularity. This level of granularity is important because the end goal is not only to define the defect based reliability growth model but also give some insight information to the software quality assurance (SQA) team and developers.

In the last chapter, the concept and validation for defect density were discussed but for any software product, it did not contain the multiple release framework and validation. To define the software quality assurance framework and process improvements, the defect density prediction is validated and extended across the multiple releases of software projects. The reliability factor is calculated to define the maturity level and to analyze the remaining defects in the current and future release.

4.4.2 FIS and ANN Approaches to Design the Reliability Management Framework

The FIS and the function $P_{R1}(\mathrm{d}, m_i, \mathrm{t})$ is implemented using the MATLAB fuzzy toolbox. For complexity let X be the universe of discourse and x_n be the value in the fuzzy boundaries [0,1]. Then equations (4.7) and (4.8) are implemented by applying fuzzy inference rules for the function $P_{R1}(\mathrm{d}, m_i, \mathrm{t})$. The MF is used to map the actual attributes values into the fuzzy range. Similarly, for line count and pre release defects, the triangular MF is used for fuzzification of the crisp values.

The proposed fuzzy logic based model considers all three factors as inputs and provides a crisp value of defect density using the rule base. All inputs can be classified into fuzzy sets *viz*. Low, Medium and High. The output defect density is classified as High, Medium, and Low. All possible combinations i.e. 27 of inputs are considered to

design the rule base. Each rule corresponds to one of the three outputs based on the expert opinions. These rules are defined to implement the $P_{R1}(\mathrm{d}, m_i, \mathrm{t})$ given in the equation (4.7).

All 27 rules are inserted into the proposed model and a rule base is created. Depending on a particular set of input, a rule is fired. Using the rule viewer, output i.e. defect density is observed for a particular set of inputs using the MATLAB fuzzy toolbox.

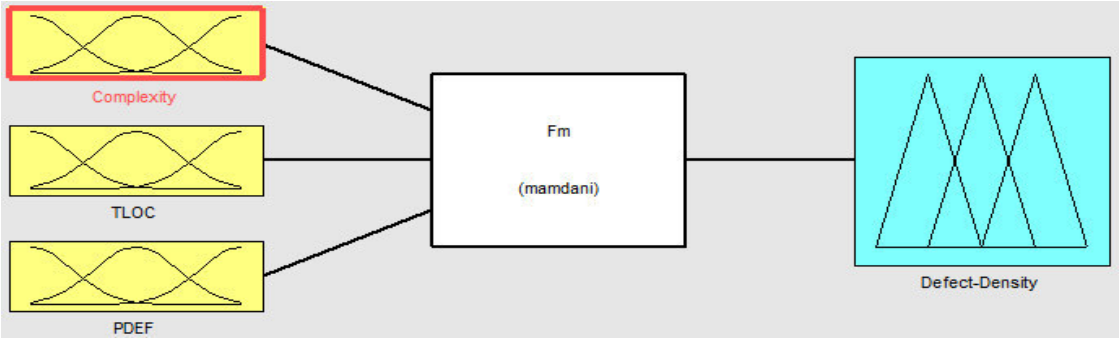


Fig. 4.2: Fuzzification of Inputs into Output (Defect Density)

There are various ANN architecture and algorithm for application in prediction of software entity. *trainlm* and *trainbr* learning algorithms are used in implementation of the ANN approach for reliability factor analysis and prediction. The combination of 5, 10, 15, 20 and 25 ANN neurons are used with these algorithms. The calculated software quality and reliability index will be given as input to redefine the software quality assurance and software process improvements.

The ANN architecture is trained using the training records of the data sets; the training performance is shown in Fig. 4.3. The three independent attributes, pre release defects, complexity and lines of count are fed into the ANN model and the derived measure defect density is predicted. When the defect density data vectors are used, during the training, the model progresses in different regression states as shown in Fig. 4.4.

ANN accuracy heavily depends upon the training performances and states during the training process of ANN architecture. The data behaviors and patterns are responsible for correct and efficient learning. The training yields to good testing performance as well, which covers the different combination of the input-output vector.

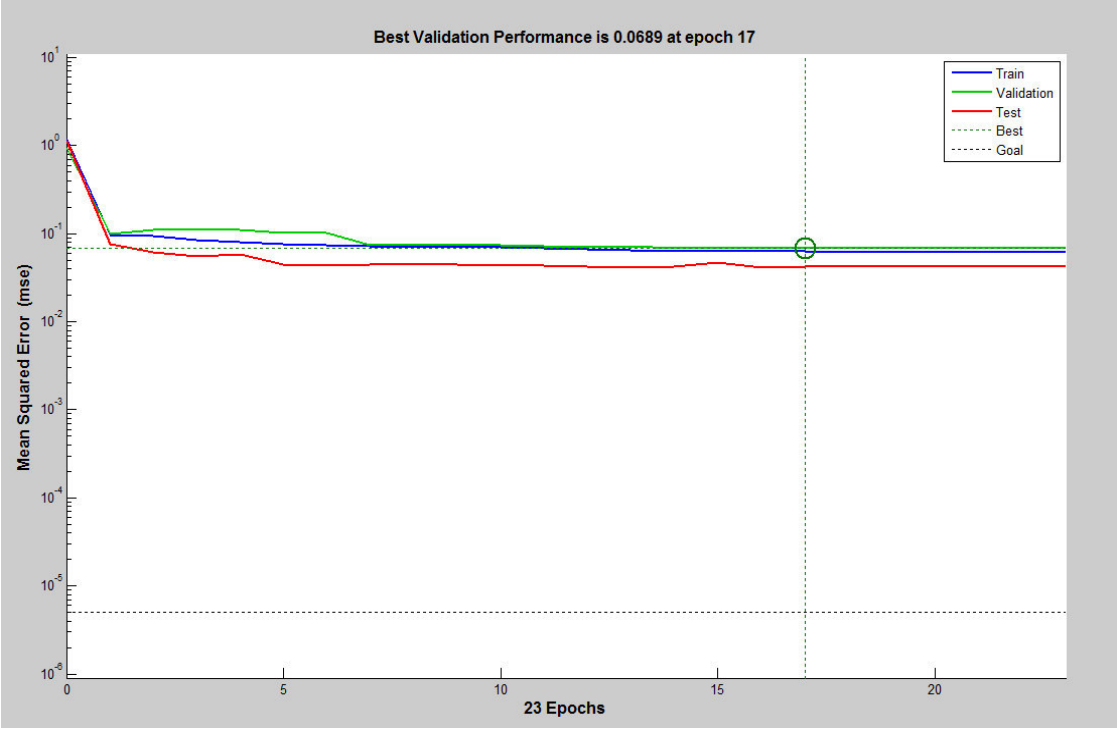


Fig. 4.3: ANN Training Performance with 5 Neurons and *trainlm* Using Reliability Data

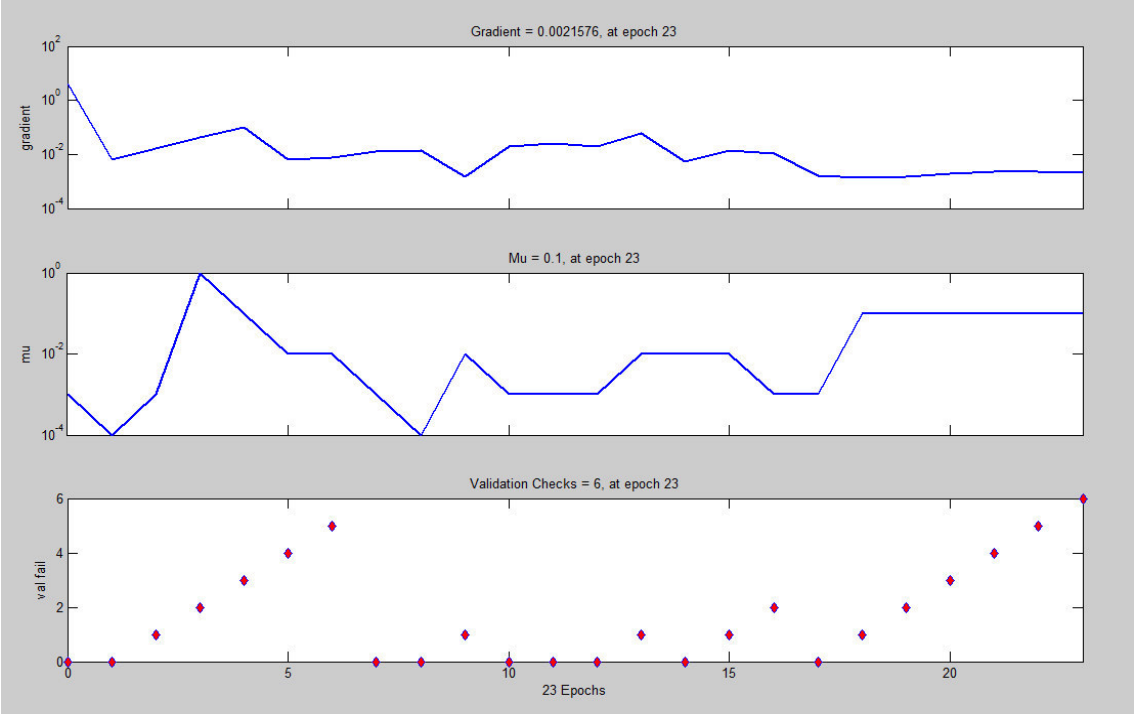


Fig. 4.4: ANN Training States with 5 Neurons and *trainlm* Using Reliability Data

4.4.3 Generic Reliability and Quality Management Framework Across the Releases of Project

In every software project, there is a SQA team which defines the specific threshold of different attributes for the particular project based on the agreement with customer. The most important factor which SQA team defines is the maximum allowed defect count to get ready the customer release after testing and integration phase is over. The data of the discovered faults will be available during the testing time interval. But, it is well known fact that there will be some more remaining bugs in the software modules and system. So, there has to be some way to predict them to get better control on software quality and reliability.

Also, there is an option to make revised checklist in the release system and integration testing based on the input from the defects based release quality management model. Another side of practicality of this model is to evaluate and enhance the test cases efficiency to put more emphasis on most defected area in the release, either before the next release or during the next release development.

The proposed process/algorithm of release reliability management involves the following steps:

- Step-1: Collect the independent data for input variables/attributes for the first release R_1 of the software system/product.
- Step-2: Apply the proposed fuzzy logic or ANN approach to predict the defect density for the next release R_2.
- Step-3: Estimate the quality of next release R_2 and hence reliability based on defect density prediction in step 2.
- Step-4: Check for the threshold level of the allowed and agreed defect count, if the predicted effects are less than the threshold, shipment of release is ready.
- Step-5: Resource planning input is provided to the SQA management team and for software process improvements of release R_2, apply the resource distribution and focus on the area which is responsible for less reliable software.
- Step-6: repeat from step 1 for release R_2.

Hence, in every software release, the defect value is predicted using the presented approach and used for the next release. In this process, the reliability growth model is formed as we progress further to the subsequent releases. Hence, the value of $\Delta(c)$ as defined in equation (4.5) can be calculated. The SQA goal is to minimize the value of $\Delta(c)$ *f*or every release.

In Fig 4.5, the state diagram shows the implementation of the proposed approaches in SQA framework and software process improvements (SPI). If, we start the process of defect prediction from release R_1 of project P_1, the defect prediction can be calculated in each release and used in next release of software product. After predicting the defects in a particular release by using proposed approach, an efficient resource distribution and effort planning can help to maintain the product quality and reliability.

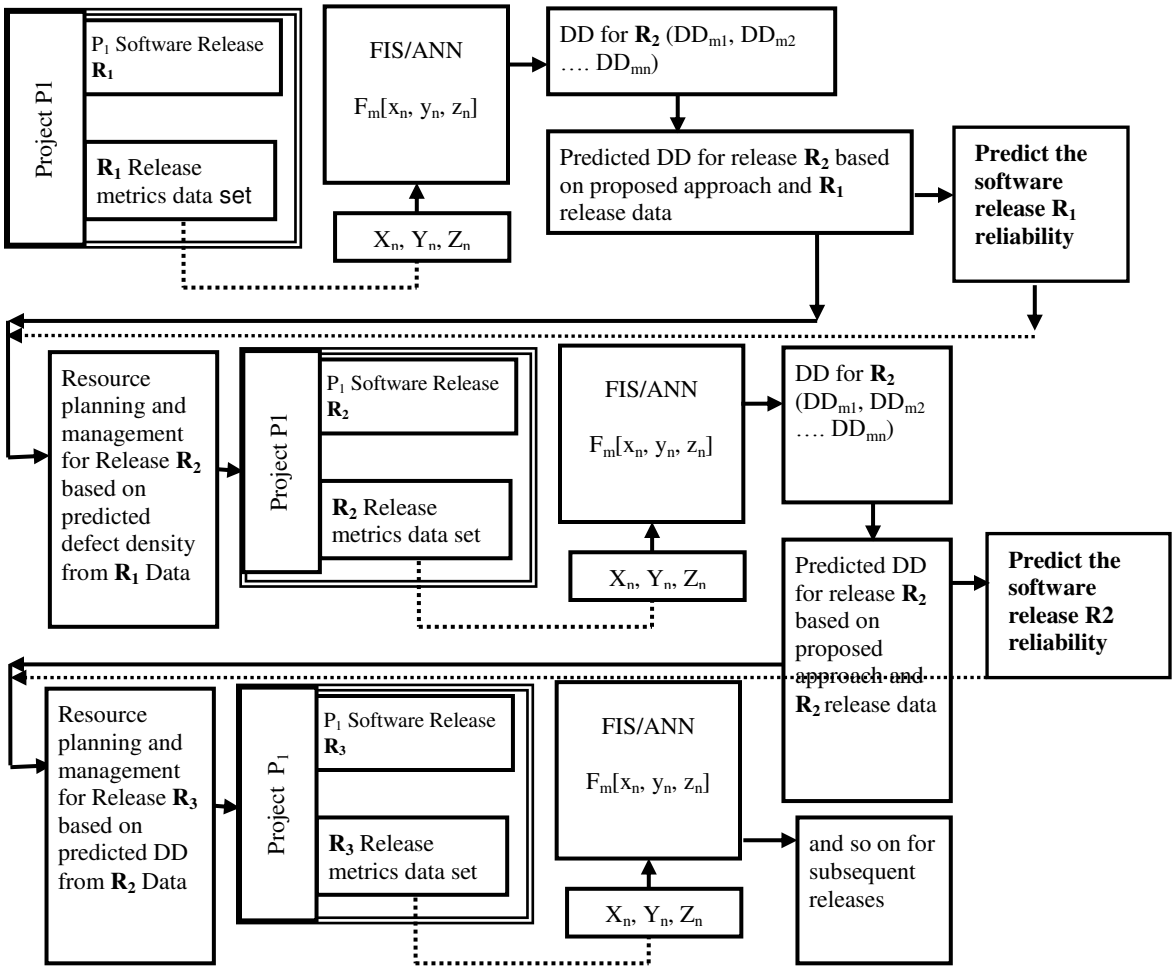


Fig. 4.5: Proposed Release Quality and Reliability Control Framework Across Releases of Project P_1 For Process Improvement

The next release planning will be done for the following management factors:

- Next release schedule forecasting based on the complexity and intensity of the predicted defect density in a module.
- Resource planning and critical resource movement towards the most defected area or module.
- Cost estimation based on the defects intensity.
- Also, there may be some modules in which there are fewer defects predicted using the proposed approach but it is highly complex, the schedule and cost shall be revised on the basis of this input factor.

4.4.4 Validation in Multiple Releases

Using the FIS and ANN approach, we are able to predict the defects at file at module level, so it provides the total defects attribute to define the reliability growth model and also the information about the area in a module where most of the defects lie. The data gathered for three software releases of two projects are being used for validation and implementation of the FIS and ANN techniques.

4.4.4.1 Validation of FIS Approach

FIS based approach is validated by calculating RMSE for three random data sets of 100 modules from project P_1. The same data set has been used as in Chapter 3, but in this chapter, the data is captured for 3 releases of project P_1. The following were the RMSE results for the project P_1 taking 3 releases.

Table 4.1: Validation Results of Defect Metric Using FIS

Project P_1	RMSE with FIS approach		
	R_1	R_2	R_3
Data set:1	0.2860	0.2967	0.3238
Data set:2	0.3127	0.3022	0.3296
Data set:3	0.2768	0.2853	0.2899

4.4.4.2 Validation of ANN Approach

In Chapter 3, it is observed that *trainlm* provides the best results with 10 neurons. But to find the best performance, ANN is trained using *trainlm* and *trainbr* algorithm and also validated using release data. 632 records from Project P_1: R_1 are used for ANN training. Then, 514, 479, 458 records are used for validation from release R_1, R_2, R_3 respectively of project P_1. The Table 4.2, Table 4.3 and Table 4.4 show the results of RMSE for three releases of project P_1.

Table 4.2: RMSE Results for R_1 Using ANN

Project P_1, R_1 ANN validation results RMSE		
	Network Algorithm	
Network Neurons	*trainlm*	*trainbr*
5	0.1951	0.1737
10	0.2122	**0.1734**
15	0.1941	0.1734
20	0.1854	0.1766
25	0.2403	0.1766

Table 4.3: RMSE results for R_2 Using ANN

Project P_1, R_2 ANN validation results RMSE		
	Network Algorithm	
Network Neurons	*trainlm*	*trainbr*
5	0.2315	0.2148
10	0.2184	**0.2144**
15	0.2481	0.2144
20	0.2341	0.2182
25	0.2481	0.2181

Table 4.4: RMSE results for R_3 Using ANN

Project P_1, R_3 ANN validation results RMSE		
	Network Algorithm	
Network Neurons	*trainlm*	*trainbr*
5	0.1981	0.1826
10	0.1981	**0.1822**
15	0.1915	0.1822
20	0.1885	0.1831
25	0.2241	0.1831

In the analysis of results, it can be seen that *trainbr* performs better than *trainlm* and giving the less RMSE. RF is calculated for three releases of the project P_1. Only ANN approach is applied as ANN provides better prediction than the FIS approach. Also, the FIS validation is carried out only on 100 selected records of each release. Hence FIS results cannot found the release maturity. The following are the RF results on the basis of actual faults discovered in the release and the predicted values using the proposed ANN approach, which are shown in Table 4.5, Table 4.6 and Table 4.7.

Table 4.5: Reliability Factor Calculation Results for R_1 Using ANN

Project P_1, R_1 actual RF = 0.7017		
Total faults discovered before release = 1788	Total faults discovered after release = 760	Total expected faults = 2548
	Predicted RF with NN algorithm	
Network Neurons	*trainlm*	*trainbr*
5	0.6432	0.6345
10	0.6506	0.6452
15	0.6461	0.6330
20	0.6643	0.6337
25	0.6131	0.6337

Table 4.6: Reliability Factor Calculation Results for R_2 Using ANN

Project P_1, R_2 actual RF = 0.6629		
Total faults discovered before release = 1676	Total faults discovered after release = 852	Total expected faults = 2528
	Predicted RF with NN algorithm	
Network Neurons	*trainlm*	*trainbr*
5	0.6716	0.6680
10	0.6711	0.6661
15	0.6661	0.6662
20	0.6661	0.6677
25	0.6490	0.6677

Table 4.7: Reliability Factor Calculation Results for R_3 Using ANN

Project P_1, R_3 actual RF = 0.6783		
Total faults discovered before release = 1662	Total faults discovered after release = 788	Total expected faults = 2450
	Predicted RF with NN algorithm	
Network Neurons	*trainlm*	*Trainbr*
5	0.6770	0.6658
10	0.6770	0.6646
15	0.6759	0.6646
20	0.6975	0.6664
25	0.6523	0.6664

4.4.5 Practical Application of Management Process Framework

The software reliability can be predicted by estimating the module wise defect density. Some preventive actions can be applied for efficient resource utilization on the major defected area or module. As per human nature, the fuzzy type of values for any attribute are available during the project execution. Although, the proposed model empirically has been evaluated on real project data, one major advantage of FIS approach is that it may also work without the data. After first release or from the similar type of project data set, three attributes values will be ready for the system, therefore, it will be a good defect predictor tool for the next release because generally, a large and long term software projects contains 50 to 100 releases during the development. It has been proved

a good estimator for different domain project. If, the system needs to enhance or add new feature, during the planning of software project, the proposed approach will be able to predict the defect for the modules and files. The proposed approach has a good granularity as it may predict the defect at file level as well as at module or subsystem level which not only reduces the project planning errors and risk but also provide the additional information for resource distribution across the project schedules. Even for a new project initiation, we can use as a defect predictor for various modules. In fact, some rough figures for the three values are available for a new project by taking the reference from the similar type of previous projects. Proposed approach may also be a good evaluator for Commercial of the Shelf (COTS) product. For a COTS product, there will be ready values of three input variables (if collected during the development of components) in advance; hence, the defect can be estimated using the proposed approach, which helps us to evaluate the COTS product reliability. It is obvious that if there are more defect possibility then the system developed using the COTS product will also have more defects. If there are attributes values for any ready COTS product, the defects can be predicted and hence may be used as effort and schedule estimator as well as a reliability indicators. Proposed FIS and ANN methods can also be used for comparative software reliability evaluation of the different projects across multiple releases.

The presented quality management model provides the following two choices for the maintenance or quality managers in industries for a release shipment:

- The approach can predict the most defected area in release R_2 of project P_1 using the R_1 release data set. Therefore, they can take decision to re-schedule the release R_2 and work on defected area to reduce the defects and improve the overall quality and reliability. In this case, the initial planning of the subsequent release should have some buffer time to identify the predicted defects and provide the solution.
- Second option is to use the proposed approach to find the quality level and deliver the release R_2. The predicted dense area can be fixed during the development of release R_2.

4.5 Conclusion

There has been a lots of research investigation to predict software reliability using various statistical methods and approaches by taking defect density as an input. Soft computing techniques have not been completely explored for this area. Therefore, our investigation and metrics design show that soft computing provides some good indicator in software reliability and management. The usages of the data across the release with fuzzy logic are the new concept introduced in this chapter which was almost unexplored earlier. The proposed research can be good tool for software release reliability and quality management and evaluation as well as improvement. We have presented the reliability and quality management process framework on the basis of the defect density in a particular release based on the previous release data. In turn, it works as an input to the software management team to divert the focus and put skilled resources on the most defected (less reliable) module/area. The predicted reliability factor can be calculated and use it to define the maturity level of the release.

The proposed framework answers the following difficult questions for software quality assurance team in industries:

- What is the current state of the reliability in terms of the discovered and predicted remaining defects using FIS/ANN approach?
- How efficient testing is? How much more testing efforts required to achieve the desired defect density threshold for an acceptable customer or field release.
- The resource planning for testing phase as well as scheduling for release.
- As there are several releases of a large software system, therefore, the trend analysis during initial releases can provide a good foundation for planning of future releases.
- Based on the proposed framework, it is good to include the FIS/ANN prediction approach for every release to improve software processes.
- In this chapter, the most important outcome of the analysis is to predict the total defects which are expected to occur after field deployment of the product release, and then either push the testing phase to minimize the difference between discovered and FIS/ANN predicted defects and achieve the desired defects threshold or define the

release reliability and quality based on the inputs. It will set right expectation to the customer.

Chapter 5

Reusability Prediction of Components

5.1 Introduction

Development time and resources utilization are very important aspects in software development industries. There has been a research focus to optimize the development time and resource allocation to achieve the industry goals and objectives. In consideration of this business need, the component based system development has been adopted by the software organizations. The component based system development helps industries to reduce the overall development time as well as optimize the resource utilization. Although, the reusability concept is applied to structure and object oriented software engineering methodologies, it is more applicable and makes sense in case of component based systems. In component based system development, to maximize the reusability of the component, it is necessary to measure reusability to improve and assess the quality of component based systems. Software development is a different type of engineering stream in which complexity is high due to the processes attached to the SDLC. In recent times, there is a trend of large scale complex development of software system as per the

The major findings of this Chapter have been published in International Journal of Computer Applications (IJCA), May, Vol. 70, Issue 15, pp: 41-47.

business requirements. The reuse of the old designs and developed components may help to reduce the complexity of the development as well as reduce the development time and efforts. The basic concept behind the software reuse is to utilize the old developed fact, design and software parts for development of new software system. Reuse is the use of previously acquired facts (concepts) or objects in a new situation. These old developed parts could be anything: design patterns, requirement documents, software components, a process document and a software quality process. The reusability is the part of software design approaches as well as in processes where the developed components play a good amount of role in creating the new software system. The first objective of reuse is to reduce the similar work, project schedule, system development cost and improve the quality of the final system. If a software industry has invested a good amount of time, efforts and cost in development of a component to make the component more reliable with good quality, then the new system using this component will also have good reliability and quality (Pandey and Goyal, 2013; Quyoum *et al*. 2010; Frakes and Tortorella, 2008). The reuse also helps to improve the maintainability of the newly developed system.

Reusability is the degree to which a component can be reused and reduces the software development cost by enabling less coding and more integration (Wang, 2003). In component based system development, software reuse is treated as a reuse without knowing the internal functionality of the component. In CBSD approach, the software developers focus on the interfacing part only rather than the details of the internal implementation. The highly reusable components have clearly defined interfaces, structured documentation of interfaces and also restrictions to access it. If there is any enhancement done during the maintenance of component, then there should not be any change in the component application part and the compilation with linking will solve the purpose. If there is logical change which affects the interfacing also, the user or customer will get impacted (Judith and Audrey, 1998). The classification is a major issue in software components reuse. It is necessary to classify and identify the component from a large set of component set. It is required to keep record of the characterization of component which can include indexing, frequent usage, enhanced retrieval and better understanding (Gill, 2006). As per Basili and Rombach (1988), software reuse has been

termed as a tool to reduce the development cost and time of the software development. The need for software reuse has become urgent because the size and complexity of software have started to escalate very fast. Software reuse is defined as the reuse of everything associated with a software project including knowledge. As per Mili *et al.* (1995), it is the process in which an organization defines a set of systematic operating procedures to specify, produce, classify, retrieve, and adapt software artifacts for the purpose of using them in its development activities.

There are two aspects in component based software engineering. First is to develop the new system using the already developed components. These components should have defined interface to interact with it. This practice will make the newly developed system more reliable as the reused components should have been tested either during the development of component or in the systems where it has been already used. It will enhance the quality of the newly developed system as well as the development time and efforts will be saved. On other hand, the second concept is to develop the system for better reuse. In this case the components are developed with focus on high reusability. The highly reusable components should be able to adapt different platform and should be independent of language. In this chapter, the focus is diverted on the first approach where the system are being developed using the existing components. In case of reusing the developed components, there is a need of quantified approach to assess the reusability of the components before taking it for reuse in development. We have presented metrics to estimate the reusability of the components using soft computing techniques.

5.2 Reusability Metrics

To achieve the specific functionality, there are similar types of components available from software product industries. Therefore, it is important to select best reusable component from the bunch of less and good quality components. To quantify and classify the components besides other quality characteristics, reusability measurement is most important. Reusability can measure a set of features that are being reused in application building using these components.

Adaptability is the key factor to selects the best component for reusability. The first sight guidelines to assess the reusability are compact code size and non-nested

structure with good documentation. Gill (2003) has explained the problems in reusability assessment and its impact on the cost and schedule and analyzed some facts to judge the reusability level of the CBSD as follows:

- A thorough reuse assessment is needed to measure the possibility of better reuse in organization to get maximum benefits.
- Cost is the driven factor for any software industry business, hence cost analysis has to be done to assess whether it is cost saving strategy or not, in case the component is reused. The estimation can be done using net present value and other factors.
- It is always better to follow the standards defined in the particular area for faster integration and understanding of the component.
- Build some specific and quick projects prototyping which can be used in wide range of applications.
- It is necessary to define the metrics for reusability measurement.

To measure the reusability of the object oriented system, various metrics have been proposed (Devanbu *et al.,* 1996; Kamiya *et al.,* 1999; Aggarwal *et al.,* 2005b; Banker *et al.*, 1991, Karunanithi and Bieman, 1993; Frakes and Terry, 1996; Barnard, 1998). In object oriented development, the measures to define the interactions between the classes and object are coupling and cohesion. In general, there are some issues in applying these metrics for components and component based development. The object oriented metrics may not be able to estimate the quality of components because in object oriented development process, the reuse is done within the class or in restricted application area. In generic CBSD, the entire component can be reused and it may interact with multiple applications, also in some cases across the different physical locations.

Poulin *et al*. (1993) analyzed a bundle of metrics to quantify that how many man hours can be saved in case of reuse. The study produces the result that if an organization spent time and cost to classify and integrate the reusable piece into product, it will reward a variety of benefits. In their study, the cost was assumed as dependent factors on shipped source instructions, reused source instructions etc. Their study proposed different cost and productivity based reusability metrics. Some of them are reuse cost avoidance, reuse value added, extra cost. Based on the source code availability of component, these

metrics are designed. These metrics are not applicable to the component for which source code is not available. Washizaki *et al.* (2003) attract the attention on importance of reusability for component reuse and developed a reusability model for black-box components. They have identified the factors affecting the reusability such as understanding the functionality of component, adaptation capability to the new environment and porting the component to the new environment.

Cho *et al.* (2001) have presented a study on metrics to measure different aspects of components such as reusability, customizability and complexity. The two approaches were proposed for reusability measure of components. The first reusability measure as Component Reusability (CR), which is calculated by dividing sum of interface methods providing commonality functions in a domain to the sum of total interface methods. The second measure, Component Reusability Level (CRL) is derived to measure the component reuse level per application in CBSD. Boxall and Araban (2004) have concluded that understandability of the components has a good amount of impact on the reusability. The understandability of component can be achieved using the component's interface properties. They also proposed a metrics analysis to define the understanding of the interfaces in components using the argument count, interface size, argument repeat scale etc. They have not validated the approach with the project data and also important reusability factor complexity is missing from the metrics. Kumar *et al.* (2012) presented a review on quality aspects of the component based systems. Reusability is an important factor of software quality. They have conducted a review of the research papers related to quality of components based systems on the basis of various factors including practicability, validation proof etc., and concluded that soft computing approaches have not been explored in this area so far. Based on the importance and applicability of soft computing approaches in software engineering, Kumar *et al.* (2013) have developed a framework to prompt a chance to decrease the software defect density in a release by predicting the defect density using soft computing techniques for the subsequent software releases of a software product. The outcomes of the research have indicated that ANN results are better than FIS in predicting the defect density. Sharma *et al.* (2009) have presented the ANN based method to predict the reusability of components. They have indicated a limitation that more number of components may produce better results for the

training and testing. Sandhu *et al.* (2008) compared the fuzzy, neuro-fuzzy and fuzzy-GA approaches for reusability evaluation for components. Using their model, the reusability of components is estimated on the basis of independent vectors combination.

Gill (2003) has discussed some critical issues related to the reusability of components and the benefits of the reusability in cost optimization as well as in schedule reduction and the process of measuring the reusability of the components has also been analyzed. Many researchers have analyzed the important metrics which are affecting reusability. Sindre *et al.* (1995) identified them as understandability, flexibility, portability and confidence to estimate the reusability. Rotaru *et al.* (2005) defined complexity, adaptability and compos-ability as most influencing factors to describe the reusability and used these to estimate reusability. Mili *et al.* (1995) find two factors only affecting the reusability in the analysis such as usability and usefulness. Singh *et al.* (2011) have developed a neuro-fuzzy framework to assess the reusability of software system. They have taken four independent factors as input to estimate the reusability. Jatin and Gaur (2012) have highlighted the importance of reusability assessment for component based system development. They used several quality factors and fuzzy inference system to estimate the reusability of components. However, they did not validate the proposed approach using real project data.

5.3 Metrics to Estimate the Reusability Components

To reduce the development time, software reuse methodologies have been used across the software industries. Software reuse is a method to assemble the software components from the existing software. To take advantage of reuse concept, it is necessary to measure the software reusability of the existing components. Although, there are various statistical methods exist to find the reusability of the components but soft computing has to be explored for components' reusability estimation. The aim of this chapter is to design, validate and compare FIS, ANN and neuro-fuzzy approaches in prediction of software reusability of software components. In the proposed research, FIS, ANN and neuro-fuzzy techniques have been applied. The neuro-fuzzy technique yields to better accuracy than the standalone fuzzy logic and ANN approach. Six important

independent factors have been used to estimate the reusability of software components. This proposed approach has also been validated against the project data sets.

5.3.1 Classification and Analysis of Influencing Factors for Reusability

If an old component is not mature enough in terms of the known defects and the reused frequency, then the new product will have reliability and quality issues where it is being used. The release version i.e. the number of tested release of the components can help to decide the future of component reusability. The higher number of releases will provide better reliability of the component. A component design will not be efficient if reusability aspect is not considered because the component has been used in multiple applications and across different platforms. Hence, the design should not be against the customization as per the multiple application and platform requirements. Complex interface will be a trouble in terms of efforts, cost and schedule. Therefore, a component should have less complex interfaces. The literature is the most powerful tool to learn about the unknown things. Hence, well documented reference will help in reuse of the component.

Based on the literature evidences (Singh et al. 2011; Boxall and Araban 2004; Gill 2003; Sindre et al. 1995; Mili et al. 1995; Jatin and Gaur 2012), the set of metrics are defined and then based on correlation in the data set, the factors are finalized. After analysis for most correlated factors from the data set, we have identified the set of software metrics which plays an important role and responsible to estimate component reusability. After the subsequent analysis, these are narrowed down to three metrics, which will be possible member for building a new metrics. These metrics can be collected easily by the software industries. The following facts have been considered while deciding the attributes:

- The metrics, which has high impact, either positive or negative impact on component reusability.
- Simplified and generally available metrics
- The attributes, which can be easily collected during the software development process.

- A large set of software attributes were available, which are narrowed down to few, using correlation.

The following six most influencing and independent factors are identified for reusability assessment of components:

- Known defects in the recent release
- Software release version i.e. number of releases
- Customizability
- Interface complexity
- Portability
- Understandability

5.3.1.1 Known Defects

Known defects are defined as the count of existing bugs in the current and latest release of the component. A component cannot be completely defect free, some defects may exist which have not been discovered yet in testing phase. Due to business need, at the time of market exposure or delivery, the component may be delivered with existing list of known defects. The component can be made available with the defined threshold of defect density. It is critical business need to showcase and deliver a component even with some defined level of defect density. The defects play an important role in component reusability. If a component is having more existing defects, the reusability and quality will decrease for the component itself as well as of the system where it is being used.

5.3.1.2 Number of Releases

The other important factor in software reusability is the number of component releases delivered in the field and being used by the various products. The higher count of the release will give more confidence in terms of maturity. Hence, it plays an important role in reusability. High release count will produce more reliable and reusable software component.

5.3.1.3 Customizability

Customizability is the ease of modification in component whenever needed in application. If customizability is high as per expectation from the component, then the component shall be more reusable and also it will be easy to maintain the component for future releases and phases (*Washizaki et al. 2003)*. In case of high customizability, the lifetime of the component will be high. This will lead to a long term business perspective in terms of the component life time and the revenue generation. It is generally measured by writable properties of the component being assessed. Washizaki *et al.* (2003) have defined the formula to evaluate this metrics as given in equation (4.1).

$$Customizability = \frac{No.\ of\ set\ methods}{Total\ number\ of\ properties} \qquad (4.1)$$

This metric can help to measure the adaptability of interface that how much the interface is customizable. Hence, this metric can be used for reusability forecasting and values may vary between 0 and 1. The high value means the component is highly customizable.

5.3.1.4 Portability

Portability is the degree of ability to perform in different environment, whenever needed from development or business perspective. The component should be able to perform with defined requirements with no or little change. Earlier, there were not many types of platforms available but recently there are different type of platform and environment being developed. Although, it may not be feasible that a component will work on all platform but there are certain standard bodies which define some global platform standards. In this line of discussion, the component should work on the standard platform for wider reusability across the platforms. At least the component should perform on selected and specific defined standard platform without any change. In case of any change requirement, it should be a little change with less efforts and cost. It can be concluded that for better and wide reusability of component, component should have high

portability. Portability can be used as influencing factors for reusability assessment of a component.

5.3.1.5 Interface Complexity

Components can be treated as black box, in which we have only doors to access and use it. In some cases source code of these components may be available but in some other cases such as library and linking, the source code is not available. The user or developer will have the only choice to use its interfaces. Hence, for high reusability, components should have well defined interfaces with less complexity. The high complexities of the interfaces will results to complex reuse and hence high efforts to change and understand the components. The approach proposed by Sharma *et al.* (2008) have been used to measure the interface complexity of components.

5.3.1.6 Understandability

If, source code of the component is not available, the documentation is the only way to understand the features and interfaces of the component. The documentation helps the user or developer in component integration and developing the module to interface with components. If, detailed and clear documentation of an Application Program Interface (API) of the component is not available, there may be a big trouble in project execution, and the organization should get ready for such high risk of over cost and over schedule. The document may be in form of manuals, demos, automatic help system and other information, which can be important for the person who is going to reuse this. A well structured documentation should include the functional description, API reference manual, system admin guide, system high level architecture and complete system reference manual etc. Here, an important aspect to be noted that there should not be any mistake or ambiguity in documentation. There may be a major issue in case of wrong and ambiguous information in the reference documents, which can lead to a reliability and quality risk for a product which is being developed using these components. Therefore, a conclusion can be drawn that the documentation about the component should be easy to understand, complete, unambiguous for better reuse of the component. It will not help

only the reusability but also help in maintenance of the component as well as in the product maintenance where it is being used.

5.3.2 Fuzzy Logic Based Reusability Estimation Metric

The objective of the proposed work is to develop a soft computing technique based tool, which can estimate the component reusability. The reusability is considered as derived metric using the combination of above six metrics described in section 3.1. The values of these metrics can be measured using the appropriate formulation. In the following sections, the principal constitutes of soft computing techniques such as fuzzy logic, ANN and neuro-fuzzy techniques are applied to develop reusability metric using the six independent attributes for component reusability estimation.

In this section, the fuzzy logic based reusability metric has been proposed to assess and predict the reusability of components. Fuzzy logic is helpful tool for decision making problems in engineering disciplines. It is not possible to predict the reusability of components directly. Hence, using the six independent attributes, fuzzy logic based reusability metrics is designed. To predict the reusability, the fuzzy inference system is selected because fuzzy logic has been widely used in software engineering discipline and work well to solve uncertainty problems with less or no data, using the expert knowledge (MacDonell *et al.*, 1999 and Ryder, 1998). This uniqueness makes fuzzy logic a better technique than the data driven regression and data models. Later, when data is available, it has capability to adapt to new environment (Sailu et al., 2004). This is the advantage of fuzzy inference system over other soft computing approaches.

5.3.2.1 Implementation and Experimental Design of Fuzzy Logic Based Reusability Prediction Metric

Zadeh (2002) defined fuzzy logic as a platform where the mapping can be generated in input and output with natural language expression. As per Sivanandam (2007), fuzzy logic is a mathematical foundation which deals with uncertainty problems and a good technique for vagueness of information. As other computing technique, such as ANN uses the training data for decision making, fuzzy logic uses natural language expressions in the form of if-then rules. These rules are designed on the basis of the

opinion from experience and experts' opinion. Musilek *et al.,* (2000) have observed that the simple and best prediction models are based on experts' opinion. Even in some specific cases, the expert opinion based model performs better that the regression models. Hence, inference can be drawn that the fuzzy based approaches can be used in reusability prediction and expert knowledge can be introduced in fuzzy based reusability prediction metric.

To implement the fuzzy logic theory in prediction model, the various input vectors are formulized using fuzzy sets. Then these fuzzy sets are represented by membership functions. The membership functions are represented by set of real numbers and range of the set is bound be in the interval [0,1]. In MATLAB tool, approximately eleven membership functions are available. In the proposed reusability prediction metrics, triangular membership function (TMF) is used for mapping of input data as it is frequently used in prediction models (Zadeh, 2002). The membership function is processed by fuzzy domain inference system and expert knowledge base which contains the rules and data. Thereafter, in the defuzzification process, the fuzzy space is mapped back to a quantifiable numerical value of reusability (Aggarwal *et al.*, 2005b).

The reusability metric has been designed using the above six independent metrics including Known defects in the recent release, Software release version i.e. number of releases, Customizability, Interface complexity, Portability, Understandability. As the reusability is a derived measure and cannot be estimated directly, hence the combinations of these are being analyzed for reusability prediction. The fuzzy logic technique uses all the six factors as input and produces the crisp value of reusability of the selected component. It uses the fuzzy inference rule base to derive the reusability as output. The inputs are classified into fuzzy input set, Low, Medium and High. The output will be as predicted reusability in terms of Low, Very Low, Medium, High and Very High. Rule base is designed for possible 81 combination of input-output vector. Based on the fuzzy expert system, each rule has direct mapping of input vector to the output. Some of the rules from FIS are described in Table 5,1:

Table 5.1: Rules from FIS

Customizability	Interface complexity	Portability	Understandability	Defects	Release	Reusability
L	H	L	L	H	L	VL
L	H	H	H	M	M	M
M	M	M	L	M	M	M
M	M	H	H	L	M	H
H	L	H	H	L	H	VH

Total 81 rules are created and activated in the proposed approach. Then rule base is generated in FIS. A rule is executed on the basis of the specific set of input vectors. MATLAB rule viewer can be used to find the reusability output corresponding to the input vector set of six independent attributes. Various input and output parameters and variables which are used in fuzzification process are described in Table 5.2.

Table 5.2: Values of Inputs and Output Variables

System	[System] Name='reusability_FIS' Type='sugeno' Version=2.0 NumInputs=6 NumOutputs=1 NumRules=81 AndMethod='prod' OrMethod='probor' ImpMethod='prod' AggMethod='sum' DefuzzMethod='wtaver’
Input6	[Input6] Name='Release' Range=[-1 1] NumMFs=3 MF1='low':'trimf',[-1 -0.62 -0.3] MF2='Medium':'trimf',[-0.415 -0.0247 0.373] MF3='high':'trimf',[0.2 0.6 1
Input5	[Input5] Name='Defects' Range=[-1 1]

	NumMFs=3 MF1='low':'trimf',[-1 -0.62 -0.3] MF2='Medium':'trimf',[-0.415 -0.0247 0.373] MF3='high':'trimf',[0.2 0.6 1
Input4	[Input4] Name='Understandability' Range=[-1 1] NumMFs=3 MF1='Low':'trimf',[-1 -0.643 -0.315] MF2='Medium':'trimf',[-0.447089947089947 0 0.41] MF3='High':'trimf',[0.267 0.7 1]
Input3	[Input3] Name='Portability' Range=[-1 1] NumMFs=3 MF1='Low':'trimf',[-1 -0.6 -0.304] MF2='Medium':'trimf',[-0.389 0 0.304] MF3='High':'trimf',[0.2 0.62 1]
Input2	[Input2] Name='Int_Complexity' Range=[-1 1] NumMFs=3 MF1='high':'trimf',[-1 -0.62 -0.3] MF2='Medium':'trimf',[-0.415 -0.0247 0.373] MF3='low':'trimf',[0.2 0.6 1
Input1	[Input1] Name='Customizability' Range=[-1 1] NumMFs=3 MF1='Low':'trimf',[-1 -0.6 -0.352] MF2='Medium':'trimf',[-0.474 0 0.41] MF3='High':'trimf',[0.32 0.7 1]

Output1	[Output1] Name='Reusability' Range=[0 1] NumMFs=5 MF1='VLow':'constant',[0] MF2='Low':'constant',[0.25] MF3='Medium':'constant',[0.5] MF4='High':'constant',[0.75] MF5='VHigh':'constant',[1

Fuzzification and defuzzfication process with the membership functions of the six independents factors are shown in Fig 5.1, Fig 5.2 and Fig 5.3.

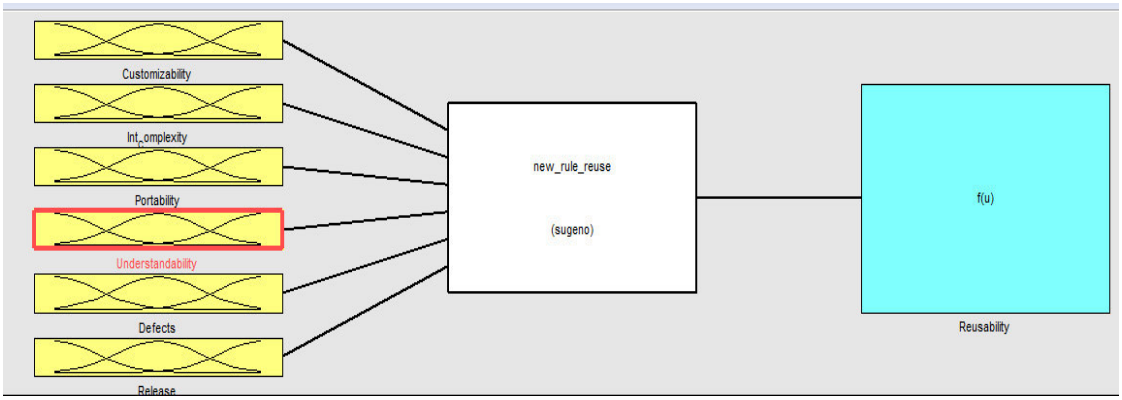


Fig. 5.1: Fuzzification of Independent Reusability Attributes into Predicted Output

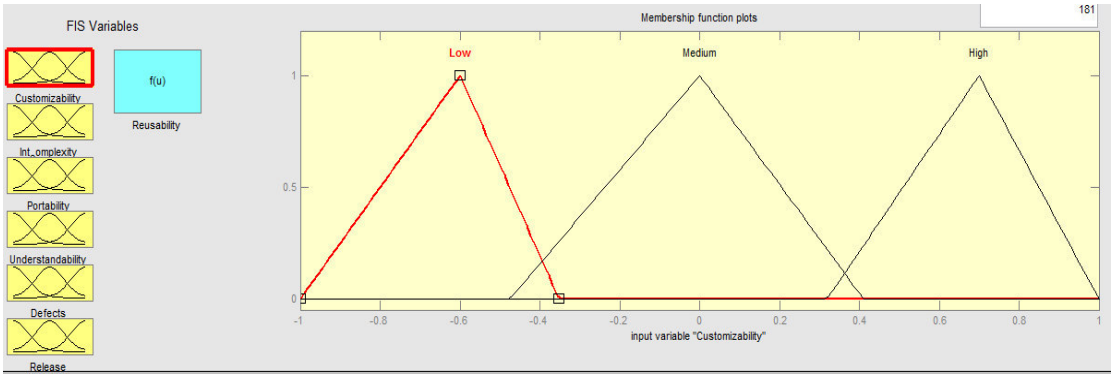


Fig. 5.2: MF for Used Customizability Attribute in FIS

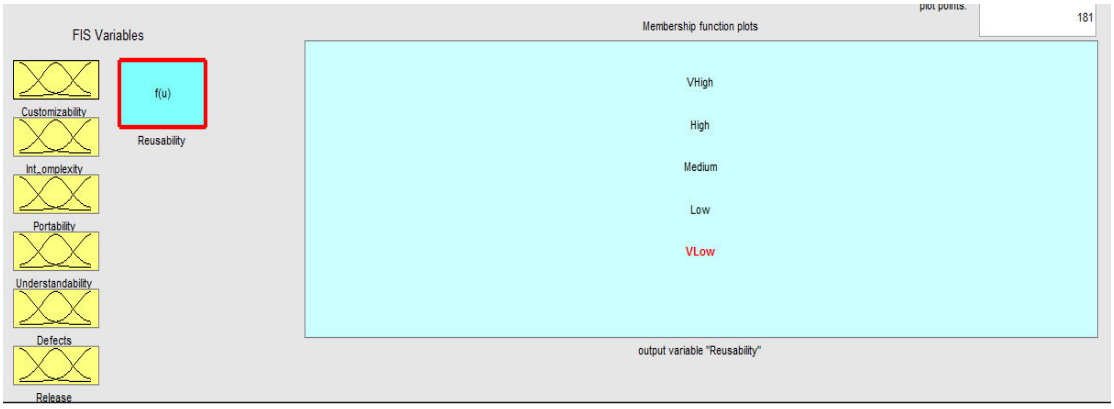


Fig. 5.3: MF for Used Reusability Attribute in FIS

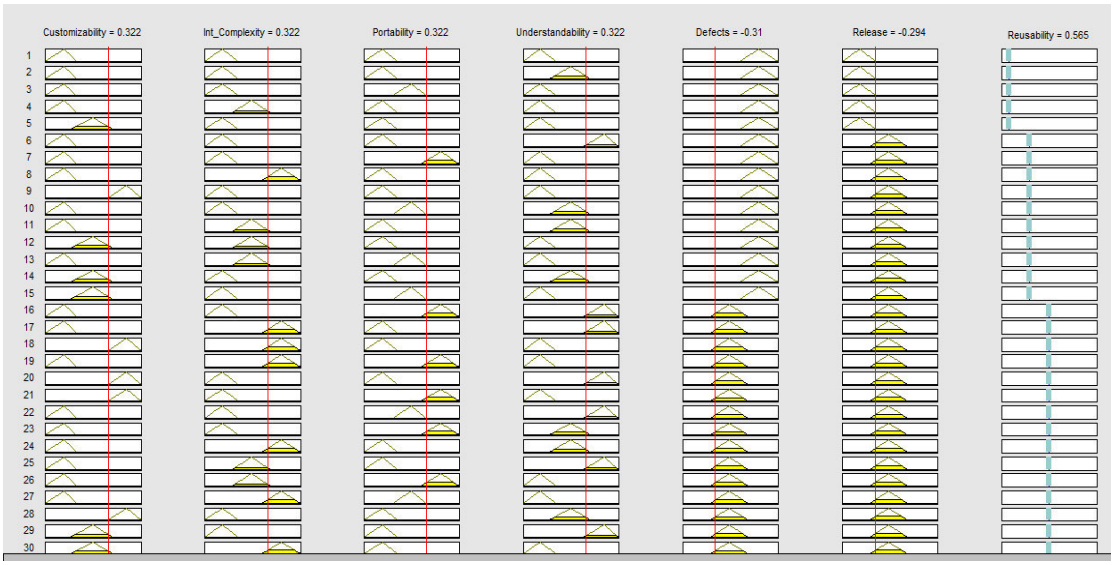


Fig. 5.4: Rule Viewer Showing Some of the Rules of FIS

Using six independent factors as input, with the help of the rule viewer, the reusability can be predicted as an output.

5.3.2.2 Empirical Evaluation and Validation

The data for several components are collected from web sources and in-house development (*www.jars.com and www.elegantjbeans.com*).. These components include very simple calculator and complex inventory management system. These are developed by using different technologies ranging from Java beans and .Net to open source technologies. Each component has been analyzed carefully and extracted the data of the

attributes which are related to parameters chosen in the current study. The goal of our work is to develop a tool for measuring reusability of the software component. Reusability is considered as a measure of six factors mentioned in section 3.1. The values of these factors can be measured by using the appropriate formulation. Customizability metric is the ratio of writable properties to the total number of properties. Documentation and portability can be classified from low to high categories. All these factors are normalized by using min-max normalization. Min-Max normalization is used for linear transformation of the original data. It transforms the original input range into a new data range (typically -1 to +1 range).

The validation is done using the 21 component data set out of total 80 data vector set. Remaining 59 component data set is being used for the training of ANN model. For the better comparison, Similar to the ANN validation, the same 21 data set vectors are used for testing process of FIS approach. Each of the 21 vector set is given input to the FIS rule viewer in MATLAB and corresponding reusability value is estimated as an output. The same process is repeated for all 21 input vectors. Taking the corresponding actual values and the FIS predicted reusability values, RMSE is calculated. In best case scenario, the RMSE value is observed as 0.1971. Using the rule base and available data set, FIS based reusability metric shows 81% accuracy. The advantage of the FIS approach is that it can work even if there is less or no data for components. It indicates that FIS approach can be used for reusability prediction before component reuse by application developers.

5.3.3 ANN Approach for Reusability Prediction Metric

Reusability metric is designed using the influencing factors of reusability and evaluated by applying ANN. As per Haykin (2003) and Mayrhauser (1995), neural network is a well proven technique for analysis of pattern clustering and classification. To quantify and predict the reusability, ANN is selected because of the following key factors:

- ANN has the adaptability to the changed environment.
- ANN is capable to classify the pattern

- ANN is adjustable in nature because it can adjust the network complexity according to the given problem, hence it is better that the traditional models.
- Frequent change in human resource happens in the software industries. Hence, capturing the expert knowledge and training the network can help to automate the processes, which will be independent of person working on that.

ANN is less complex in terms of making the model and training (Hudson, 2003). The main characteristic of neural network is learning capability and it can generalize the one form of data to other. Also, in case of incomplete and erroneous data, the ANN is able to adapt to the environment and produce the correct output.

ANN models have been used for various engineering disciplines such as medical diagnosis, data mining and prediction in engineering and economic analysis (Widrow *et al.*, 1994). However, researchers have started using ANN for prediction and quantification problems in software engineering but there is less progress for component based system. ANN has been used extensively in efforts estimation and scheduling using the history data analysis. There is a scope of ANN application in prediction of components' attributes. Adhering to these facts, ANN approach has been used to predict the component reusability.

5.3.3.1 Artificial Neural Network

In ANN Architecture, there are variety of learning algorithms and activation functions. The learning capability can be defined as adjustment in values of weight connection that infer to collection of information and which can be later recalled, simulating the human brain behavior. Before training of the ANN, the default weights are set to some random values but within the specified range. The researchers have used two types of neural network learning, supervised and unsupervised.

The ANN has learning capability, it can learn from history with simulation of human learning. ANN is also capable enough to establish the complex set of relationships between the independent variables (input) and the dependent variable (output). The disadvantage of the ANN application is that it is not easy to understand by the users and generally considered as black box. Also, there is no standard guideline for the topology, layers count, units per layer count, default weight etc.

5.3.3.2 Neural Network Architecture

ANN architecture is differentiated by the types of interconnections between neurons. The feed-forward architecture contains connections which pass through the output in a single direction towards the neurons of the next layer. In other type of architecture called feedback network architecture, the architecture permits the outputs of neurons to the preceding neuron layers as well. The network architecture with closed loops mode are referred as recurrent networks. Feed forward architecture based ANN are faster than the feedback architecture ANN because there is no multiple pass needed in case of feed forward architecture. The feed forward architecture based ANN has been frequently used to analyze and solve the prediction problems (Mukherjee and Deshpande, 1995).

5.3.3.3 Steps for Designing a Neural Network

The ANN modeling process involves the following steps:

- The input data pattern and desired output as target is presented to the neural network. The complete data is partitioned into two parts; training data set and testing data set. Training data set is used in algorithmic learning process of the ANN model and test data is used for the validation and also to define the threshold when to stop training.
- ANN structures are defined by declaring the hidden layers and the number of neurons per layer. The ANN internal parameters are declared and set before the training starts.
- In next step, training is initiated. The training contains the output generation based on the given inputs and defined weights. Back-propagation learning algorithm is used for network training with weights adjustments to minimize the delta between the ANN output and the desired outputs.
- The evaluation process starts to check learning level of the ANN to solve the given problem. To achieve the acceptable and defined accuracy, training process halted periodically and performance is tested at each step. When the defined accuracy level is achieved, the ANN model and network can be used for utilization in prediction problem.

- At the final step, the ANN need to be trained and tested with variations of ANN parameters' values to select the best performed structure for the given data set and problem. To finalize the efficient model involves training, and the results comparisons of different ANN parameters and algorithms.

5.3.3.4 Learning Algorithms

There are well defined set of rules for the learning, these set of the rules are called learning algorithm. The different ANN learning algorithms and their various variations exist in MATLAB. In the current study, feed forward back-propagation ANN algorithm has been used. Back-propagation algorithm is the general form of Widrow-Hoff learning rules. In this algorithm, input and vectors are used for training till mapping established in associated input with a particular output vector to approximate the function.

The *trainlm* and *trainbr* functions of back-propagation algorithm are used for training. The results are analyzed and compared with each function and neuron frequency. *trainlm* is Levenberg-Marquardt optimization based network training function. It is the mostly used algorithm to solve the prediction issues. *trainlm* is fast but need more memory resources than others. *trainbr* is also Levenberg-Marquardt optimization based network training function. It minimizes a combination of squared errors and weights, and then determines the correct combination to produce a network that generalizes well. The process is called bayesian regularization. For the experimentation, the function *trainbr and trainlm* are used in the current study.

5.3.3.5 Design and Experimentation of ANN Model to Predict Reusability

The values of independent attributes are measured using the appropriate metrics. The data set for input training vector are collected from eighty exemplars of Java Beans components (*www.jars.com and www.elegantjbeans.com*). The input data vector set of 80 components is divided into 80% training and 20% validation data set. Hence, 59 vectors are considered for training and 21 vectors for testing and validation. The neural network tool in MATLAB expect the input values to lie in-between the 0 and 1. Hence, all the metrics need to be normalized in this range. *Min-Max* normalization is used to fit the data

in this range. *Min-Max* is the linear transformation of the original collected data into the -1 to +1 range.

If, Min_A and Max_A are the minimum and maximum values of A, then the linear transformation will map the value v to v' in the target range -1 (min_{target}) to 1 (max_{target}). The formula, shown in equation (5.1) is used to transform the values.

$$v' = \left(Max_{target} - Min_{target}\right) * \left(\frac{v - Min_A}{Max_A - Min_A}\right) + Min_{target} \qquad (5.1)$$

The designed neural network contains the different number of neurons, 5, 10, 15, 20, 25 for hidden layers. By adjusting the weights of neurons and layers, the network is trained to minimize the difference between the desired and real network output. ANN is able to learn by using the neuron connection and weight vector to minimize the errors in training data set. Both learning functions, *trainlm* and *trainbr* are used to cover different aspects of algorithms. The comparative analysis is done for results of both algorithm.

learngdm function is used for adaptation learning in experiment. The transfer function *tan-sigmoid* is selected for adaptation and hidden layers. The transfer function is used to convert the integrated input to net output. Taking the combination of above architectural attributes, the network is trained using 1000 epochs and goal was set as zero. Other than this parameter, the parameters were set to default in *nntool* of the MATLAB.

The evaluation decides the success or failure of any technique. To evaluate the efficiency of proposed ANN architecture, the measurement of the achieved learning capability is required. RMSE is the commonly used parameter to judge the performance of ANN. RMSE is a good measure to estimate whether the predictions from network is close to the target or not. RMSE will be increasing in the initial training stages and after having sufficient number of training iterations, it becomes stable. RMSE is calculated by standard formula as:

$$RMSE = \sqrt{\frac{(A_1 - P_1)^2 + (A_2 - P_2)^2 \ldots + (A_n - P_n)^2}{n}}, \qquad (5.2)$$

Here, $A_1, A_2, \ldots, A_n$ are the original output and $P_1, P_2, \ldots, P_n$ are predicted output. The best model is characterized which has minimum values of RMSE. n is the number of input vectors.

The different number of neuron in ANN architecture attributes are used, which range from 5 to 25 to find out the best performer case. The output layer neuron kept constant to 1 for all the cases.

After the successful training, the validation was done using the 21 exemplars input vector data set. The validation is done by simulation of both the training functions to analyze the difference in performance. The better performance of feed forward neural network is achieved when using the *trainlm* function with 20 neurons because the least RMSE was 0.1852. All the results with different combination of neurons and performance of both functions are shown in table 5.2.

The training performance using 10 neurons and *trainlm* is captured in Fig. 5.5. The Fig. 5.6 and Fig. 5.7 depict the training states and regression results respectively. The *trainbr* performance using 5 neurons is shown in Fig. 5.8. The *trainbr* training states and regression are represented in Fig. 5.9 and Fig. 5.10, respectively.

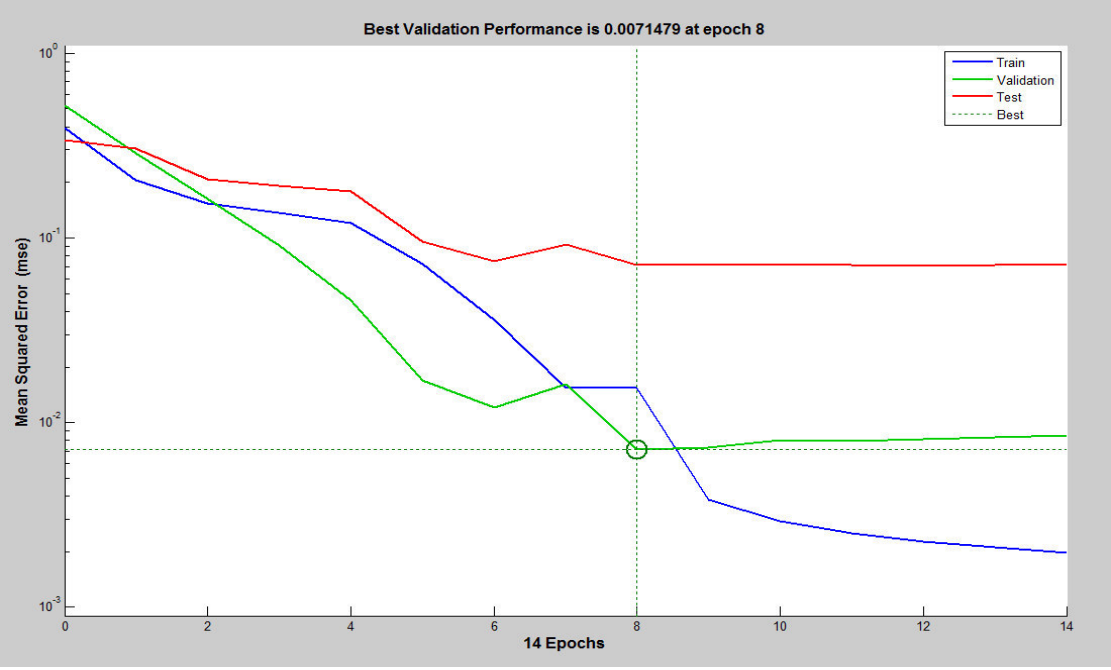


Fig. 5.5: ANN Training Performance with *trainlm* and 10 Neurons Using Reusability Data

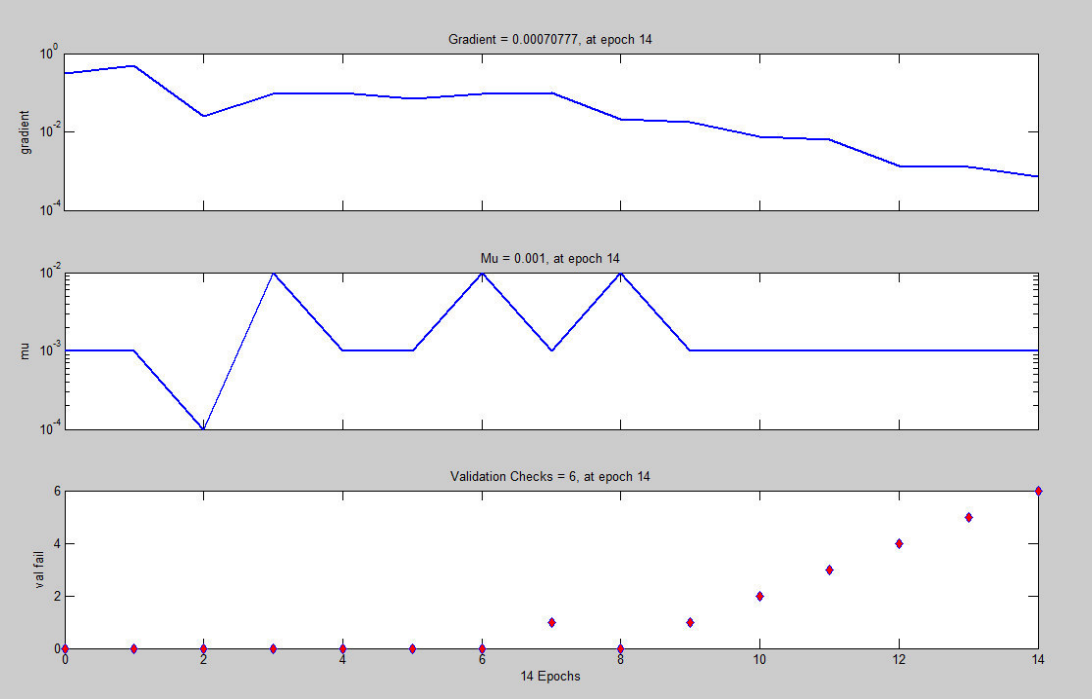


Fig. 5.6: ANN Training States with *trainlm* and 10 Neurons Using Reusability Data

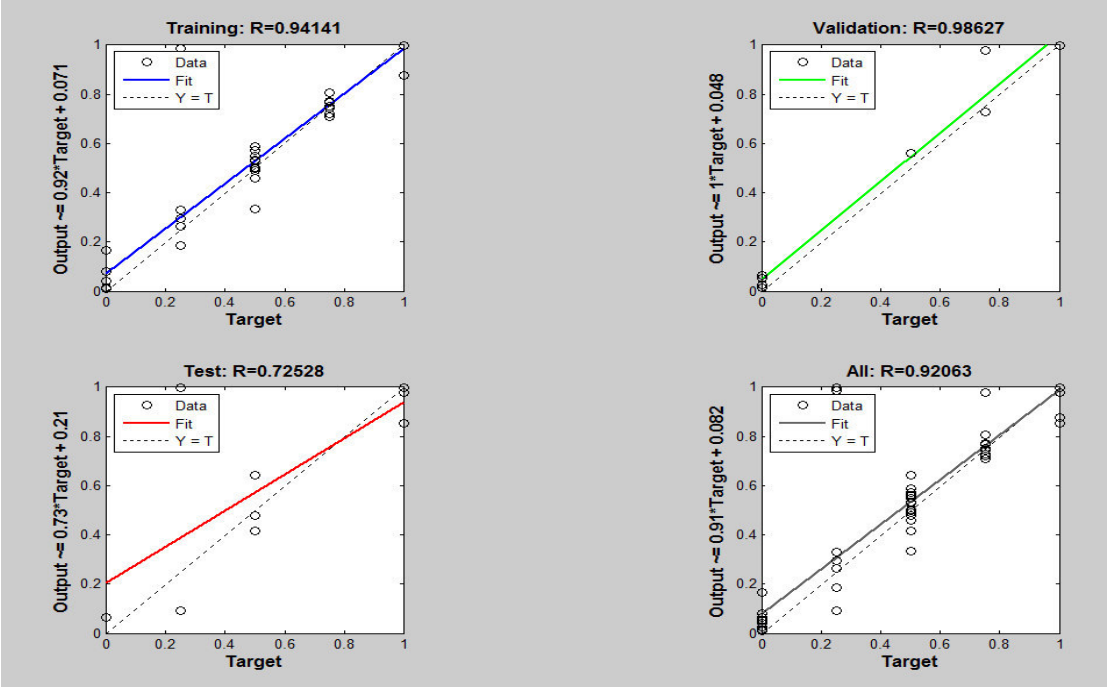


Fig. 5.7: ANN Regression States with *trainlm* and 10 Neurons Using Reusability Data

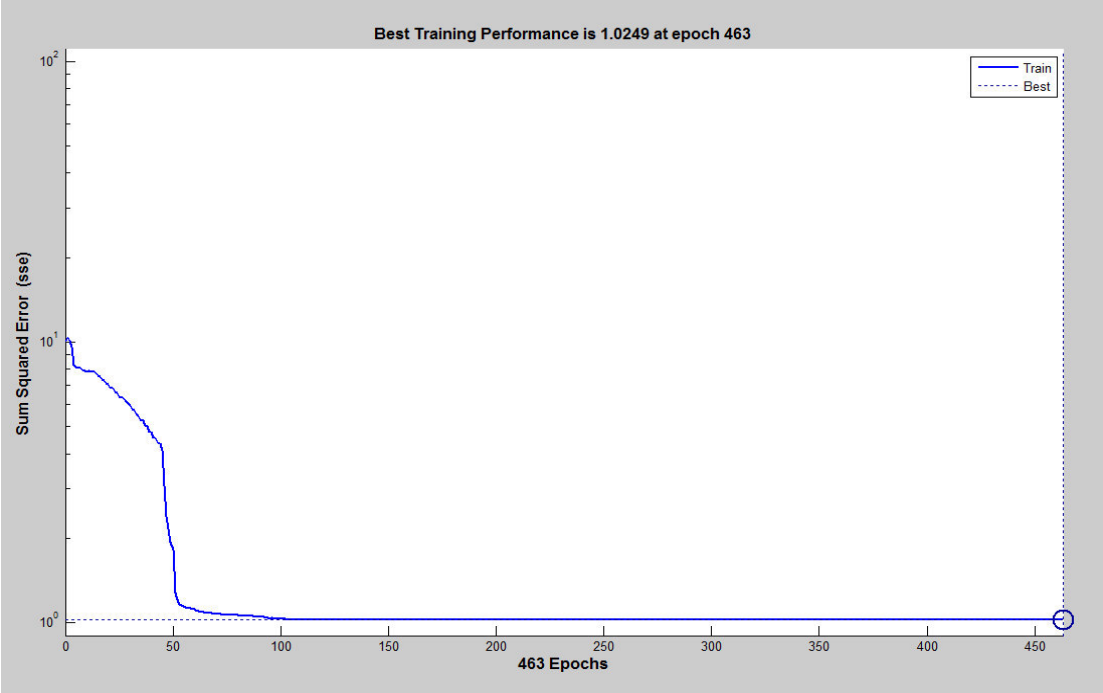


Fig. 5.8: ANN Performance with *trainbr* and 5 Neurons Using Reusability Data

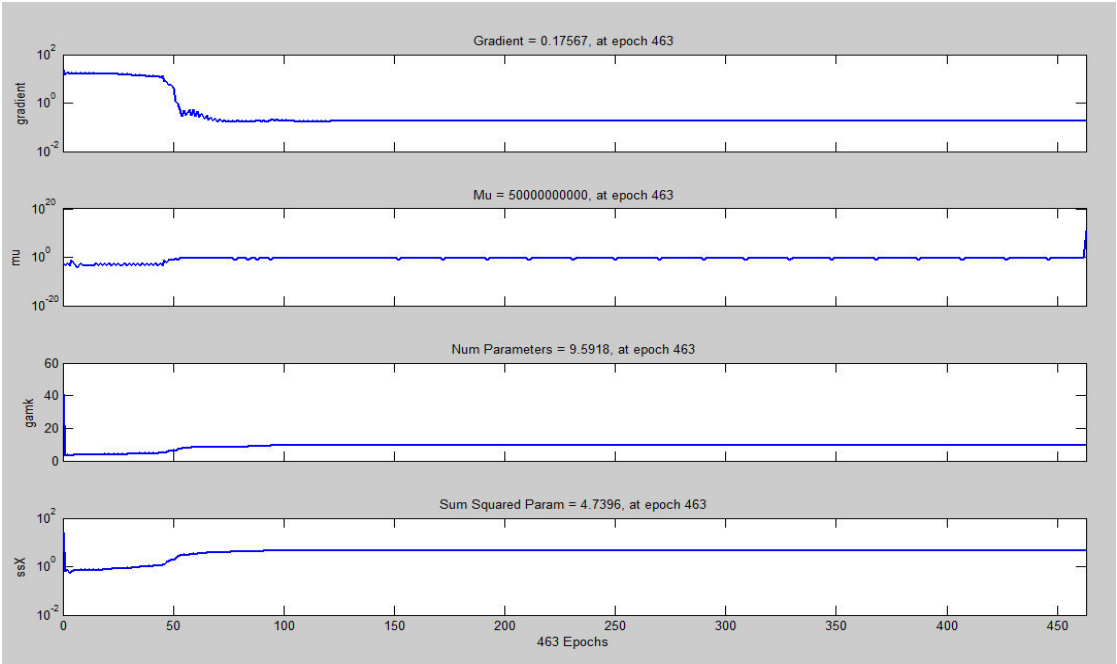


Fig. 5.9: ANN Training States with *trainbr* and 5 Neurons Using Reusability Data

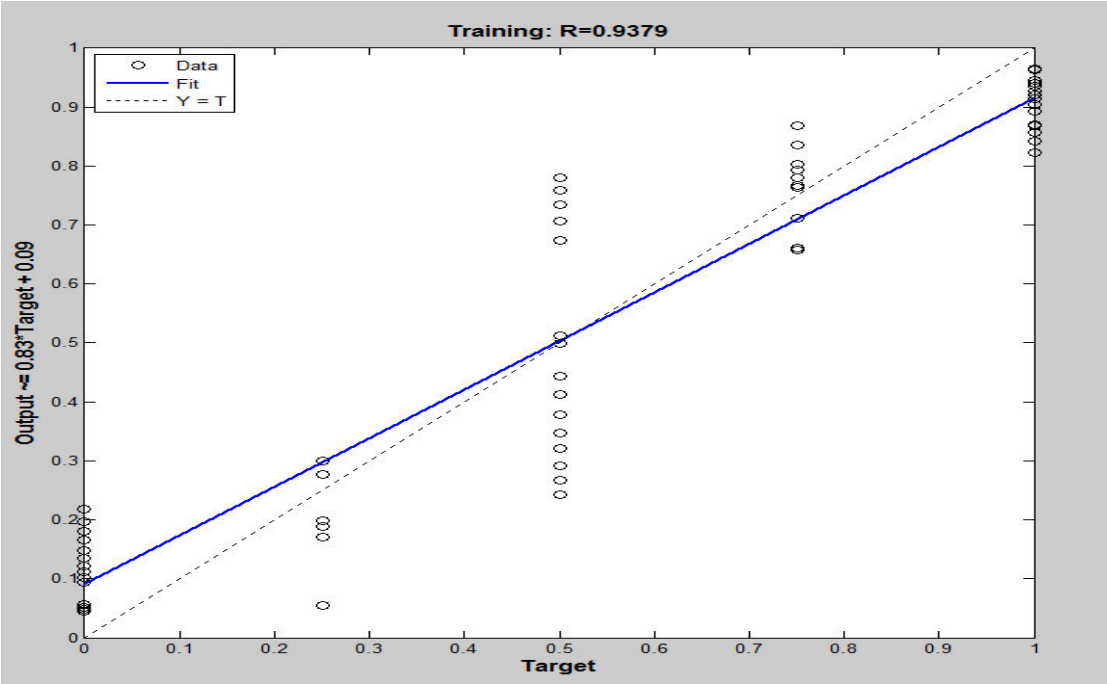


Fig. 5.10: ANN Regression with *trainbr* and 5 Neurons Using Reusability Data

The proposed ANN based reusability metrics can predict the reusability with a good accuracy level. This may be used to access the reusability of the software component for better control on component based development.

Table 5.3: RMSE Results of the Proposed Reusability Prediction Metric Using ANN Technique

	RMSE with FFBP NN algorithm for Reusability	
Network Neurons	*trainlm*	*trainbr*
5	0.1871	0.1870
10	0.1891	0.1944
15	0.1946	0.1923
20	**0.1852**	0.1909
25	0.2161	0.1859

5.3.4 Neuro-fuzzy Application to Estimate Reusability of Components

In earlier sections, the fuzzy logic and ANN techniques have been discussed to design the reusability metrics but there are advantages and disadvantages of both the

approaches. FIS is rule based system which uses expert knowledge to predict the attribute. On other side, ANN has decision capability based on the learning and training with simulation of human brain functionality. If the two approaches are combined to work in a single metric design, it may enhance the capability of the approach in metrics prediction. If fuzzy inference system and artificial neural network are combined, it forms the adaptive fuzzy inference system, which is called neuro-fuzzy system.

5.3.4.1 Neuro-fuzzy Technique

In interdisciplinary era, the recent trend is to combine the different strategies by taking complementary advantage of strategies and create a more efficient robust system. Following this approach, various hybrid soft computing techniques have been introduced such as neuro-fuzzy, evolutionary-neural, evolutionary-fuzzy and even the combination of three as evolutionary-neuro-fuzzy systems. The hybrid techniques perform better to solve the specific type of real world problems. The neuro-fuzzy is the commonly used hybrid soft computing techniques. Neuro-fuzzy technique is implemented by giving the inputs to fuzzy inference system, and then feed to ANN learning algorithm for training and testing. This technique uses the logical decision characteristic of fuzzy logic and adaptability advantage from ANN. This combination increases the predicting capability for data analysis problem.

5.3.4.2 Designing Neuro-fuzzy Method for Reusability Prediction

In the current research, a hybrid soft computing approach, neuro-fuzzy is being used with combination of independent factors for components' reusability. In literature (Singh *et al.* 2011; Aggarwal *et al.* 2006; Pedrycz *et al.* 2004; Ardil and Sandhu 2010; Kaur *et al.* 2010; Kumar *et al.* 2012; Beldjehem 2010), it is shown that neuro-fuzzy provide the better results as compare to standalone FIS or ANN because it inherits the FIS and ANN characteristic properties in a single system together.

Let $x_n,\ y_n,\ z_n,\ f_n, g_n, h_n$ be the values of the known defects, release version, customizability, complexity, understandability and portability, respectively, then

$$R_{c_n} = f_{c_n}(x_n, y_n, z_n, f_n, g_n, h_n), \qquad (5.3)$$

here, R_{c_n} is the reusability of component C_n. f_{c_n} is implemented using ANFIS in MATLAB taking x_n, y_n, z_n, f_n, g_n, h_n as input independent variables.

In the proposed approach, the neuro-fuzzy system is designed, trained and tested in MATLAB as per the following steps:

- Build fuzzy inference system in MATLAB.
- Load the training data in neuro-fuzzy system using MATLAB toolbox.
- Neuro-fuzzy based system is trained using both least square method and back-propagation. In the forward pass, the consequent parameters are calculated using least squares, and in the backward pass the premise parameters are calculated using back-propagation.
- When training is completed, trained neuro-fuzzy system is validated using the testing data.

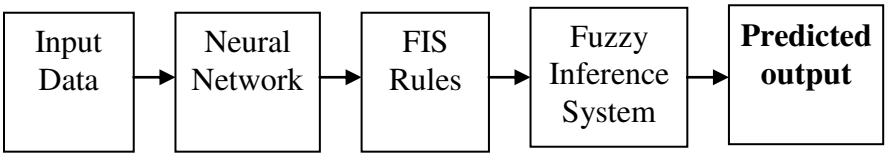


Fig. 5.11: Neuro-fuzzy Modeling in MATLAB for Reusability Metric

In the neuro-fuzzy based approach, the six variables have been used with the same data set which has been used in fuzzy logic and ANN based metrics. A rule base is designed after getting the expert opinion on the relationship of these six variables with reusability by using fuzzy inference engine. The data collected from 59 components were used to train the network by using back propagation neural network.

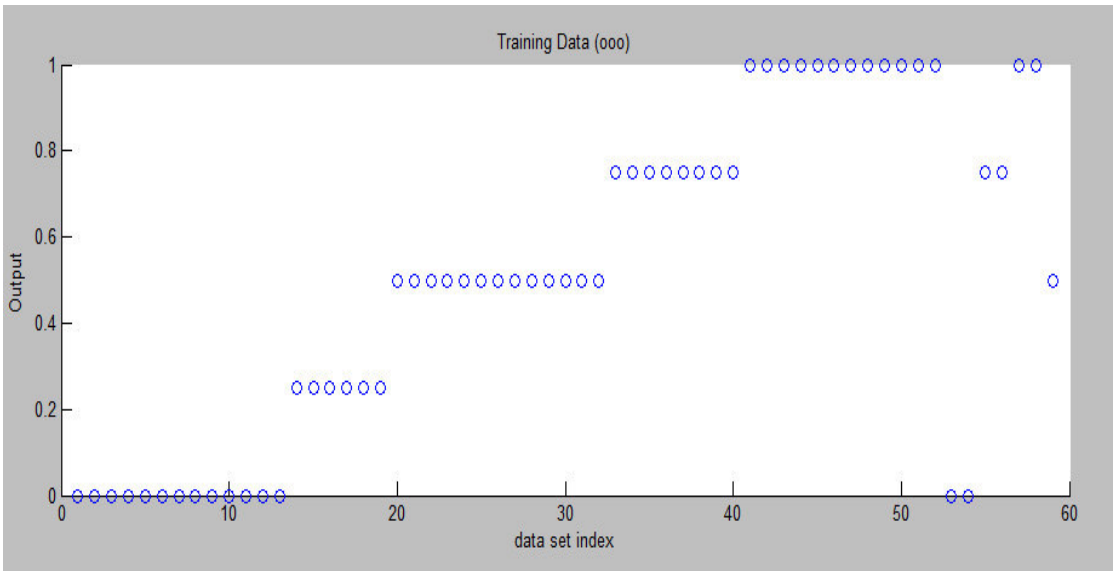


Fig. 5.12: ANFIS Training Data Pattern with Reusability Data

Table 5.4: ANFIS attributes

ANFIS Attribute	Value
Number of nodes	1503
Number of linear parameters	729
Number of nonlinear parameters	54
Total number of parameters	783
Number of training data pairs	59
Number of checking data pairs	0
Number of fuzzy rules	729

The ANFIS training data pattern is shown in Fig. 5.12. The training performance is captured in Fig. 5.13. Surface view of inputs and output can be seen in Fig. 5.14. The ANFIS structure with six input parameters is shown in Table 5.3.

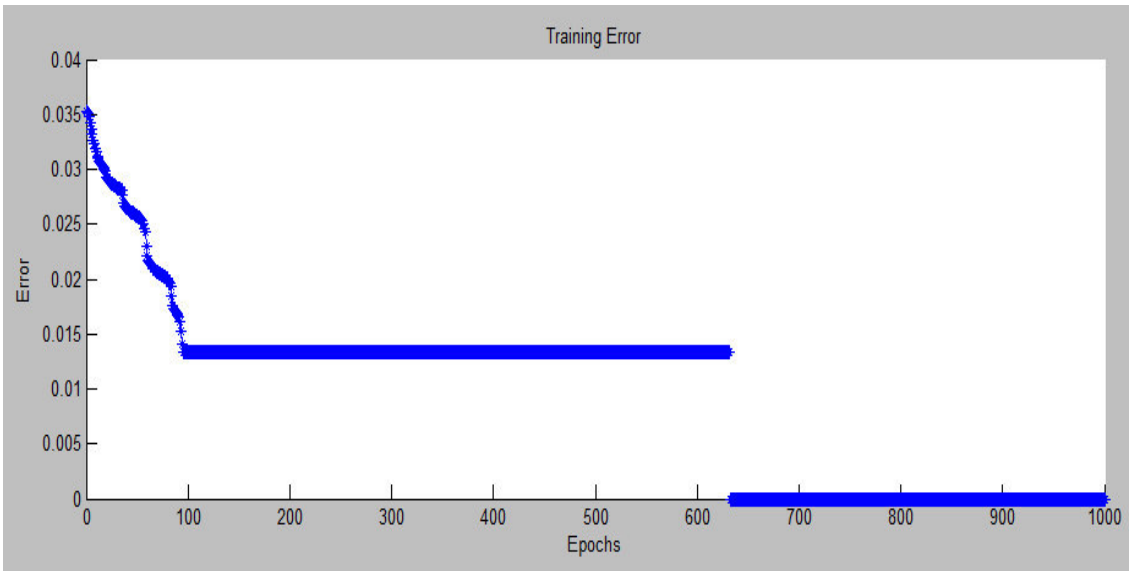


Fig. 5.13: Training Performance of ANFIS

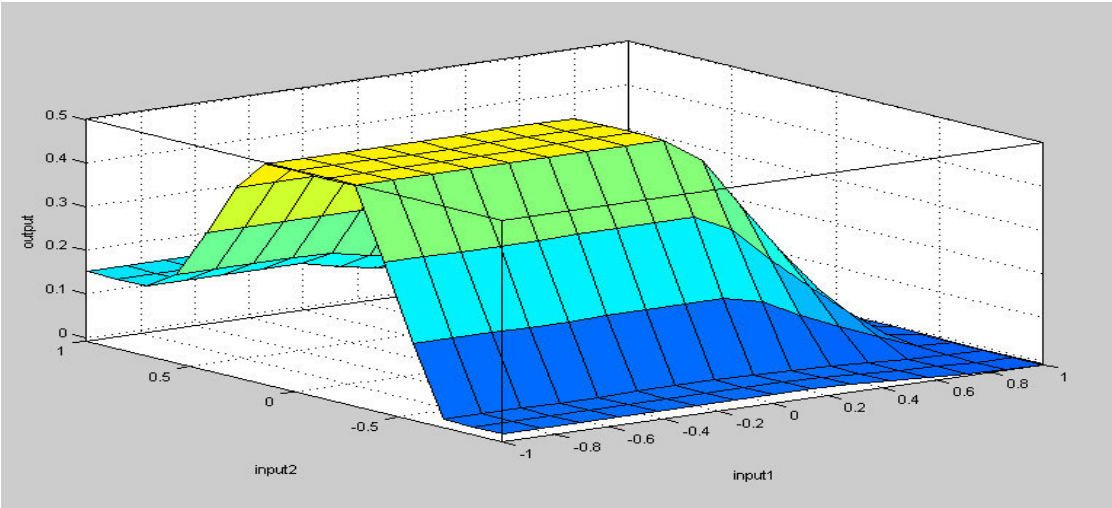


Fig. 5.14: Surface View of ANFIS Rules Output and Two Inputs

5.3.4.3 Validation of Neuro-fuzzy Approach

In validation process of the neuro-fuzzy approach, the ANFIS is loaded and test data is fed into the model. The training performance is achieved at epoch 100. The presented ANFIS model is able to predict the component reusability with RMSE 0.1695 which is better than the standalone FIS and ANN approach.

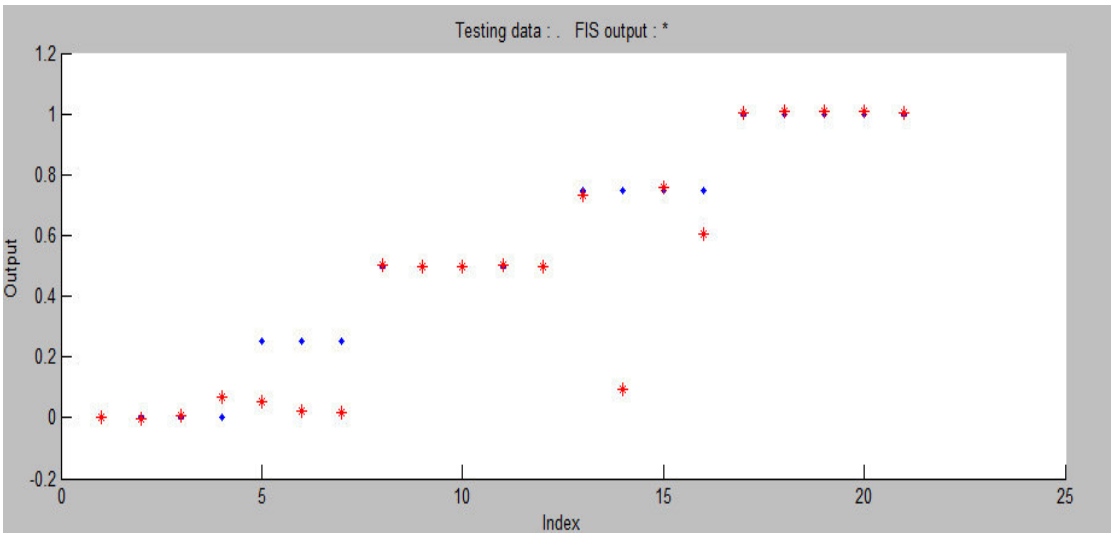


Fig. 5.15: Predicted and actual outputs of ANFIS based Reusability Metric

5.4 Comparative Analysis of FIS, ANN and ANFIS Approaches

The results of validation show that neuro-fuzzy performs better than other two approaches. For neuro-fuzzy, the RMSE is 0.1695 and for FIS and ANN, the best case RMSE is 0.1971 and 0.1852, respectively. The performance varies in case of different set of neurons, 20 neurons with *trainlm* algorithm gives the best results using the available data set. Some more analysis can be done to tune the performance of the fuzzy inference system for better results as it has advantage that the model can work without data also, which is very useful and practical in software industries because it is a overhead to collect the data during the development and design. But, in long term it is very useful and worthwhile to collect the data at every phase of development process for software quality assurance and software process improvements.

5.5 Conclusion

Considering the importance of the reusability assessment of the components, variations of soft computing based metrics have been proposed for reusability assessment of components. The metrics can classify the highly reusable components from the set of existing components' library. Also, the main advantage of the metric is adaptability because the soft computing approach can be tuned to perform better in case of any need of specific type of reusability assessment. The validation results indicate that the

proposed metrics are able to predict the reusability with good accuracy. Neuro-fuzzy produces the better results for reusability prediction than the fuzzy logic and ANN approach. Using neuro-fuzzy approach, the better accuracy was achieved with RMSE as 0.1695. Reusability is the most important for component quality and also plays an important role in selection of component for CBSD. Highly reusable component will need low maintenance and integration efforts in application. Realizing the importance of reusability prediction, the three principal constitutes of soft computing techniques i.e., fuzzy logic, ANN and ANFIS have been applied and individually validated to get the better metric for prediction.

In the present study, data used to train the network and validation is limited to 80 components. More number of components for training may produce better results. Neuro-fuzzy gives better results as compare to FIS and ANN approaches but depends on the availability of data during the development process.

Chapter 6

Estimation of Maintenance Severity and Maintainability

6.1 Introduction

In software industries, most of the work is carried out for maintenance of the existing systems. The maintenance is necessary due to some technological upgrade or any requirement change as per business needs. In software development process, software maintenance can be considered as broad activity which may include optimization, capabilities enhancements, fault removal and removal of unnecessary functionalities etc. If, there is a specific change request from the client, all the changes agreed and specified in the requirement document are included in maintenance. The maintenance can be divided into adaptive, perfective and corrective type maintenance. Removal of the residual faults comes under corrective maintenance. In case of any change in functionality to meet changed requirement, adaptive maintenance is necessary. The enhancement and optimization comes under perfective maintenance. In case of deployment of the system in field operations, the corrective maintenance is very

The major findings of this Chapter have been published in Middle-East Journal of Scientific Research, Vol. 19, Issue 10, pp: 1412-1420.

important as the live system will be idle or halted while performing the upgrade to the correct release of the system. In other maintenance, the development can be done in parallel. As per observations by Pressman (2005), 70% of the total efforts are part of maintenance of the existing system. They conducted a survey and analyzed that 17% maintenance is carried out to correct the faults, 18% for portability and the major portion 65% maintenance is done to implement the requirement change. Because the changes cannot be avoided, a well defined approach needs to be developed to quantify, control the modifications. Therefore, there is a need of effective maintenance approach need to be developed to reduce the efforts and time.

A software system contains several releases during the lifetime in the operation. The new release may be due to the enhancements as pre business need or due to the existing defects in the system. In a large and complex system, it is not possible to fix all the defects in the first software release. Hence, the system will have several releases due to the existing discovered and uncovered defects in the system and also based on severity levels of the defects.. The complete software system may have several components and modules. The models and components do not have the linear distribution of defects. Few modules or components may have more defects as compared to the other components and modules. The defects distribution defines the maintenance need, which is called maintenance severity based on the defects frequency and criticality. The maintenance severity quantifies the impact of the defect on the overall environment with high being most severe to low h being least severe. For, example low severity may imply that the defect caused a loss of functionality without a workaround where high severity may mean that the impact is superficial and did not cause any disruptions to the system. A module or component can have more maintenance severity than others.

In maintenance operation due to higher maintenance severity or business need, the system may need the enhancement.. The early prediction of maintainability can help to maintain the quality and maintainability of the subsequent releases. The maintainability estimation of the components will help in use of components in different systems. It is a critical business requirement to estimate the maintenance severity of the module and components, which defines the maintenance need of the system. The different approaches (Vaos 1998; Singh et al. 2004; Aggarwal et al. 2005a; Jianhong et al. 2010; Malhotra and

Chug 2012) have been proposed to estimate the maintenance severity and maintainability, but there is need of more efficient and practical method to estimate the maintenance severity at module and component level granularity with fewer efforts and cost investments. Present chapter analyzed the influencing factors for maintenance severity and maintainability. The maintenance severity metrics to predict the severity of the software modules and components is proposed using artificial neural network approach. Also the ANN approach to predict the maintainability index is conceptualized and validated.

6.2 Maintenance Severity and Maintainability Estimation

Maintainability can be defined as ease with which the existing software can be enhance or its state can be maintained to meet the future and business needs. In software, for better development process, and for the better maintainability of the system, there has to be some way to estimate and predict it for improvements.

As per IEEE (1998), maintainability is the ease with which a software system can be changed or enhanced to remove the defects, improve the performance or adapt to the different environment. ISO (2001) defined the maintainability as the modification capability of a software system. There can be any type of modifications such as corrective measure, improvements to adapt in new environment, and to meet the new functional requirements.

Baisch *et al.* (1995) proposed a model for improvements in software productivity based on quality. There were observations that the changes in modules can be predicted using neuro-fuzzy inference system. They used the history data to build the neuro-fuzzy model. The authors suggested that the neuro-fuzzy model can be build using the data set from the similar type of past project. The fuzzy inference system rules have to be understandable and interpretation is necessary to avoid the accidental combination. The authors also claim that if the maintenance data per module exists and gathered, the proposed approach can predict the maintainability based on the metrics data and module.

Vaos (1998) analyzed different factors of component based system maintainability. These factors includes problem in maintenance because of freeze in functionality, upgrade incompatibility, faults in components, component complexity,

middleware defects etc. The authors also provided some suggestions for maintenance work which are as follows:

- It is always good to have detailed documentation for understating the component interfaces and functionality.
- In component based systems, it is critical to select the components which are having good performance and fulfill the required functionality. The repositories should be maintained to have choice of selecting the best component as per the requirements and performance.
- During the initial estimation and planning, it can be concluded that system is large and complex. In this case, it is suggested to use CBSD as software engineering process. For small and simple applications, it is not suggested to use CBSD.
- If a component is shared by multiple applications and any change in these is not digested, then it is good to have different copied of similar components in the system.

The study presented by Vaos (1998) was quite fundamental and theoretical it gives some basic thought for understanding and quantifying the maintainability of CBSD. As per analysis by Judith and Audrey (1998), the following are the major issues in maintenance of components and component-base systems.

- The modified component may not perform well with the hardware of the application. There may be some memory and execution time related issues after modifying the component as the system may not be able to meet the new components requirements. This will results some minor or major changes in the system.
- The enhanced component needs support of additional tools and language which may not be supported by CBS.
- Modified component may need changes in security parameters of the software system.
- There may be compatibility issues after component functionality upgradation or modifications. Hence, the component and system both need some changes.
- During the integration process, the upgraded components may create the issues and may need some good amount of changes in interface code also.

In conventional software development process, the Mean Time To Repair (MTTR) is related to the maintainability, for good maintainability MTTR should be low.

Traditional measure relies on source code visibility, which may not applicable for component base development and applicable in case the source code of the components is available. In case of non-availability of component source code, the component based system can be treated as black boxes and for the developer the sources code is not available in many cases. In case of component based systems, it is difficult to identify that problem is due to internal problem in component or there may be issue in interfacing. If, the component based system is enhanced or upgraded as per new requirements, the compatibility of the enhanced system with the existing system may not work well. In this scenario, component will also have to upgrade or need some additional components to achieve the compatibility. Understanding these situations, we can conclude that to solve the maintenance issues in component based system development, we will have to analyze the components and also the component system.

In the line of discussion, to meet all the challenges related to the component based system, component developers needs to develop the components which can be easily integrated into the good amount of software systems and extensible.

Singh *et al.* (2004) considered readability of source code, documentation quality, understandability of software, and average cyclomatic complexity as input independent variable to measure the maintainability of the software systems by using the back propagation algorithm of an ANN. The produced results were very encouraging with 91.42 % prediction quality.

Aggarwal *et al.* (2005a) used a fuzzy-based approach to measure the software maintainability by considering the average number of live variables, average life span of variables, average cyclomatic complexity, and the comments ratio. The fuzzified inputs and outputs are used in experimental setup. The combination of total 81 rules used for making the fuzzy inference system to predict the maintainability. To validate the proposed model, they considered ten software projects of procedure oriented methodology. The selection criteria of the ten projects were based on the defined set of input availability in data set of the projects. By introducing some errors in these projects, the time was measure for maintenance to correct the system. Then the maintainability was measured with the fuzzy model and data sets. They found a strong correlation between maintainability and maintenance time.

Nachiappan and Thomas (2005) analyzed the general regression models and developed a code change based relative measure model for defect density prediction. The presented method was able to predict the faulted files with good level of accuracy. Pizzi and Pedrycz (2006) demonstrated the efficacy of fuzzy integration in combining classification outcomes from multiple classifiers in the discrimination of complexity and maintainability of software objects. The fuzzy integration method produced 15% improvement in predicting software complexity in comparison with the best individual classifier using linear discriminate analysis (0.86 vs 0.75), and 29% improvement in predicting software maintainability (0.75 vs 0.58). These results were achieved using less than a quarter of the original software metrics.

Grover *et al.* (2007) extended the ISO 9126 model to add one sub characteristic, trackability under maintainability and proposed a fuzzy logic based approach to predict the maintainability of the CBS. Authors considered interaction complexity, reusability, testability, understandability and trackability as inputs to predict the maintainability as output. Inputs are designed by fuzzy sets as low, medium and high, while maintainability as very low, low, medium, high and very high. A total of 243 rules were provided to the fuzzy inference engine to get the output. The value of maintainability is measured then by using a defuzzification process. The proposed model is then applied on a classroom-based project to evaluate its maintainability and the results are encouraging. Sharma *et al.* (2009) used fuzzy logic for prediction of maintainability of component based systems. The discussion in this paper is about the maintainability related issues for component based system development and implemented the fuzzy inference system to predict the maintainability as an important quality factor. The influencing factors of maintainability are identified and then grouped into different fuzzy sets. Although, the approach has not been validated on large and complex component based system but authors have successfully validated the presented approach using the classroom project data. The approach is validated using analytical hierarchy process with two case studies, which are based on class room projects. The artificial neural network based approach is considered to be very helpful in estimating/predicting the maintainability of a system. ANN is inspired from the biological nervous system. Georgios (2009) proposed a framework to predict the defects and quality of complex software by narrowing down the set of

dependent metrics to five. The author concluded that a small number of metrics set can also have good correlation with the defect.

Ardil and Sandhu (2010) used branch count, cyclomatic complexity, design complexity, essential complexity and number of lines as inputs and implemented mamdani based fuzzy inference system and ANFIS to predict the maintenance severity of software application. Neuro-fuzzy based implementation is found to be the better than FIS. They concluded that as compared to other machine learning techniques, simple logistic methods and trees perform better with accuracy of 65%, in case of fault prediction. Jianhong *et al.* (2010) proposed ANN based techniques to predict the severity of faults in a software system and the data used from function based system. Using ANN they are able to identify the area of the major faults and need immediate attention for quality and reliability improvements, although the RMSE was higher in the predicted results. Malhotra and Chug (2012) used few machine algorithms to predict the maintainability of the object oriented system. The proposed models are based on group method of data handling, genetic algorithm and probabilistic neural network for prediction of maintainability. Group method of data handling results are better than other techniques in case of maintainability prediction.

6.3 ANN Based Metric to Estimate Maintenance Severity and Maintainability

Artificial neural network is applied to simulate and establish the mapping between the attributes for maintainability and maintenance severity prediction. The different combination of ANN architecture, learning algorithm and number of learning neurons are used to get the best combination.

6.3.1 ANN Architecture

In the implementation of ANN, various architectures have been developed and applied to solve prediction problems. Mainly, the following types of ANN are used in software engineering are as follows:

- Feed forward neural network
- Radial Basis Function (RBF) network

- Kohonen self-organizing network
- Learning vector quantization
- Recurrent neural network

Feed forward back propagation architecture is the most commonly used architecture in software metrics prediction. The two common algorithms have been used for implementation of the proposed concept. The results obtained from both algorithms are compared to choose the best out of the two for prediction of maintenance severity.

6.3.2 Learning Algorithms

Different basic learning algorithms can be used to implement the learning process of ANN such as *trainlm, trainbr, trainbfg, traincgb, traincgf, traincgp, traingd, traingdm, traingdx, traingda, trainoss, trainr, trainrp, trainscg.*

The most commonly used learning algorithm in ANN to analyze the software metrics are *trainlm* and *trainbr.* The *trainlm* is a Levenberg-Marquardt back propagation algorithm. The network training algorithm *trainlm* is a function which updates the weights and bias values using Levenberg-Marquardt optimization. It is oftenly used algorithm in estimation applications. It is generally first choice for supervised learning algorithms. It is the fastest but need more memory than other algorithms. The *trainlm* function takes the different inputs such as neural network, initial training record, training data, validation data and test data.

The *trainbr* is a Bayesian regulation back propagation algorithm. Although, it uses the same inputs and Levenberg-Marquardt optimization similar to *trainlm,* but it uses Bayesian regularization. Bayesian regularization process minimizes a combination of squared errors and a weight, in turn, determines the correct combination so as to produce a network that generalizes well. Both learning algorithms have been used in implementation of metrics models to find the algorithm which provides the best results with the available data sets.

6.3.3 Predicting the Maintenance Severity

To distribute the efforts and cost across the modules and components' development plan, it is necessary to predict the maintenance severity at module level

granularity. The prediction of maintenance severity input can play an important role for software reliability and quality improvements. The maintenance severity is inversely proportional to maintainability of the system. If, the maintenance severity is low, then the maintainability of the module or component is considered to be high.

The first step to establish the metric is to identify set of factors which impact the maintenance severity. The second step is to establish a relationship between them to estimate the output metric. In the proposed methodology, we have analyzed the factors influencing the maintenance severity and artificial neural network is used to establish the relationship between the six identified metrics. In Chapter 3, Chapter 4 and Chapter 5, we have used fuzzy, ANN, neuro-fuzzy approaches, and ANN proved to be a simple and efficient technique for metrics attribute prediction. Therefore, we have thoroughly investigated the ANN approach for maintenance severity prediction with different ANN algorithms and neurons frequency in ANN training and simulation.

6.3.3.1 Identified Input Factors

To identify the most influencing factors, we analyzed the various data sets (Menzies *et al.* 2012). The focus was on the two factors while identifying the set of influencing metrics, which are easily collected during the development process and have more correlation to the maintenance severity of the software module or component in the data set (Menzies *et al.* 2012).

- *Halstead difficulty:* It is Halstead difficulty metric of a module or component. Halstead difficulty metric is calculated as follows:

  n_1 = The count of distinct operators

  n_2 = The count of distinct operands

  N_1 = Total number of operators

  N_2 = Total number of operands

  Then, the program vocabulary n = $n_1 + n_2$, program length N = $N_1 + N_2$

  Then the difficulty D is calculated as follows:

$$D = \frac{n_1}{n_2} \times \frac{N_1}{N_2}$$

- *Multiple condition count:* It is a count of the multiple conditions extracted from module and component structure.
- *Decision count:* It is counted as number of decision points in a module or component.
- *Cyclomatic complexity:* It is developed by Thomas J. McCabe, Sr in 1976. The McCabe's cyclomatic complexity of the module is calculated using the following formula:

$$V(G) = E - N + 2P,$$

where, E are the number of edges of the graph. N are the number of nodes in the graph. P are the number of connected components i.e. exit nodes.

- *Design complexity:* It is design complexity of a module or component.
- *TLOC:* This is size metrics which is the count of number of lines in a module or component.

6.3.3.2 Maintenance Severity Metrics

To design the maintenance severity metrics, the six independent variables are considered, These are easily available, collected and calculated during the SDLC. This minimizes the efforts and cost of implementation, also increases the practicality and applicability in software industries. Six factors as defined in the earlier section are considered as deciding factors to estimate the component's or module's maintenance severity. ANN algorithm architecture and algorithms are built to estimate the maintenance severity of a module or component, which can be aggregated for maintenance severity of the entire component based or structured based software systems.

Let x_n, y_n, z_n, a_n, b_n, c_n be the values of the halstead difficulty, multiple condition count, decision count, cyclomatic complexity, design complexity and TLOC, respectively, for the n^{th} data record in the data set. Then, the ANN model corresponding to this approach is described in Fig 6.1.

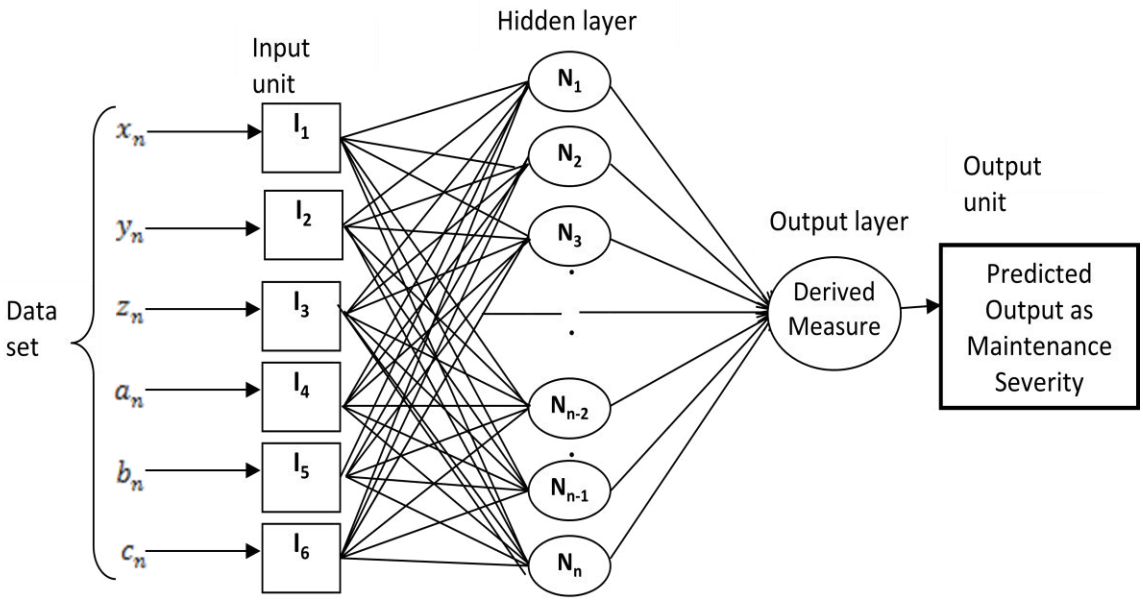


Fig. 6.1: Proposed Layers and Units in ANN Architecture

In the implementation of the above relation, *nntool* of MATLAB is used. The results of the *nntool* and RMSE are shown in Table 6.2, Table 6.3 and Table 6.4. The same process has to be repeated for each module of the component or system. MATLAB tool is used for experimentation of the established relationship in the above section. The following combination of neural network architecture, neurons count and learning algorithms are used for experimentation, training, testing and validation.

Table 6.1: Network Architecture for Maintenance Severity

Network Type	Feed-Forward Back propagation
Training Function	*trainlm,trainbr*
Adaptive Learning Function	*learngdm*
Performance Function	RMSE
Number of Layers	02
Number of Neurons	5,10,15,20
Transfer Function	*tansig*
Error	0.00005
Epoch	Maximum 1000

6.3.3.3 ANN Training and Validation Results

MATLAB is used to build ANN model which is trained using the data from three projects. Various combinations are considered to find the best results. RMSE is calculated from the actual maintenance severity values and predicted values from the proposed approach. The least RMSE is the criteria of the best predictor. Three projects PC5, PC4, PC2 data from PROMISE repository of empirical software engineering data (Menzies *et al.* 2012) is used for validation and training.

The data sets from project PC5 are used for the training of the proposed ANN model. Fig 6.2, Fig 6.3 and Fig 6.4 show the *trainbr* training performance, states and regressions, respectively.

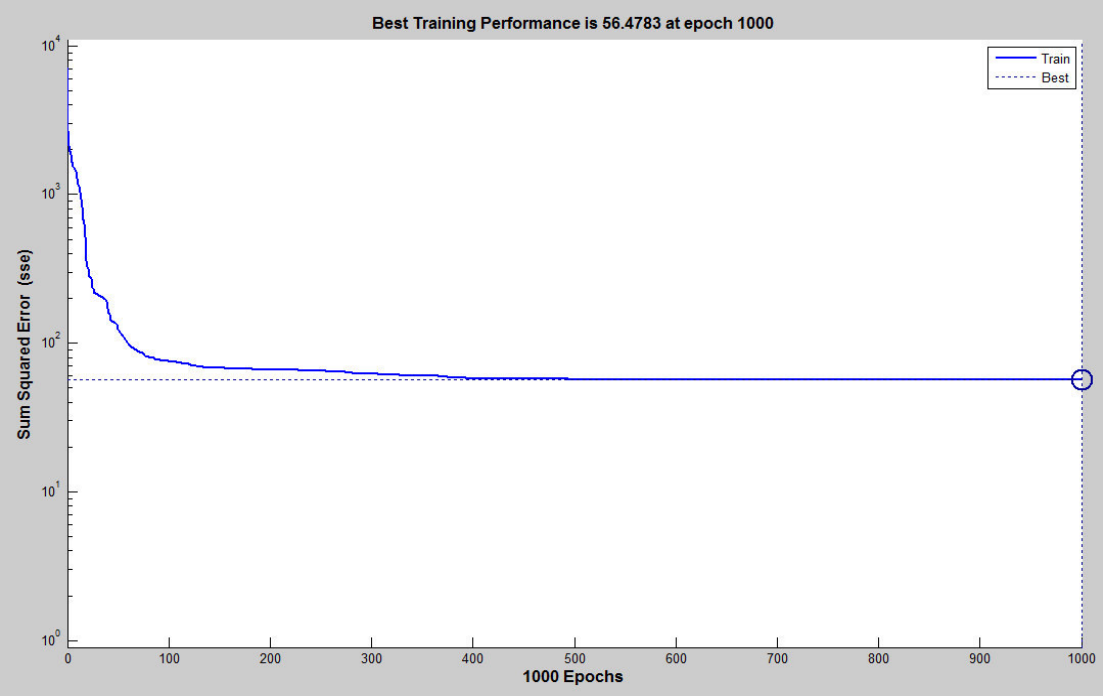


Fig. 6.2: ANN Performance Results with *trainbr* and 15 Neurons Using Maintenance Severity Data

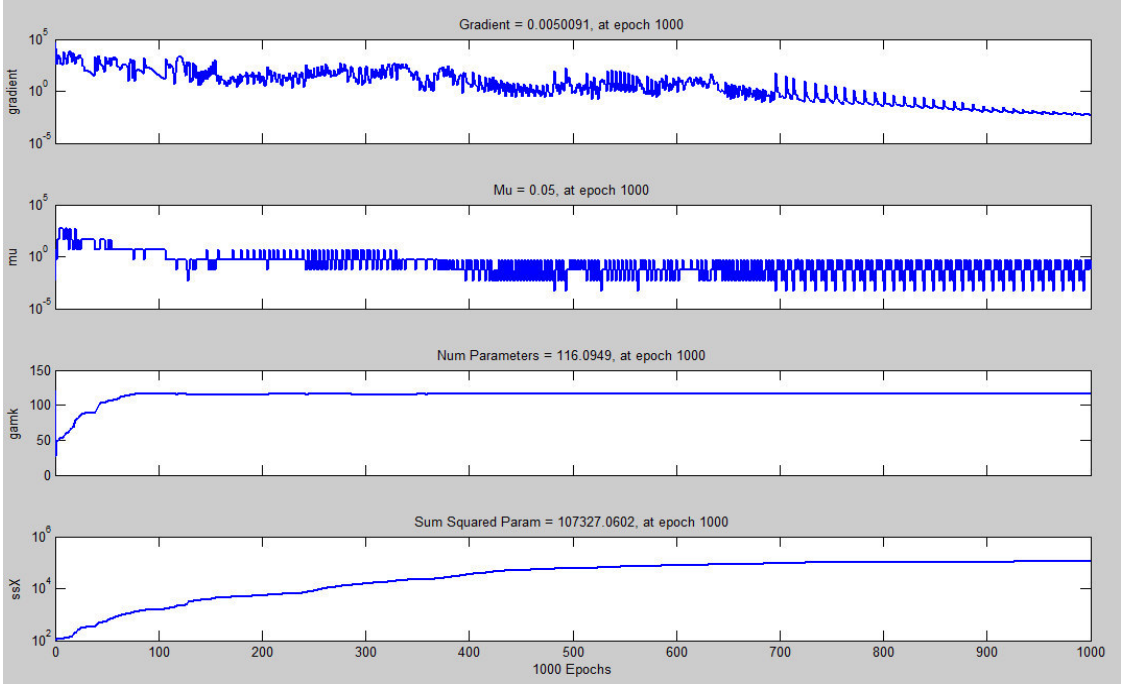


Fig. 6.3: ANN Training State Results with *trainbr* and 15 Neurons Using Maintenance Severity Data

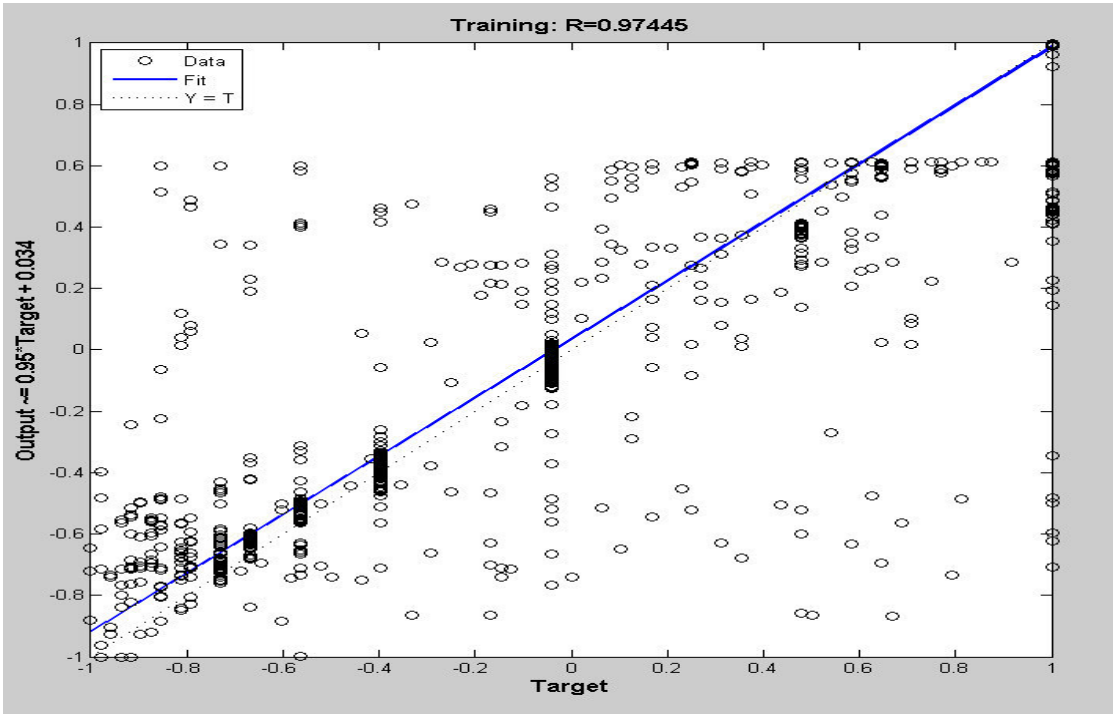


Fig. 6.4: ANN Regression Results with *trainbr* and 15 Neurons Using Maintenance Severity Data

Fig. 6.2 illustrates the performance of 1000 epochs in terms of the sum squared errors. Fig. 6.3 shows the training states of 1000 epochs. Fig. 6.4 depicts the regression results of training during the ANN training, validation and testing.

The Fig. 6.5 illustrates the difference between the predicted and actual maintenance severity of 100 randomly selected modules. It can be observed from the graph that in most of the modules the predicted values matched with the actual project data values.

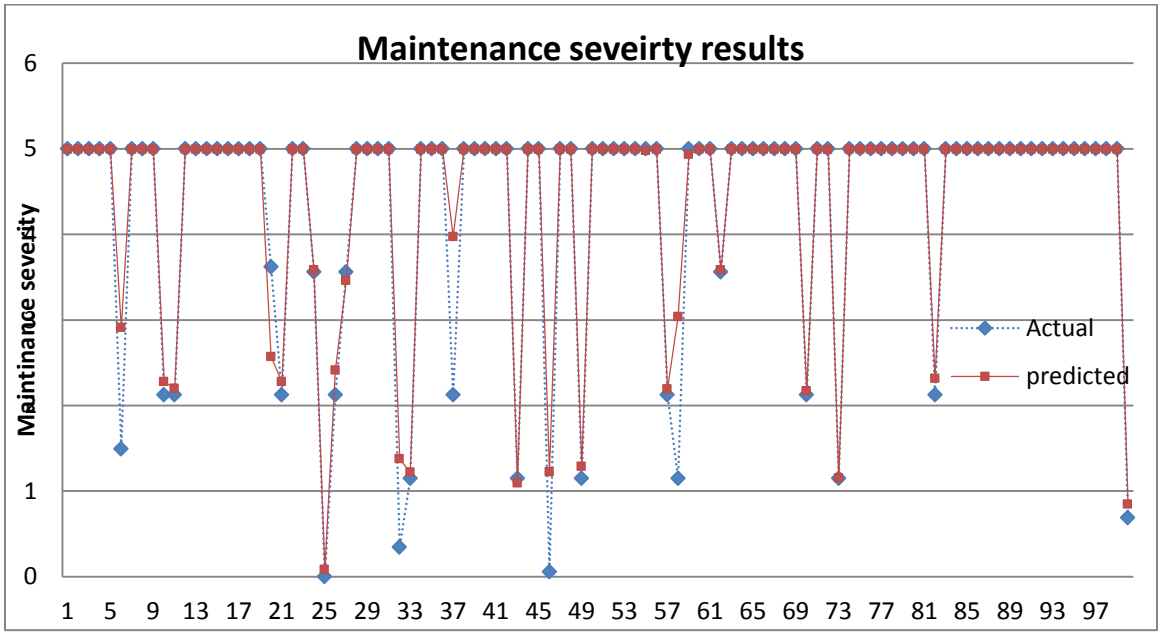


Fig. 6.5: Predicted Vs Actual Maintenance Severity of 100 Modules

Table 6.2: RMSE Results of Project P_1 Data Results for Maintenance Severity

Project P_1		
	RMSE with FFBP NN algorithm	
Network Neurons	*trainlm*	*trainbr*
5	0.1188	0.1045
10	0.1311	**0.0992**
15	0.1084	0.1065
20	0.1350	0.1122

Table 6.3: RMSE Results of Project P_2 Data Results for Maintenance Severity

Project P_2		
	RMSE with FFBP NN algorithm	
Network Neurons	*trainlm*	*trainbr*
5	0.1459	0.1285
10	0.1486	**0.1230**
15	0.1241	0.1527
20	0.1542	0.1706

Table 6.4: RMSE Results of Project P_3 Data Results for Maintenance Severity

Project P_3		
	RMSE with FFBP NN algorithm	
Network Neurons	*trainlm*	*trainbr*
5	0.1439	0.1253
10	0.0893	**0.0652**
15	0.0804	0.0697
20	0.1282	0.0670

The best results are achieved with *trainbr* and 10 neurons as 0.0992, 0.1230 and 0.0652 for project P_1, P_2 and P_3, respectively.

6.3.4 Estimating Maintainability

The maintainability of a software system cannot be measured directly. It is an indirect measure and hence needs to establish the relationship between the metrics and attributes to estimate it. There are various conventional and statistical method exists to compute the maintainability of the modules or components. The commonly used maintainability index is calculated using the cyclomatic complexity, halstead volume and lines of code in module or system.

The main issue in the existing maintainability estimation techniques is that it uses the complex derived measure such as complexity and halstead volume. ANN is used to build the relationship between the simplest software attributes for the better applicability and adaptation in the industries.

6.3.4.1 Identified Input Factors

To build the maintainability metric, the four simple measures are identified. The selected four input metrics can be easily collected from a module by analyzing the code.

- *Multiple condition count:* It is the count of multiple conditions in the program. As multiple conditions are nested in the code, the maintainability will tend to decrease.
- *Node count:* It is simple count of the nodes in a given module. If, the node count increase and cross a threshold count, it will yield to a complex module which can be difficult to maintain.
- *Percentage comments:* It is the percentage of comments that have been incorporated in the code with respect to the amount of code. If the comments are more, it will increase the understandability, in turn, it will ease the maintenance.
- *Total lines of code:* It is the total number of lines present in the module programs excluding the comments.

6.3.4.2 ANN Application in Metric Design

To build the maintainability prediction model, four simple metrics are identified. These four metrics can be easily computed from the source code of modules.

This minimizes the efforts and cost of implementing the approach, also increases the practicality and applicability in software industries. Four factors described in the earlier section are considered as deciding factors to estimate the component's or module's maintainability. ANN algorithm architecture and algorithms are built to estimate the maintainability of a module or component, which can be aggregated for maintainability of the entire component based or structured based software systems.

Let x_n, y_n, z_n, a_n be the values of the multiple condition count, node count, percentage comments and TLOC respectively, then

$$MI_m(R_1, m) = f(x_n, y_n, z_n, a_n), \qquad (6.3)$$

here, x_n, y_n, z_n, a_n are value of the n^{th} row in data set of independent attribute in a module m. $MI_m(R_1, m)$ is the maintainability index of a module m in a release R_1.

By taking the sum of maintainability index of each module, the system's average

maintainability can be calculated for better planning and process improvements. Here, the function f given in equation (6.3) is being implemented using the ANN approach as explained in next section.

In the experimental design of ANN approach, f is considered as ANN learning algorithmic function. The maintainability index of a module m in release R_1, using ANN function can be derived as:

$$MI_m(R_1, m) = O = f(net) = f\left(\sum_{n=1}^{m} w_{x_n} x_{m_n}, w_{y_n} y_{m_n}, w_{z_n} z_{m_n}, w_{a_n} a_{m_n}\right), \quad (6.5)$$

here, w_{x_n}, w_{y_n}, w_{z_n}, w_{a_n} are the weight vector for multiple condition count, node count, percentage comments and TLOC, respectively. To implement the above relation, *nntool* of MATLAB is used. RMSE is calculated using the results of the *nntool* and actual maintainability of modules. The same process has to be repeated for each module of the component or system.

The data from three projects PC5, PC4, PC2 from PROMISE repository of empirical software engineering data (Menzies *et al.* 2012) is used for validation and training. MATLAB tool is used for experimentation of the established relationship given in equation (6.5). The combination of neural network architecture, neurons count and learning algorithms are used for experimentation, training, testing and validation. Table 6.5 captures the architecture and ANN attributes used in experimentation.

Table 6.5: ANN Architecture Attributes Used in Training and Testing

Network Type	Feed-Forward Back propagation
Training Function	*trainlm,trainbr*
Adaptive Learning Function	*learngdm*
Performance Function	RMSE
Number of Layers	02
Number of Neurons	5,10,15,20, 25
Transfer Function	*tansig*
Error	0.00005
Epochs	Maximum 1000

The two learning algorithms *trainlm* and *trainbr* are used for the training and testing of the proposed metrics concept. Fig. 6.5, Fig. 6.6 and Fig. 6.7 are the screenshot of ANN performance, training states and regression results, respectively, while the ANN is trained using *trainlm* with 15 neurons in MATLAB. To find the better combination of ANN attributes in maintainability prediction, *trainbr* is also explored. Fig. 6.8, Fig. 6.9 and Fig. 6.10 illustrate the training performance, states, and regression results respectively using *trainbr* with 15 neurons.

When the combination of *trainlm* and 15 neurons are used, Fig. 6.6 illustrates the performance of 1000 epochs in terms of the sum squared errors. Fig. 6.7 shows the training states of 1000 epochs using the *trainlm* and 15 neurons. Using *trainlm* and 15 neurons, Fig. 6.8 depicts the regression results of training during the ANN training, validation and testing. If, combination of *trainbr* and 15 neurons are used, Fig. 6.9 illustrates the performance of 1000 epochs in terms of the sum squared errors. Fig. 6.10 shows the training states of 1000 epochs using the *trainbr* and 15 neurons. Using *trainbr* and 15 neurons, Fig. 6.11 depicts the regression results of training during the ANN training, validation and testing.

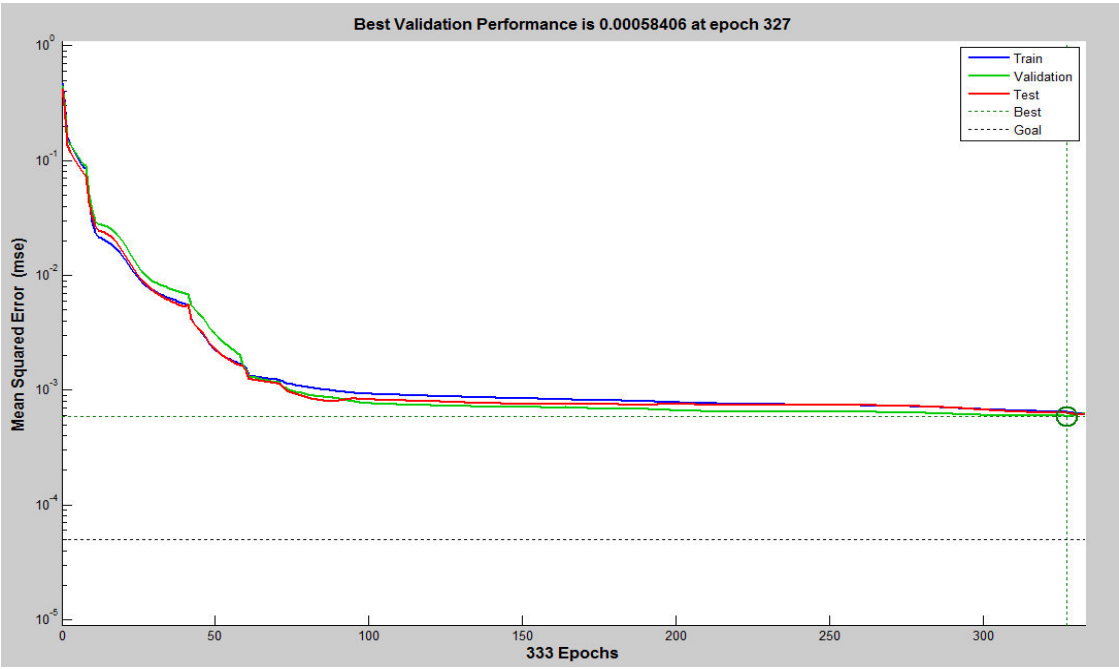


Fig. 6.6: ANN Performance Results with *trainlm* and 15 Neurons using Maintainability Data

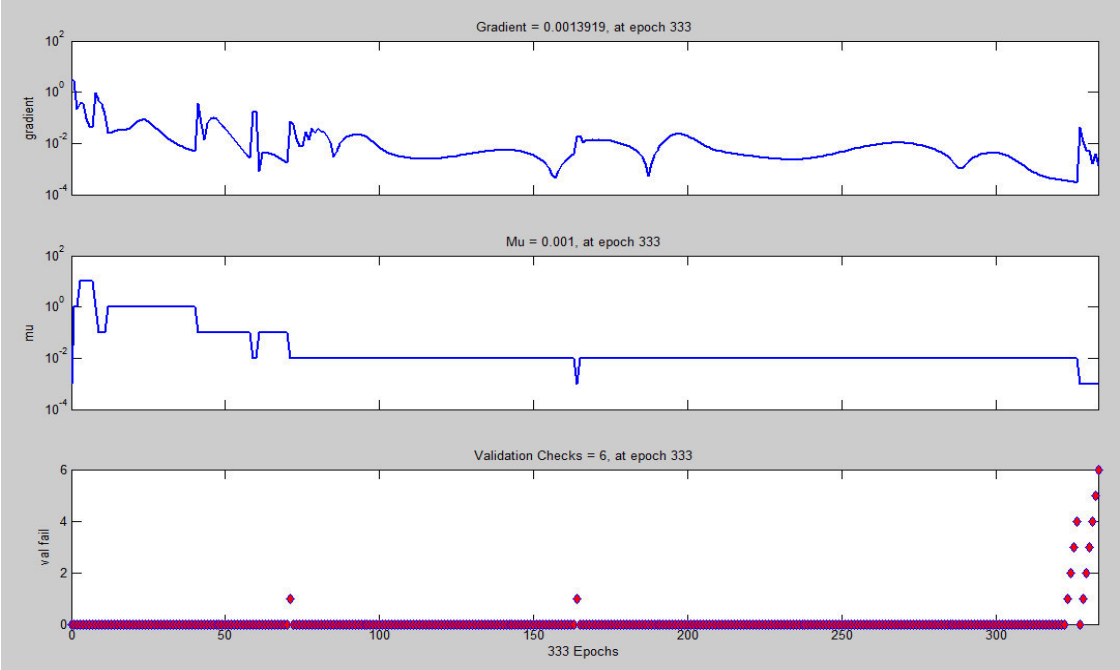


Fig. 6.7: ANN Training States with *trainlm* and 15 Neurons Using Maintainability Data

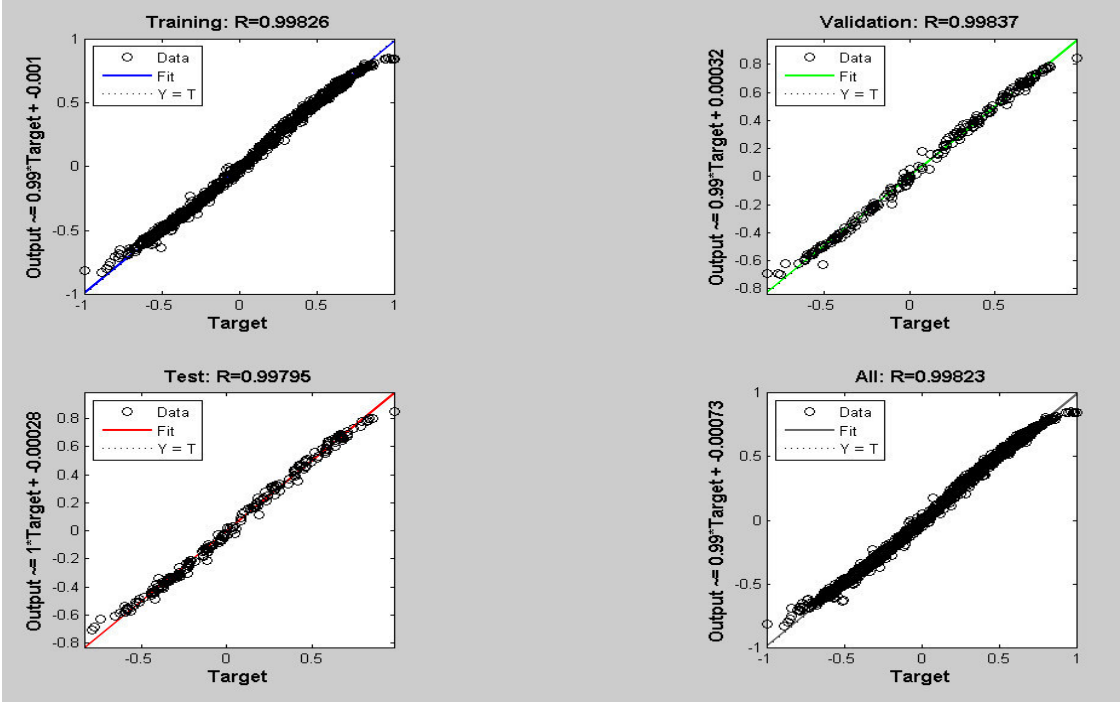


Fig. 6.8: ANN Regression Results with *trainlm* and 15 Neurons Using Maintainability Data

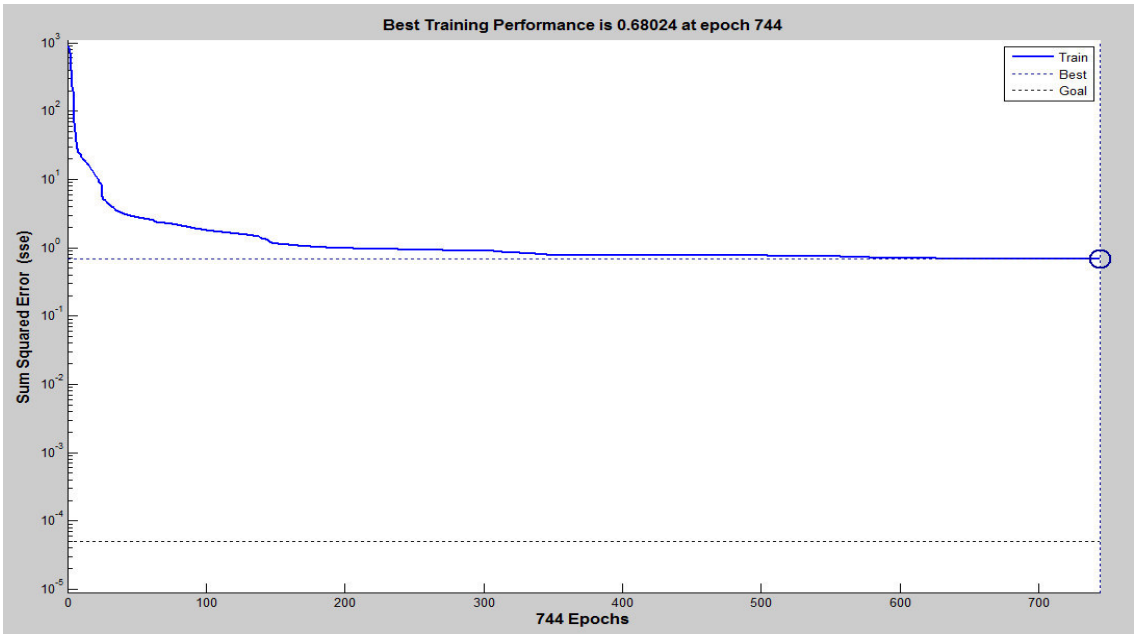


Fig. 6.9: ANN Performance Results with *trainbr* and 15 Neurons Using Maintainability Data

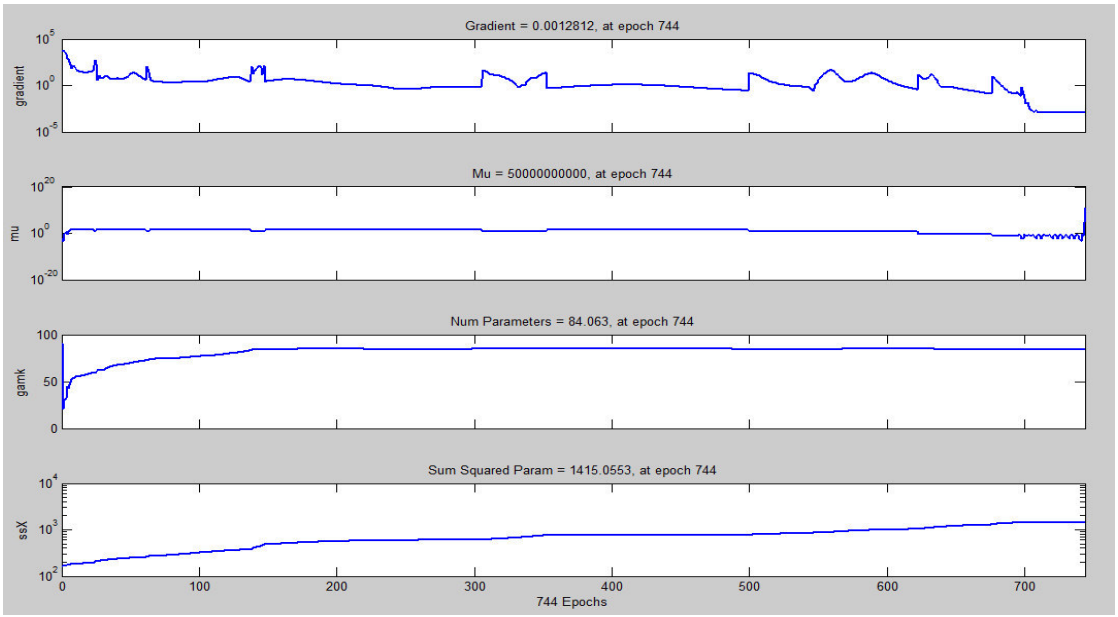


Fig. 6.10: ANN Training States with *trainbr* and 15 Neurons Using Maintainability Data

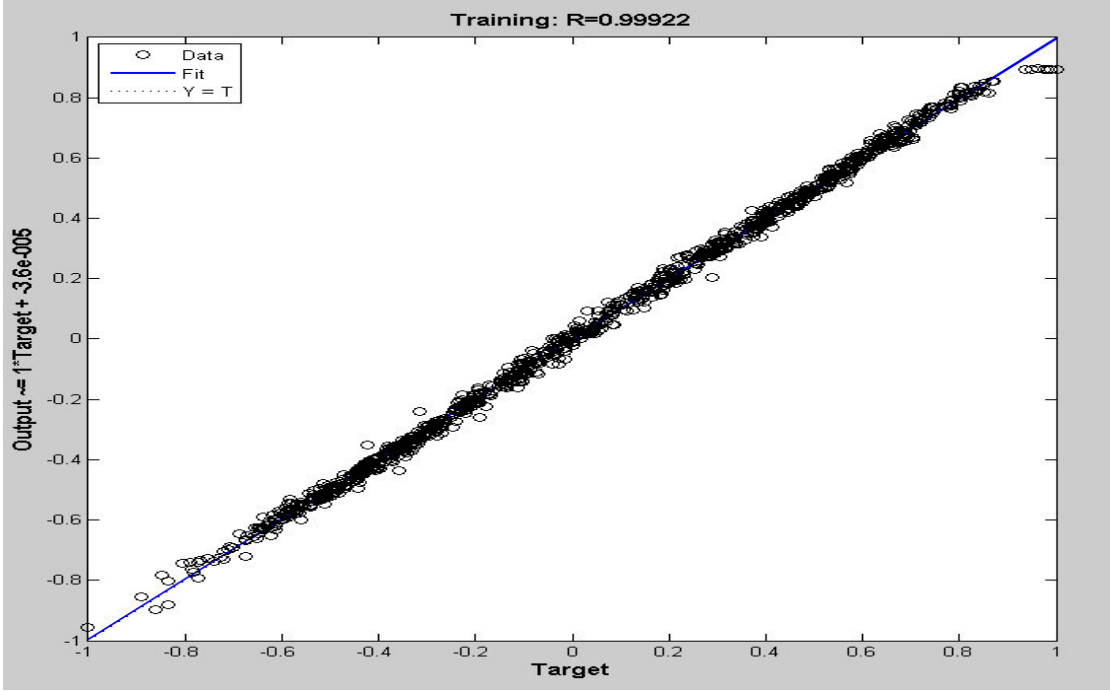


Fig. 6.11: ANN Regression Results with *trainbr* and 15 Neurons Using Maintainability Data

6.3.4.3 Validation

The maintainability prediction metric is validated on three projects PC5, PC4, PC2 data from PROMISE repository of empirical software engineering data (Menzies *et al.* 2012). For each project, predicted maintainability values from the presented ANN model are captured as an output from MATLAB experimentation. The RMSE is calculated using predicted and actual maintainability index values. The best RMSE results are achieved with 15 neurons and *trainbr* learning algorithm for project P_1, P_2 and P_3. This combination gives the RMSE as 0.1134, 0.0870 and 0.1683 for project P_1, P_2, P_3, respectively.

Table 6.6: Results of Maintainability Validation Results with Project P_1 Data Using ANN

Project P_1		
	RMSE with FFBP NN algorithm	
Network Neurons	*trainlm*	*trainbr*
5	0.1507	0.1158
10	0.1735	0.1297
15	0.1235	**0.1134**
20	0.1780	0.1738
25	0.1664	0.1600
30	0.1737	0.1696

Table 6.7: Results of Maintainability Validation Results with Project P_2 Data Using ANN

Project P_2		
	RMSE with FFBP NN algorithm	
Network Neurons	*trainlm*	*trainbr*
5	0.1155	0.0909
10	0.1131	0.0872
15	0.0978	**0.0870**
20	0.1267	0.1000
25	0.1128	0.1092
30	0.1192	0.1245

Table 6.8: Results of Maintainability Validation Results with Project P_3 Data Using ANN

Project P_3		
	RMSE with FFBP NN algorithm	
Network Neurons	*trainlm*	*trainbr*
5	0.2067	0.1707
10	0.1780	0.1688
15	0.1889	**0.1683**
20	0.1808	0.1909
25	0.2104	0.1905
30	0.2073	0.2015

6.4 Conclusion

The proposed maintenance severity metric is able to predict the maintenance severity of module or component with good accuracy. The maintenance severity is predicted by using six metrics from a modules or components. The model is implemented using artificial neural network techniques of soft computing. The different algorithms and attributes are explored from available ANN architecture. The conclusion of better approach is drawn on the basis of on the least RMSE of the combination of ANN architecture, learning algorithm and neurons frequency. In case of maintenance severity prediction, the best result is achieved with 10 neurons in FFBP algorithm and *trainbr* learning algorithm. The maintenance severity is necessary to identify the need of next maintenance release. Maintainability is predicted using ANN and four simple software program attributes. In maintainability prediction, best results are achieved with *trainbr* and 15 neurons for FFBP architecture. The predicted maintenance severity is useful to allocate the time, cost and efforts for the maintenance release. The focus can also be diverted towards the most maintenance severe components or module. A component may have several modules, and with the help of maintenance severity prediction, only few modules which are having high maintenance severity will have to be enhanced and changed in complete component. Less severe modules may not need high level of attention; it will save the schedule and cost as well as provide the insight information to improve the quality of the component or module. The maintainability prediction will help the managers to focus on the modules, which are having less maintainability index as predicted using the proposed approach.

Chapter 7

Conclusion and Future Scope

7.1 Conclusion

The aim of the study was to design and analyze some software metrics using software computing techniques for different software engineering paradigms as per applicability. In the proposed research work, fuzzy logic, artificial neural network and neuro-fuzzy techniques have been applied to design different metrics for component, structural and object oriented software engineering approaches. The soft computing based software metrics and approaches have been proposed for prediction of software defects, reusability, maintenance severity, maintainability and reliability. The proposed approaches are defined for the appropriate software engineering paradigms as per importance and need. The proposed defect density and reliability metric have been validated using the 3 releases of two complex and large telecommunication product development projects. The cross-project validation is carried out for maintenance severity and maintainability metrics using the data from three projects. The reusability metric is validated using the data from 80 components. The major findings of this study have been

summarized briefly and some pointers to future research in this direction have been described in this chapter.

Defect density metric has been designed and described in Chapter 3. This metric is designed using the complexity, size and defects before release. Two main soft computing approaches such as fuzzy logic, artificial neural network and neuro-fuzzy are applied to build the framework. The validation is performed using the real project from two software products. These two software product produces several releases during the software development. The defect metric is designed using the three independent factors for the structural, object oriented and component based software systems. The proposed metric is applicable on the basis of availability of three independent factors. The performance analysis is done using the root mean square error calculation of results from fuzzy logic and artificial neural network approaches. The analysis and discussion is described by comparing the two methodologies. The study and analysis shows that the three dependent factors are very strongly correlated to the defects. This relationship is demonstrated quantitatively with experiment using MATLAB tool.

In Chapter 4, the quality and reliability management framework using the defect prediction has been discussed. In the proposed research, the reliability factor is calculated based on the predicted defects, which are independent of structural, object or component based development. The Reliability factor is calculated using predicted defects. Soft computing approaches such as fuzzy logic and artificial neural network are being applied in prediction of defects across software releases. The presented approach gives the chance to software quality assurance and management team to assess the resource, cost and schedule based on the reliability growth model curve, which is represented using the predicted defect after every phase of testing and integration cycle. The predicted reliability factor can play an important role to decide the release maturity after every testing phase. Using the defects and reliability prediction, the reliability and quality management framework is build for the application across the software releases. The release management framework concept may look obvious but this is entirely new concept and has a high possibility for application and adaptation in software industries. This concept has been validated on the data set from multiple releases of two large and

complex software products. The concept is quantitatively analyzed and proved with the performance results using RMSE calculations.

In Chapter 5, the exhaustive analysis of component reusability aspects have been discussed. Here, reusability metric and assessment framework for components have been presented using the soft computing techniques. Three principal constitutes of soft computing techniques such as fuzzy logic, artificial neural network and adaptive fuzzy inference system have been applied for prediction of the component reusability. On the basis of correlation analysis of the data, six most influencing software attributes are extracted which play important roles to decide the reusability of the software components. These attributes are release version, existing defects, portability, interface complexity, customizability and understandability. Several real-life components are used for the training and testing of soft computing techniques. The validation is done using the components data, and the quantified results show that adaptive fuzzy inference system performs better than other two soft computing approaches in terms of RMSE calculations. The proposed approaches can be adopted very easily by the software industries as well as by application developers to select the best reusable components from the different available options of components. In the proposed approaches, input factors have been used considering main influencing aspects of reusability. It will lead to the better reliability, quality and maintainability of the software product where these highly reusable components are being used.

Chapter 6 explains the maintenance severity as an important factor for software maintainability and quality. It proposes the maintenance severity and maintainability prediction using artificial neural network. Six metrics have been analyzed and used to predict the maintenance severity at the module level of the software system. The proposed approach is independent of software engineering methodology used for software development. The data for six independent factors such as halstead difficulty, multiple condition count, decision count, cyclomatic complexity, design complexity and lines of count should be collected for component based, structural or object oriented software development strategy. The maintainability prediction metric is designed using four simple metrics, multiple condition count, node count, percentage comments and lines code. These can be easily collected from the program code. The neural network

approach has been used in other chapters also but to predict the maintainability, neural network application is applied and analyzed with different number of neurons, learning algorithm and network architecture in consideration of enough data availability for training and testing. The results from two different learning algorithms, *trainlm* and *trainbr* are compared. The training and testing is done with 5, 10, 15, 20 neurons for particular ANN architecture and learning algorithm. Results are very encouraging and suggest for use of the neural network for prediction problem in software engineering.

7.2 Future Scope

The important quality metrics such as defect density, reliability, reusability, maintainability, reliability are estimated and validated using the three principal constitute of soft computing paradigms. It is a big challenge to get the data for cross validation of the approaches due to security issues and financial implications of software industries. Considering these facts, to get the attributes' data of more components is very difficult. Although, in Chapter 5, we have made a good attempt to validate the proposed reusability metric using the data of 80 components, but in case of more components' data availability, the results may be improved. In our proposed metric, the results got from the study are very interesting, and indicate a hope that industries will implement in their software improvement processes.

References

1. Abraham, A., 2005. Artificial Neural Networks - Handbook for Measurement Systems Design, John Wiley and Sons Ltd., London.
2. Aggarwal, K. K., Singh, Y., Chandra, P., Puri, M., 2005. Measurement of Software Maintainability Using a Fuzzy Model, Journal of Computer Sciences, Vol. 1, pp: 538-542.
3. Aggarwal, K. K., Singh, Y., Kaur, A., Malhotra, R., 2005. Software Reuse Metrics for Object-Oriented Systems, Proceedings of the Third ACIS International Conference on Software Engineering Research, Management and Applications, pp: 48-55.
4. Aggarwal, K. K., Singh, Y., Kaur, A., Malhotra, R., 2006. Application of Artificial Neural Network for Predicting Maintainability Using Object-Oriented Metrics, Transactions on Engineering, Computing and Technology, Vol. 15, pp: 285-289.
5. Aggarwal, K. K., Singh, Y., Chandra, P., Puri, M., 2005. Sensitivity Analysis of Fuzzy and Neural Network Models, ACM SIGSOFT Software Engineering Notes, Vol. 30, Issue 4, pp: 1-4.
6. Ahn, Y., Suh, J., Kim, S., Kim, H., 2003. The Software Maintenance Project Effort Estimation Model Based on Function Points, Journal of Software Maintenance: Research and Practice, Wiley Online, Vol. 20, pp: 71-85.
7. Albrecth, Gaffney, 1983, Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation, IEEE Transactions on Software Engineering, Vol. 9, Issue 6, pp: 639-648.
8. Andrew, R. G., Stephen, G. M., 1999. Fuzzy Logic for Software Metric Models Throughout the Development Life-Cycle, Proceeding of Fuzzy Information Processing Society and 18th International Conference of the North American, pp: 258-262.

9. Ardil, E., Sandhu, P. S., 2010. A Soft Computing Approach for Modeling of Severity of Faults in Software Systems, International Journal of Physical sciences, Vol. 4, pp: 74-85.

10. Ardimento, P., Bianchi, A., Visaggio, G., 2004. Maintenance-oriented Selection of Software Components, Proceedings of 8th IEEE European Conference on Software Maintenance and Re-engineering, Tampere, Finland, pp: 115-124.

11. Ayse, T., Burak, T., Ayse, B., 2008. Ensemble of Software Defect Predictors: A Case Study, Proceeding of Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, pp: 318-320.

12. Bai, C., Cai, K., Hu, Q., Ng, S., 2008. On the Trend of Remaining Software Defect Estimation Systems, Man and Cybernetics, IEEE Transactions on Systems and Humans, Vol. 38, Issue: 5, pp: 1129-1142.

13. Baisch, E., Bleile, T., Belschner, R., 1995. A Neural Fuzzy System To Evaluate Software Development Productivity, Proceeding of IEEE International Conference on Systems, Man and Cybernetics, pp: 4603-4608.

14. Balikuddembe, J. K., Osunmakinde, I. O., Bagula, A., 2009. Software Project Profitability Analysis Using Temporal Probabilistic Reasoning - An Empirical Study with the CASE Framework, Advances in Security Technology Communications in Computer and Information Science, Vol. 29, pp: 138-150.

15. Bandini, S., Paoli, F. D., Manzoni, S., Mereghetti, P. A, 2001. Support System to COTS-based Software Development for Business Services. Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, Ischia, Italy, Vol. 27, pp: 307-314.

16. Banker, R. D., Kauffman, R. J., Wright, C. R., Zweig, D., 1991. Automating Output Size and Reusability Metrics in an Object-Based Computer Aided Software Engineering (CASE) Environment, Information Systems Working Papers Series, pp: 23-29.

17. Barnard, J., 1998. A New Reusability Metric for Object-Oriented Software,

Software Quality Control, Vol. 7, Issue 1, pp: 35-50.

18. Basili, V. R., Rombach, H. D., 1988. Towards A Comprehensive Framework for Reuse: A Reuse-Enabling Software Evolution Environment, Technical Report CS-TR-2158, Department of Computer Science, University of Maryland, College Park, MD 20742.

19. Beizer, B., 1990. Software Testing Techniques, Van Nostrand Reinhold, New York.

20. Beldjehem, M., 2010. A Unified Granular Neuro-fuzzy Framework for Predicting and Understanding Software Quality, International Journal of Software Engineering and Its Applications, October, Vol. 4, pp: 17-35.

21. Bhattacherjee, V., 2006. The Soft Computing Approach to Program Development Time Estimation, Proceeding of 9th IEEE International Conference on Information Technology, pp: 291-292.

22. Boehm, B., 1981. Software Engineering Economics, NJ: Prentice- Hall.

23. Booch, G., 1994. Object-Oriented Analysis and Design, with Applications, 2nd edition, Benjamin/ Cummings, San Mateo, CA.

24. Boxall, M. A. S., Araban, S., 2004. Interface Metrics for Reusability Analysis of Components. Proceedings of Australian Software Engineering Conference, Melbourne, Australia, pp: 40–46.

25. Burgin, M., Debnath, N., Debnath, J., 2006. Fuzziness and Imprecision in Software Engineering, Proceeding of Automation Congress, pp: 1-8.

26. Cai, K. Y., 1998. Software Defect and Operational Profile Modeling. Boston, MA: Kluwer Academic.

27. Cangussu, J. W., Karcich, R. M., Mathur, A. P., DeCarlo, R. A., 2004. 15th International Symposium on Software Reliability Engineering, pp: 440-450.

28. Card, D. N., 2002. Managing Software Quality with Defects, Proceeding of

Computer Software Application Conference, August, pp: 472–474.

29. Challagulla, V. U. B., Bastani, F. B., Paul, I. Y., 2005. Empirical Assessment of Machine Learning Based Software Defect Prediction Techniques, 10th IEEE International Workshop on Object- Oriented Real-Time Dependable Systems, WORDS, Feb, pp: 263-270.

30. Chang, C. W., Wu, C. R., Lin, H. L., 2008. Integrating Fuzzy Theory and Hierarchy Concepts to Evaluate Software Quality, Software Quality Journal, Vol. 16, Issue 2, pp: 263-27

31. Chidamber, S., Kemerer, C., 1991. Towards a Metrics Suite for Object Oriented design. Proceeding of Conference on Object-Oriented Programming: Systems, Languages and Applications, SIGPLAN Notices, Vol. 26, pp: 197-211.

32. Cho, E. S., Kim, M. S., Kim, S. D., 2001. Component Metrics to Measure Component Quality, Proceedings of 8th Asia-Pacific Software Engineering Conference, Macau, pp: 419-426.

33. Dahiya, S. S., Chhabra, J. K., Kumar, S., 2007. Use of Genetic Algorithm for Software Maintainability Metrics' Conditioning, International Conference on Advanced Computing and Communication, pp: 87-92.

34. Dapeng, L., Shaochun, X., 2007. New Quality Metrics for Object-Oriented Programs, Proceeding of Eighth IEEE International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, pp: 870-875.

35. David, N. C., 2002. Managing Software Quality with Defects, Proceedings of the 26th International Computer Software and Applications Conference, IEEE Computer Society. pp: 472-474.

36. Devanbu, P., Karstu, S., Melo, W., Thomas, W., 1996. Analytical and Empirical Evaluation of Software Reuse Metrics, Proceeding of 18th International Conference on Software Engineering, Berlin, Germany, pp: 189-199.

37. Dick, S., Sadia, A., 2006. Fuzzy Clustering of Open-Source Software Quality Data: A Case Study of Mozilla, International Joint Conference on Neural Networks, pp: 4089-4096.

38. Dumke, R., Schmietendorf, A., 2000. Possibilities of the Description and Evaluation of Software Components. Metrics News, Vol. 5, pp: 13-26.

39. El-Emam, K., 2001. Ethics and Open Source, Empirical Software Engineering, Vol. 6, Issue 4, pp: 291-292.

40. Fenton, N. E., Neil, M., 1999. A Critique of Software Defect Prediction Models. IEEE Transactions on Software Engineering, Vol. 25, pp: 675-689.

41. Fioravanti, F., Nesi, P., 2001. Estimation and Prediction Metrics for Adaptive Maintenance Effort of Object-Oriented Systems. IEEE Transactions on Software Engineering, Vol. 27, pp: 1062-1084.

42. Frakes, W., Terry, C., 1996. Software Reuse: Metrics and Models, ACM Computing Surveys, Vol. 28, Issue 2, pp: 415–435.

43. Frakes, W. B., Tortorella, M., 2008. Foundational Issues in Software Reuse and Reliability, International Doctoral Symposium on Empirical Software Engineering, pp: 23-28.

44. Furulund, K. M., Molokken-Ostvold, K., 2007. Increasing Software Effort Estimation Accuracy-Using Experience Data, Estimation Models and Checklists, Proceedings of QSIC 2007, pp: 342-347.

45. Gao, K., Khoshgoftaar, T. M., Wang, H., Seliya, N., 2011. Choosing Software Metrics for Defect Prediction: An Investigation on Feature Selection Techniques, Software: Practice and Experience, Vol. 41, Issue 5, pp: 579-606.

46. Georgios, L., 2009. Software Metrics Suites for Project Landscapes, Proceeding of IEEE European Conference on Software Maintenance and Reengineering, pp: 317-318.

47. Gill, N. S., 2003. Reusability Issues in Component-based Development, ACM

SIGSOFT Software Engineering Notes, Vol. 28, Issue 4, pp: 1-5.

48. Gill, N. S., 2006. Importance of Software Component Characterization for Better Software Reusability, ACM SIGSOFT Software Engineering Notes, Vol. 31, Issue 1, pp: 1-3.

49. Goel, A. L., Okumoto, K., 1979. A Time Dependent Error Detection Model for Software Reliability and Other Performance Measures, IEEE Transaction on Reliability, August, Vol. 28, pp: 206-211.

50. Goodman, P., 1993, Practical Implementation of Software Metrics, McGraw Hill, London.

51. Goulao, M., Abreu, F. B., 2004. Independent Validation of a Component Metrics Suite. 8th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, Oslo, Norway, pp: 17-22.

52. Graves, T. L., Karr, A. F., Marron, J. S., Siy, H., 2000. Predicting Fault Incidence Using Software Change History. IEEE Transactions on Software Engineering, Vol. 26, July, pp: 653-661.

53. Grover, P. S., Kumar, R., Sharma, A., 2007. Few Useful Considerations for Maintaining Software Components and Component-Based Systems. ACM SIGSOFT Software Engineering Notes, Vol. 32, Issue 4, pp: 1-5.

54. Gui, G., Paul, D. S., 2000. Ranking Reusability of Software Components Using Coupling Metrics. Journal of Systems and Software, Vol. 80, pp: 1450-1459.

55. Gyimothy, T., Ferenc, R., Siket, I., 2005. Empirical Validation of Object Oriented Metrics on Open Source Software for Fault Prediction, IEEE Transaction on Software Engineering, October, Vol. 31, pp: 897-910.

56. Han, J., Kamber, M., 2001. Data Mining: Concepts and Techniques, Harchort India Private Limited.

57. Haykins, S., 1999. A Comprehensive Foundation on Neural Networks, Prentice Hall.

58. Hong, S., Kim, K., 1997. Identifying Fault-Prone Function Blocks Using the Neural Networks - An Empirical Study, Proceeding of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, pp: 790-793.

59. Hu, Q. P., Xie, M., Ng, S. H., Levitin, G., 2007. Robust Recurrent Neural Network Modeling for Software Fault Detection and Correction Prediction, Reliability Engineering and System Safety, Vol. 92, No. 3, pp: 332-340.

60. Huang C. Y., Lin, C. T., 2006. Software Reliability Analysis by Considering Fault Dependency and Debugging Time Lag, IEEE Transaction on Reliability, Vol. 55, No. 3, pp: 436-450.

61. Huang, C. Y., 2005. Performance Analysis of Software Reliability Growth Models with Testing-effort and Change-point, Journal of System Software, Vol. 76, No. 2, pp: 181-194.

62. Huang, C. Y., Kuo, S. Y., Chen, I. Y., 1997. Analysis of a Software Reliability Growth Model with Logistic Testing-effort Function, Proceeding of 8th International Symposium on Software Reliability Engineering, pp: 378-388.

63. Huang, X., Ho, D. H., Ren, J., Capretz, L. F., 2005. A Soft Computing Framework for Software Effort Estimation, Soft Computing, Springer-Verlag, pp: 170-177.

64. Hudson, D. L., Cohen, M. E., 2003. Neural Networks and Artificial Intelligence for Biomedical Engineering, Prentice Hall of India.

65. IEEE Standard 982.2, 1988. Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software, Available: http://standards.ieee.org.

66. IEEE Standard 610.12-1990, 1993. Standard Glossary of Software Engineering Terminology, IEEE Computer Society Press, Los Alamitos, CA.

67. International Standard, ISO/IEC 9126, (2001). Institute of Electrical and Electronics Engineering, Part 1, 2, 3: Quality Model.

68. Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G., 1992. Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, 1992.

69. Jalote, P., Murphy, B., Sharma, V. S., 2008. Post-release Reliability Growth in Software Products, ACM Transactions on Software Engineering and Methodology, Volume 17, Issue 4, pp: 1-20.

70. Jatin, A., Gaur, D., 2012. Estimation of Component Reusability by Identifying Quality Attributes of Component: A Fuzzy Approach, Second International Conference on Computational Science, pp: 738-742.

71. Jeske, D. R., Zhang, X., 2005. Some Successful Approaches to Software Reliability Modeling in Industry, Journal of System Software, Vol. 74, No. 1, pp: 85-99.

72. Jianhong, Z., Sandhu, P. S., Rani, S., 2010. A Neural Network Based Approach for Modeling of Severity of Defects in Function Based Software Systems, International Conference on Electronics and Information Engineering, pp: 568-575.

73. Jingyue, L., Stålhane, T., Conradi, R., Kristiansen, J. M. W., 2012. Enhancing Defect Tracking Systems to Facilitate Software Quality Improvement, IEEE Software, Vol. 29, Issue 2, pp: 59-66.

74. Jorgensen, M., Shepperd, M., 2007. A Systematic Review of Software Development Cost Estimation Studies, IEEE Transaction on Software Engineering, Vol. 33, issue 1, pp.33-53.

75. Judith, A. C., Audrey, E. T., 1998. A Management Guide to Software Maintenance in COTS Based Systems, A Technical Report of Mitre, and Center for Air Force C2 Systems: Bedford, MA, pp: 1-35.

76. Kajko-Mattsson, M., Canfora, G., Chorean, D., Deursen, V. A., Ihme, T., Lehmna, M., Reiger, R., Engel, T., Wernke, J. A., 2006. Model of Maintainability-Suggestion for Future Research, Proceedings of International Multi-Conference in Computer Science & Computer Engineering, Las Vegas, NV, pp: 436-441.

77. Kamiya, T., Kusumoto, S., Inoue, K., Mohri, Y., 1999. Empirical Evaluation of Reuse Sensitiveness of Complexity Metrics, Information and Software

Technology, Vol. 41, pp: 297-305.

78. Kan, H. S., 2003. Metrics and Models in Software Quality Engineering, 2nd Edition, MA: Addison-Wesley.

79. Kanmani, S., Uthariaraj, V. R., Sankaranarayanan, V., Thambidurai, P., 2004. Object Oriented Software Quality Prediction Using General Regression Neural Networks, ACM SIGSOFT Software Engineering, Vol. 29, Issue 5, pp: 1-6.

80. Karunanithi, N., Whitley, D., Malaiya Y. K., 1992. Prediction of Software Reliability Using Connectionist Models. IEEE Transactions on Software Engineering, Vol. 18, No. 7, pp: 32-39.

81. Karunanithi, S., Bieman, J. M., 1993. Candidate Reuse Metrics for Object Oriented and Ada Software, Proceedings of IEEE International Software Metrics Symposium, pp: 78-85.

82. Kaur, A., Kaur, K., Malhotra, R., 2010. Soft Computing Approaches for Prediction of Software Maintenance Effort, International Journal of Computer Applications, Vol. 1, Issue 16, pp: 80-86.

83. Kehan, G., Khoshgoftaar, T. M., Wang, H., Seliya, N., 2011. Choosing Software Metrics for Defect Prediction: An Investigation on Feature Selection Techniques. Software: Practice and Experience, Vol. 41, pp: 579-606.

84. Khairuddin, H., Elizabeth, K., 1996. A Software Maintainability Attributes Model, Malaysian Journal of Computer Science, Vol. 9, Issue 2, pp: 92-97.

85. Khatatneh, K., Mustafa, T., 2009. Software Reliability Modeling Using Soft Computing Technique, European Journal of Scientific Research, Vol. 26, No. 1, pp.154-160.

86. Khoshgoftaar, T. M., Munson, J. C., 1990. Predicting Software Development Errors Using Complexity Metrics, IEEE Journal of Selected Area in Communication, Vol. 8, No. 2, pp: 253-261.

87. Khoshgoftaar, T. M., Allen, E. B., Xu, Z., 2000. Predicting Testability of Program

Modules Using a Neural Network, IEEE Conference on Soft Computing, pp: 57-62.

88. Khoshgoftaar, T. M., Allen, E. B., Hudephol, J. P., Aud, S. J., 1997. Application of Neural Networks to Quality Modeling of A Very Large Telecommunication System, IEEE Transactions on Neural Networks, Vol. 8, pp: 902-909.

89. Kothari, C. R., 1998. Research Methodology, Methods and Techniques, New Age International Limited.

90. Kumar, R., Rai, S., Trahan, J. L., 1998. Neural Network Techniques for Software Quality Evaluation, Proceedings of Annual Reliability and Maintainability Symposium, pp: 155-161.

91. Kumar, V., Sharma, A., Kumar, R., 2013. Applying Soft Computing Approaches to Predict Defect Density in Software Product Releases: An Empirical Study. Computing and Informatics, North America, Vol. 32, No. 1, March, pp: 203-224.

92. Kumar, V., Sharma, A., Kumar, R., Grover, P. S., 2012. Quality Aspects for Component-based Systems: A Metrics Based Approach, Software: Practice and Experience, John Wiley & Sons, December, Vol. 42, Issue 12, pp: 1531-1548.

93. Liu, X., Kane, G., Bambroo, M., 2003. An Intelligent Early Warning System for Software Quality Improvement and Project Management, Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence, pp: 32-28.

94. Lo, J., 2009. The Implementation of Artificial Neural Networks Applying to Software Reliability Modeling, Proceedings of the 21st Annual International Conference on Chinese Control and Decision Conference, pp: 4349-4354.

95. MacDonell, S. G., Gray, A. R., Calvert, J. M., 1999. FLSOME: Fuzzy Logic for Software Metric Practitioners and Researchers, Proceedings of the 6th International Conference on Neural Information Processing, Perth, pp: 308-313.

96. Madsen, H., Thyregod, P., Burtschy, B., Popentiu, F., Albeanu, G., 2005. On Using Soft Computing Techniques in Software Reliability Engineering, Advances

in Safety and Reliability, Taylor & Francis Group, London, pp: 1317-1323.

97. Malhotra, R., Chug, A., 2102. Software Maintainability Prediction Using Machine Learning Algorithms, Software Engineering: An International Journal, Vol. 2, No. 2, pp: 19-36.

98. Mauricio, J., Hisham, M., 2008. The State of Metrics in Software Industry, Proceeding of Fifth International Conference on Information Technology: New Generations, pp: 453-458.

99. Mayrhauser, A., Anderson, C., Mraz, R., 1995. Using a Neural Network to Predict Test Case Effectiveness, Proceedings of IEEE Aerospace Applications Conference, Snowmass, CO, pp: 77-91.

100. McCall, J., Richards, P., Walters, G., 1977. Factors in Software Quality, Vol. 3, US Rome Air Development Center Reports NTIS, pp: 959-969.

101. McConnell, S., 1996. Software Quality at Top Speed, Software Development, Vol. 4, No. 8, pp: 38-42.

102. McCulloch, W., Pitts, W. 1943. A Logical Calculus of the Ideas Immanent in Nervous Activity, Bulletin of Mathematical Biophysics, Vol. 7, pp: 115-133.

103. Meine, J. P., Miguel, A. R., 2007. Correlations Between Internal Software Metrics and Software Dependability in a Large Population of Small C/C++ Programs, Proceeding of 18th IEEE International Symposium on Software Reliability Engineering, pp:203-208.

104. Mili, H., Mili, F., Mili, A., 1995. Reusing Software: Issues and Research Directions, IEEE Transaction on Software Engineering, Vol. 21, Issue 6, pp: 528-561.

105. Misra, S., Kilic, H., 2006, Measurement Theory and Validation Criteria for Software Complexity Measures, ACM SIGSOFT Software Engineering Notes, pp: 1-3.

106. Mukherjee, A., Deshpande, J. M., 1995. Neural Network Based Expert Systems

for Structural Design, Computers and Structures, Vol. 54, Issue 3, pp: 367-375.

107. Munro, M. J., 2005. Product Metrics for Automatic Identification of Bad Smell Design Problems in Java Source-Code, Proceeding of 11th IEEE International Software Metrics Symposium (METRICS 2005), pp: 15-24.

108. Musa, J., Anthony, L., Kazuhira, O., 1987. Software Reliability, McGraw-Hill.

109. Musilek, P., Pedrycz, W., Succi, G., Reformat, M., 2000. Software Cost Estimation with Fuzzy Models, ACM SIGAPP Applied Computing Review, Vol. 8, pp: 24-29.

110. Myers, G. J., 1979. The Art of Software Testing, John Wiley & Sons, New York.

111. Nachiappan, N., Thomas, B., 2005. Use of Relative Code Churn Measures to Predict System Defect Density, Proceeding of ACM International Conference on Software Engineering, St. Louis, MO, USA, pp: 284-292.

112. Ohlsson, N., Alberg, H., 1996. Predicting Error-Prone Software Modules in Telephone Switches, IEEE Transaction on Software Engineering, Vol. 22, No. 12, pp. 886-894.

113. Pandey, A. K., Goyal, N. K., 2013. Background: Software Quality and Reliability Prediction, Early Software Reliability Prediction, Studies in Fuzziness and Soft Computing, Springer, pp: 17-33.

114. Park, R. E., Goethert, W. B., Florac W. A., 1996. Goal Driven Software Measurement - A Guidebook, Software Engineering Institute, Carnegie Mellon University.

115. Paul, R. A., Bastani, F., Yen, I., Challagulla, V. U. B., 2000. Defect Based Reliability Analysis for Mission-Critical Software, Computer Software and Applications Conference, pp: 439-444.

116. Pedrycz, W., Reformat, M., Pizzi, N., 2004. Neuro-fuzzy Analysis of Software Quality Data, Studies in Fuzziness and Soft Computing, Vol. 159, pp: 254-273.

117. Pfleeger, L. S., 1992. Measuring software reliability, IEEE Spectrum, Vol. 29,

Issue 8, pp: 56-60.

118. Pizzi, N. J., Pedrycz, W., 2006. Predicting Qualitative Assessments Using Fuzzy Aggregation, Proceeding of Fuzzy Information Processing Society, pp: 267-272.

119. Pomorova, O., Hovorushchenko, T., 2011.Research of Artificial Neural Network's Component of Software Quality Evaluation and Prediction Method, Proceeding of 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, pp:959-962.

120. Pressman, R. S., 2005. Software Engineering: A Practitioner's Approach, 6th Edition, McGraw Hill.

121. Quah, T., Thwin, M. M. T., 2005 Application of Neural Networks for Software Quality Prediction Using Object-Oriented Metrics, ACM Journal of Systems and Software, Vol. 76, Issue 2, pp: 147-156.

122. Quyoum, A., Dar, M., Quadri, S. M. K., 2010. Improving Software Reliability using Software Engineering Approach- A Review, International Journal of Computer Applications, Volume 10, No.5, pp: 41-47.

123. Raimund, M., Witold, P., Giancarlo, S., 2008. A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction, Proceeding of International Conference on Software Engineering, Leipzig, Germany, pp: 181-190.

124. Reformat, M., Pedrycz, W., Pizzi, N. J., 2002. Software Quality Analysis with the Use of Computational Intelligence International Conference on Fuzzy Systems, 2002, Vol. 2, pp: 1156-1161.

125. Reussner, R. H., Schmidt H. W., Poernomo, I. H., 2003. Reliability Prediction for Component-based Software Architectures, The Journal of Systems and Software, Vol. 66, pp: 241-252.

126. Rosenberg, J., Microsyst, S., 1997. Some Misconceptions about Lines of Code, Proceedings of Fourth International Software Metrics Symposium, pp: 137-142.

127. Rotaru, O. P., Dobre, M., Petrescu, M., 2005. Reusability Metrics for Software Components. Proceedings of the 3rd ACS / IEEE International Conference on Computer Systems and Applications, Cairo, Egypt, pp: 24-29.

128. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., 1991. Object-Oriented Modeling and Design, Prentice-Hall, New York.

129. Ryder, J., 1998. Fuzzy Modeling of Software Effort Prediction, Proceedings of IEEE Information Technology Conference, Syracuse, New York, pp: 53-56.

130. Saliu, M. O., Ahmed, M., AlGhamdi, J., 2004. Towards Adaptive Soft Computing Based Software Effort Prediction, IEEE Annual Meeting of the Fuzzy Information Processing, pp: 16-21.

131. Sandhu, P. S., Salaria, D. S., Singh, H., 2008, A Comparative Analysis of Fuzzy, Neuro-Fuzzy and Fuzzy-GA Based Approaches for Software Reusability Evaluation, World Academy of Science, Engineering and Technology, Vol. 15, pp: 342-345.

132. Sedigh-Ali, S., Ghafoor, A., Paul, R. A., 2001. Metrics Guided Quality Management for Component-based Software Systems, Proceedings of the 25th International Computer Software and Applications Conference on Invigorating Software Development, pp: 303-308.

133. Sharma, A., Kumar, R., Grover, P. S., 2008. Empirical Evaluation of Complexity for Software Components, International Journal of Software Engineering and Knowledge Engineering, Vol. 18, Issue 5, pp: 519-530.

134. Sharma, A., Kumar, R., Grover, P. S., 2009. Predicting Maintainability of Component-based Systems by Using Fuzzy Logic, Communications in Computer and Information Science, Springer Berlin Heidelberg, USA, Vol. 40, Issue 11, pp: 581-591.

135. Sharma, A., Kumar, R., Grover, P. S., 2009. Reusability Assessment for Software Components - A Neural Network Based Approach, International IEEE

Conference, Thapar University, Patiala, March, pp: 56-63.

136. Shing, J., Jang, R., 1993. ANFIS: Adaptive Network Based Fuzzy Inference System, IEEE Transactions on Systems, Man and Cybernetics, Vol. 23, pp: 665-685.

137. Shukla, R., Mishra, A. K., 2008. Estimating Software Maintenance Effort - A Neural Network Approach, Proceedings of 1st conference on Software Engineering, Hyderabad, India, pp: 107-112.

138. Sindre, G., Conradi, R., Karlsson, E. A., 1995. The REBOOT Approach to Software Reuse, Journal of Systems and Software, Vol. 30, Issue 3, pp: 201-212.

139. Singh, Y., Bhatia, P. K., Sangwan, O. P., 2011. Software Reusability Assessment Using Soft Computing Techniques, ACM SIGSOFT Software Engineering Notes, Vol. 36, Issue 1, pp: 1-7.

140. Singh, Y., Kaur, A., Sangwan, O. P., 2004. Neural Model for Software Maintainability, Proceedings of International Conference on ICT in Education and Development, pp: 1-11.

141. Sivanadam S. N., Sumathi, S., Deepa, S. N., 2006. Introduction to Neural Networks Using MATLAB 6.0, Tata McGraw Hill, NewDelhi.

142. Sivanandam, S. N., Sumathi, S., Deepa, S. N., 2007. Introduction to Fuzzy Logic Using MATLAB, Springer.

143. So, S. S., Cha, S. D., Kwon, Y. R., 2002. Empirical Evaluation of A Fuzzy Logic Based Software Quality Prediction Model, Fuzzy Sets and Systems, Vol. 127, pp: 199–208.

144. Stieber, H. A., 2007. A Family of Software Reliability Growth Models, Proceeding of 31st Annual International Computer Software and Applications Conference, Vol. 2, pp: 217-224.

145. Su Y. S., Huang, C., Chen, Y. S., Chen, J. S., 2005. An Artificial Neural-Network-Based Approach to Software Reliability Assessment, Proceeding of IEEE Region

Conference, pp:1-6.

146. Menzies, T., Caglayan, B., Kocaguneli, E., Krall, J., Peters, F., Turhan, B., 2012. The PROMISE Repository of Empirical Software Engineering Data http://promisedata.googlecode.com, West Virginia University, Department of Computer Science.

147. Tagi, A. I., Khoshgoftaar, M., Abran A., 2002. Can Neural Networks Be Easily Interpreted in Software Cost Estimation, IEEE Transactions on Software Engineering, pp: 1162-1167.

148. Tahir, A., MacDonell, S. G., 2012. A systematic Mapping Study on Dynamic Metrics and Software Quality, Proceeding of 28th IEEE International Conference on Software Maintenance, pp: 326-335.

149. Vivanco, R., Pizzi, N., 2004. Finding Effective Software Metrics to Classify Maintainability Using a Parallel Genetic Algorithm, Genetic and Evolutionary, Computing, Springer, Vol. 31, pp:1388-1399.

150. Voas, J., 1998. Maintaining Component-Based Systems, IEEE Software, Vol. 15, Issue 4, pp: 22-27.

151. Voas, J., 2000. Can Chaotic Methods Improve Software Quality Predictions?, IEEE Software, pp: 20-22.

152. Wadhwa, B., Stella., L. F. F., Jarzabek, S., 2008. A Comparative Study of Maintainability of Web Applications on J2EE, .NET and Ruby on Rails, Proceeding of 10th IEEE International Symposium on Web Site Evolution, Los Alamos, October, pp: 93-99.

153. Wang, A. J. A., 2002. Reuse Metrics and Assessment in Component-based Development, Proceedings of 6th IASTED International Conference on Software Engineering and Applications, Cambridge, Massachusetts, USA, pp: 583-588.

154. Wang, L. X., Mendel, J. M., 1992. Fuzzy Basis Function, Universal Approximation, and Orthogonal Least Squares Learning, IEEE Transaction on

Neural Networks, Vol. 3, No. 5, pp: 807-814.

155. Wang, S., Wu, Y., Lu, M., Li, H., 2012. Discrete Nonhomogeneous Poisson Process Software Reliability Growth Models Based on Test Coverage, Quality and Reliability Engineering, Wiley Online, Vol. 29, Issue 1, pp:103-112.

156. Washizaki, H., Hirokazu, Y., Yoshiaki, F., 2003. A Metrics Suite for Measuring Reusability of Software Components. IEEE Proceedings of the 9th International Symposium on Software Metric, Sydney, NSW, Australia, pp: 211-223.

157. Xu, Z., Zheng, X., Guo, P., 2006. Empirically Validating Software Metrics for Risk Prediction Based on Intelligent Methods, IEEE Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications, pp: 1049-1054.

158. Yang, C., Hou, C., Kao, W., Chen, I., 2012.An Empirical Study on Improving Severity Prediction of Defect Reports Using Feature Selection, Proceeding of 19th Asia-Pacific Software Engineering Conference, pp: 240-249.

159. Yasunari, T., Toshifumi, T., Naoki, N., Keishi, S., Shinji, K., Tohru, K., 1995. Analysis of Review's Effectiveness Based on Software Metrics, Proceeding of Sixth International Symposium on Software Reliability Engineering, pp: 34-39.

160. Yingjie, T., Chuanliang, C., Chunhua, Z., 2008. AODE for Source Code Metrics for Improved Software Maintainability, Proceedings of the Fourth International Conference on Semantics, Knowledge and Grid, pp: 330-335.

161. Yu, T., Ding, X., 2012. The Research of Software Reliability Hybrid Model Based on Weighted Metrics, International Conference on Wavelet Active Media Technology and Information Processing, pp: 377-380.

162. Yuan, X., Khoshgoftaar, T. M., Allen, E. B., Ganesan, K., 2000. An Application of Fuzzy Clustering to Software Quality Prediction, Proceedings of IEEE Symposium on Application Specific Systems and Software Engineering Technology, pp: 85-90.

163. Yuan, D., Zhang, C., 2011. Evaluation Strategy for Software Reliability Based on ANFIS, International Conference on Electronics, Communications and Control, pp: 3738-3741.

164. Zadeh, L. A., 1965. Fuzzy Sets, Journal of Information and Control, Vol. 8, pp: 338–353.

165. Zadeh, L. A., 2002. From Computing with Numbers to Computing with Words-from Manipulation of Measurements to Manipulation of Perceptions, International Journal of Applied Mathematics and Computer Science, Vol.12, Issue 3, pp: 307-324.

166. Zadeh, L. A., 1994. Fuzzy Logic, Neural Networks, and Soft Computing, Communication of the ACM, Vol. 37, No. 3, pp: 77-84.

167. Zeng, F., Chen, A., Tao, X., 2009. Study on Software Reliability Design Criteria Based on Defect Patterns, Proceeding of 8th International Conference on Reliability, Maintainability and Safety, pp: 723-727.

168. Zhang, X., Pham, H., 2006. Software Field Failure Rate Prediction Before Software Deployment, Journal of System Software, Vol. 79, No. 3, pp. 291-300.

169. Zhang, Y., Xiao, J., 2012. A Software Reliability Modeling Method Based on Gene Expression Programming, Applied Mathematics & Information Sciences, Vol. 6, No. 1, pp: 125-132.