**TABLE OF CONTENT**

[CHAPTER ONE 4](#_Toc34654315)

[INTRODUCTION 4](#_Toc34654316)

[1.0 Background of Study 4](#_Toc34654317)

[1.1 Statement of the Problem 5](#_Toc34654318)

[1.2 Aim and Objectives 6](#_Toc34654319)

[1.3 Research Methodology 7](#_Toc34654320)

[1.4 Significance of the Study 8](#_Toc34654321)

[CHAPTER TWO 9](#_Toc34654322)

[LITERATURE REVIEW 9](#_Toc34654323)

[2.0 Introduction 9](#_Toc34654324)

[2.1 Expert Judgment 10](#_Toc34654325)

[2.2 Analogy-Based Estimations 12](#_Toc34654326)

[2.2.1 Introduction to Analogy Based Estimation 13](#_Toc34654327)

[2.2.2 Selection of Similar Past Projects 14](#_Toc34654328)

[2.2.3 Short-Listing of Past Projects 16](#_Toc34654329)

[2.3 Constructive Cost Model (COCOMO) 19](#_Toc34654330)

[2.3.1 Overview of COCOMO Model 19](#_Toc34654331)

[2.3.2 Source Lines of Code 20](#_Toc34654332)

[2.3.3 The Scale Drivers 20](#_Toc34654333)

[2.3.4 Cost Drivers 20](#_Toc34654334)

[2.3.5 COCOMO II Effort Equation 21](#_Toc34654335)

[2.3.6 Effort Adjustment Factor 21](#_Toc34654336)

[2.3.7 COCOMO II Schedule Equation 22](#_Toc34654337)

[2.3.8 The SCED Cost Driver 22](#_Toc34654338)

[2.4 Putnam Model 24](#_Toc34654339)

[2.5 Function Point Analysis 25](#_Toc34654340)

[2.6 Soft Computing Based Effort Evaluation Techniques 26](#_Toc34654341)

[2.6.1 Overview of fuzzy logic 26](#_Toc34654342)

[2.6.1.1 Fuzzy Set 27](#_Toc34654343)

[2.6.2 Neural Networks 35](#_Toc34654344)

[2.6.3 Particle Swarm Optimization (PSO) Algorithm 39](#_Toc34654345)

[2.7 ANFIS (Adaptive Neuro-Fuzzy Inference System) 43](#_Toc34654346)

[CHAPTER THREE 53](#_Toc34654347)

[RESEARCH METHODOLOGY AND DESIGN 53](#_Toc34654348)

[3.1 Description of the Existing System 53](#_Toc34654349)

[3.2 The Neuro-Fuzzy based Evaluation of Software System. 54](#_Toc34654350)

[3.2.1 Sub-models 55](#_Toc34654351)

[3.2.2 Advantages of the Existing System 55](#_Toc34654352)

[3.2.3 Problem of the Existing System 55](#_Toc34654353)

[3.2.4 System Design 56](#_Toc34654354)

[3.3.1 Description of the proposed system 57](#_Toc34654355)

[3.3.2 Dataset 58](#_Toc34654356)

[3.3.2.1 Data gathering 59](#_Toc34654357)

[3.3.2.3 Description of Attributes 60](#_Toc34654358)

[3.3.3 Data Preprocessing 65](#_Toc34654359)

[3.3.4 Principal Component Analysis (PCA) 65](#_Toc34654360)

[3.3.4.1 PCA Algorithm 65](#_Toc34654361)

[3.3.5 Rosetta Stone 71](#_Toc34654362)

[3.4 Adaptive Neuro-Fuzzy Inference System 75](#_Toc34654363)

[3.5 Fuzzy Inference System (FIS) 82](#_Toc34654364)

[3.5.1 Description of Fuzzy Logic Model 82](#_Toc34654365)

[3.5.2 Inference Process of Fuzzy Expert Model 83](#_Toc34654366)

[3.5.3 Fuzzification 83](#_Toc34654367)

[3.5.4 Inferencing 83](#_Toc34654368)

[CHAPTER FOUR 84](#_Toc34654369)

[4.1 Results 84](#_Toc34654370)

[4.1.1 Feature Extraction (PCA) Result 84](#_Toc34654371)

[4.1.2 Adaptive Neuro-Fuzzy (ANFIS) Result 92](#_Toc34654372)

[4.1.2.1 ANFIS Training 96](#_Toc34654373)

[4.1.3 Performance Evaluation Result 109](#_Toc34654374)

[4.2 Discussion 110](#_Toc34654375)

[4.3 Implication of Research on Software Developers and Researchers 113](#_Toc34654376)

# CHAPTER ONE

## INTRODUCTION

### **1.0** **Background of Study**

Software development effort estimation or simply put software effort estimation is the process of predicting the most realistic amount of effort (expressed in terms of person-hours or money) required to develop software based on the incomplete, uncertain and noisy input. Incomplete, uncertain and noisy in the above definition means that at the early stage of software development, only less information is available (Kaushik et al. 2013). Rijwani (2016) defined software effort estimation as a process by which one can predict the development time and cost of developing a software process or product.

The term software evaluation means assessing various aspects of software systems in order to determine its effectiveness and also to select the best alternative. The goal of any successful software project is to develop quality software within time, cost, and resource constraints. Idri et al (2010) stated that a project underestimated, once it seems running out of the schedule, the project manager and his team is put under high pressure to finish the software.

Also, according to Parkinson’s Law, overestimation can cause a work expand to fill the time available. For example, if the software has to be delivered in 12 months and 5 people are available, the effort required is estimated to be 60 person-months. The primary purpose of evaluation, in addition to gaining insight into prior or existing initiatives, is to enable reflection and assist in the identification of future change. A qualitative evaluation can improve software development through effective management of the software development process.

Well defined measures of the process and the product are necessary to exercise control and to bring about improvement in the software development process. Software metrics are quantitative measures that provide the basis for effective management of the software development process. Software metrics deals with the evaluation and measurement of different attributes of the software product and the software development process.

There are three kinds of software metrics: procedure metrics, project metrics, and product metrics.

Software estimation models are majorly classified into analogy and expert judgment and formal estimation models. The major difference between the two is the use of input parameters in the later while the former requires only finding analogies based on historical project. These available techniques have been used in measuring or estimation software development effort: Use Case Point, Fuzzy Logic, Neural Network, Genetic Algorithm, Function Point, ANFIS, and a combination of one or more

### 1.1 Statement of the Problem

Software effort evaluation is one of the most essential and crucial parts of software project planning for which efficient effort metrics is required.

However, estimates that are computed early in the lifecycle are typically associated with uncertainty. To get a near to accurate effort has always been a challenge in software development. To deal with this problem, many researchers have contributed in various areas by applying many techniques. These techniques include regression analysis, analogy-based evaluation, comparison-based evaluation and machine learning based evaluation.

The development of software has always been characterized by parameters that possess a certain level of fuzziness. The attributes in the COCOMO based NASA dataset are analyst capability, programmers capability, application experience, modern programming practice, use of software tools, virtual machine experience, language experience, schedule constraint, main memory constraint, database size, time constraint for CPU, turnaround time, machine volatility, process complexity, required software liability and the physical kilo line of source code and the actual effort which possess certain level of imprecision.

Programmers’ capability (pcap): Because planning is at the early stage of the life cycle, one cannot determine the programmer’s capability for the problem domain at that early stage, if we do there are some uncertainties and imprecision associated with such estimation.

Modern Programming Practice (mcap): This multiplier too at the early stage has some imprecision, it is not certain at such time to determine which programming practice to employ whether top-down requirement analysis and design or a structured coding or a top-down incremental development, though there are degree (very low, low, nominal, high, very high and extra high) to which modern programming practices are used in developing software.

### 1.2 Aim and Objectives

The aim of this work is to develop an efficient software effort evaluation model using ANFIS. The objectives are to;

* Subject dataset gathered from National Aeronautics and Space Administration (NASA)
* To Convert dataset from COCOMO I to COCOMO II by following the COCOMO II Model definition manual and Rosetta stone.
* Use ANFIS to train the Fuzzy Inference System by using least square and back propagation methods and also their hybrid (of forward pass where consequent parameters are identified by least square method and backward pass where premise parameters are updated by gradient descent) learning algorithms
* Reduce the number input parameters of the dataset using Principal Component Analysis (PCA)
* Evaluate the methods using recognized performance measures such as Magnitude of Relative Error (MRE), and Mean Magnitude of Relative Error (MMRE).

### 1.3 Research Methodology

The research methodology for this work will be conducted using the following approaches:

1. Review Literature on Software Evaluation (Both traditional and soft computing methods)
2. Data acquisition from promise repository – NASA COCOMO Dataset.
3. Proposed framework includes the evaluation of software effort Optimization of Sugeno Fuzzy Inference System parameters with Least Square, Back Propagation and Hybrid algorithm of Adaptive Neuro Fuzzy Inference System (ANFIS) using MATLAB R2015a.
4. The performance measures MMRE and RMSE are used in evaluating the performance of the outputs of various ANFIS algorithms.

### 1.4 Significance of the Study

The output of the study will help project managers and software development groups at large to evaluate the effort required to develop a software product. The study will also serve as a full reference material to other researchers seeking for information concerning the subject.

# CHAPTER TWO

## LITERATURE REVIEW

### 2.0 Introduction

Software development efforts estimation is the process of predicting the most realistic use of effort required to develop or maintain software based on incomplete, uncertain and/or noisy input. Effort estimates may be used as input to project plans, iteration plans, budgets, and investment analyses, pricing processes and bidding rounds. Software effort estimation is a necessary feature that guides and supports the planning of software projects. Software effort estimation refers to the predictions of the likely amount of effort, time, and staffing levels required to build a software system. An extremely helpful form of effort prediction is the one made at an early stage of a project, when the costing of the project is proposed for approval. Agrawal et al (2015)

According to Popli et al (2014), an effort estimate is a prediction of how long a development activity will take to finish. Effort estimates can change depending on what stage they are done, meaning that there can be different effort estimates within the same project. This means that an early effort estimate is based on incomplete and uncertain information, then a planning estimate is based on the actual requirement specification and later on during development the effort is re-estimated.

To deal with this problem, many effort evaluation techniques (algorithmic and non-algorithmic) and metrics are developed by many researchers based on many different methods. Traditionally, there are various evaluation techniques based on comparison, analogy, equations which are broadly categorized as macro evaluation techniques. Software estimation models are majorly classified into analogy and expert judgment and formal estimation models. The major difference between the two is the use of input parameters in the later while the former requires only finding analogies based on historical project.

### 2.1 Expert Judgment

The term “expert” is used to denote all individuals with competence in estimating software development effort. The expert is a software development professional, but the term “expert” can also be used to denote, for example, a student with previous experience in effort estimation and the development of software for the type of task under consideration. In this type of estimation an expert studies the specification and makes his estimation which is mainly based on the expert knowledge. One or more experts in both software development and the application domain use their experience to predict software costs. The process iterates until some consensus is reached. An early technique was named after the Ancient Greek oracle, who could predict the future. It involves the collection and aggregation of expert opinion and was initially used by the military to estimate the probable effects of massive atomic bombing. The Delphi Technique is a method used by a group of experts to estimate the efforts in a software project. It is based on expert judgment.

Anda et al (2005) noticed that some effort estimates were probably subject to typical field setting-biases, example was a bias towards over optimism. He used a mix of field and laboratory studies. Expert judgment field setting was compared with a laboratory setting model. Ten projects were estimated.

Four of the projects were based on same requirement specification. The Case Point-based model was used with a low calibration level. The input to the model contained the information necessary for using the Use Case Point estimation model.

Results: In terms of accuracy, the model-based estimates were more accurate than those of the expert judgment-based. The Mean Absolute Percentage Error was used as a performance measure and it shows that the model-based estimate performed better than the expert judgment estimates. In variance, the expert judgment estimates had a stronger tendency towards strong over-optimism. The result of the study deviated from those of other studies in that the model was not calibrated to the organizations using them, but still produced the most accurate estimates. One reason for this may be that the model was based on projects quite similar to those in the study.

Grimstad et al (2007) defined judgment-based effort estimation to be based on a tacit (intuition-based) quantification step, and model-based effort estimates to be based on a deliberate (mechanical) quantification step. The quantification step is the final step of the process leading to an effort estimate for the total project or a project activity. If the final step is judgmental, the process is categorized as judgment-based. If the final step is mechanical, the process is categorized as model-based. There will be a range of quite different estimation processes belonging to each of the categories, i.e., neither expert judgment nor model-based effort estimation should be considered simply as “one method”. When the outputs of two or more completed estimation processes are combined, we categorize the process as combination-based, and describe whether the combination step is judgmental or mechanical.

#### **2.1.1 Expert Judgment-based Effort Estimation Processes**

Most of the steps in the expert judgment-based effort estimation processes, e.g., the breaking down of the project into activities, may be explicit and can be reviewed readily. The quantification steps, however, are based on three intuitions to a significant degree, and are seldom based on explicit, analytical argumentation. This assessment of the quantification steps as being based on intuition is indicated both by a lack of analytical argumentation and by the frequent use of phrases such as “I think that …” and “I feel that …” The poor understanding of the quantification step is also an indication that it is intuition-based.

Grimstad et al (2007) in their article used Simple rule-based model with a medium calibration level to compare expert judgement using eighteen projects estimated. They proposed that there were possible motivational biases in estimation situation probably subject to typical field setting-biases. The expert judgment-based effort estimates were probably based on textual and oral information about the project requirements. The input into the models contained a judgement-based classification of the elements of the software to be developed.

Results: When not adjusting the differences in estimation complexity, the models yielded more accurate estimates than did the experts. When comparing only projects with similar estimation complexities, the estimation accuracy is similar, e.g. when comparing projects where the estimator had some previous experience. There was similar variance in estimation accuracy. Other results illustrated the importance of adjusting for potential biases in methods of selecting estimation methods.

### 2.2 Analogy-Based Estimations

Analogy Based Effort Estimation (ABE) is a simplified process of finding nearest analogies of the current system being developed based on notion of retrieval by similarity of completed software projects or components, Azzeh (2012). Conventional approaches to software cost estimation have focused on algorithmic cost models, where an estimate is calculated from one or more numerical inputs via a mathematical model. Where there is limited or incomplete data and limited expertise in numerical techniques, these models can be daunting to calibrate and use.

Estimating software project effort by analogy is an example of a case-based reasoning strategy. Case-based reasoning is a form of analogical reasoning where the potential analogues and target are examples of the same thing, for example software projects. An estimate of the effort to complete a new software project is made by analogy with one or more previously completed projects.

Estimating software project effort by analogy usually involves a number of steps:

1. Measuring or estimating the values of project metrics for the target project;

2. Searching a repository of completed projects for projects similar to the target and selecting one or more projects as source analogues;

3. Using the effort value of the source analogue(s) as an initial estimate for the target project;

4. Comparing the known metric values for the target and source projects; and

5. Adjusting the effort estimate in light of the differences between the target and source projects.

### 2.2.1 Introduction to Analogy Based Estimation

Analogy Based Software Estimation is based on the principle that actual values achieved within the organization in an earlier and similar project are better indicators and predict the future project performance much better than an estimate developed a fresh from scratch. It also facilitates bringing the organizational experience to bear on the new projects.

However, to use this technique, it is necessary for the organization to put in place certain prerequisites, such as:

1. The organization ought to have executed a number of projects

2. The organization should be keeping meticulous records of past projects

3. The organization must be conducting project post mortem for every project and causes for variances must be identified using meticulous methods and the actual values validated depending on the causes. Care must be taken to prevent erroneous data to influence future projects.

4. The organization should have a well-organized and maintained Knowledge Repository from which it is feasible to locate similar past projects and extract the validated project data

5. The estimators should be trained in drawing analogies accurately and in accessing the Knowledge Repository and extracting validated data and extrapolate the same for the current project

Once these pre-requisites are in place, this technique can very profitably be used in the organization.

### 2.2.2 Selection of Similar Past Projects

This is a crucial step in this methodology and care must be taken in short-listing similar projects. First criteria in short-listing is the type of project, namely;

1. Full Life Cycle Software Development Project - normally referred to as Development Project

2. Software Implementation Projects such as ERP, Supply Chain Management, CRM etc.

3. Conversion Project: convert an application to make it usable in the new circumstances - such as Y2K projects, Euro Conversion projects etc.

4. Porting Projects: Porting from one software platform to another such as porting from one version of Unix to another version of Unix

5. Migration Projects: Migrate the application from one hardware platform to another - say from Data General to IBM Mainframe

The most crucial aspect for the success of Analogy Based Estimation is the selection of right set of past projects. The following parameters need to be considered.

1. Application Domain: This is perhaps the single most important feature to be considered. It would not make sense to draw analogy between two different domains. For example - would it make sense to select a Marketing Information project to draw analogy for a Material management Information project? Therefore, draw analogy from similar application domain.

2. Organization size of the prospective client: The extent of functionality would differ between different sizes of organizations even if the domain is the same. The functionality of Material management, for example, for medium-sized organizations would significantly differ from a large sized organization. Select a past project that is comparable in size with the current project.

3. Number of Locations of the prospective Client: The functionality for a single location would be vastly different for a multi-location organization. Therefore, select a past project that is similar in number of locations of the client organization with the current project.

4. Nature of modules in the application: the past project selected needs to include majority of the modules that the current project has. We can adjust and extrapolate for extra modules for one or two modules but not for a majority. The following parameters from the development platform need to be considered;

1. Number of application tiers: A two-tier application would significantly differ from a three tier project.

2. Backend: In present day, almost all applications are built with a Relational Database Management System (RDBMS). As long as the backend is an RDBMS in both the cases it can be considered equivalent. However, if one of them is flat files and the other is RDBMS, then they would be different

3. Web Server: Different web servers cause different amount of work. We may need to extrapolate based on the web servers used. This would be applicable in web-based application development.

4. Middleware: Different middleware have different impacts on the amount of effort required.

5. Rules Engines: if the proposed project uses a Rules Engine, it would be desirable to select a past project that also used a Rules Engine. Also significant is the fact that different rules engines would have different impacts on the amount of effort required for software development.

6. Programming language: The amount of work is influenced to a large extent by the language in which programs are developed. If the past project used a different programming language than the present project, we may need to adjust the estimate for difference in programming language.

7. Development environment: The type of tools used for editing the programs, debugging, compiling etc. have a large impact on the productivity of programmers. Hence it is important to select past projects that have similar software development environments.

8. Software Development Process used: It is also important to select projects that are similar in the manner of developing software conforming to the process that is likely to be used in the current project.

9. Location of Development: Development locations can be either in-house or at client location. It is better to select a past project that used similar location.

### 2.2.3 Short-Listing of Past Projects

The overriding criterion for selection of projects is the application domain. It is futile to shortlist projects that are in different application domain. Therefore, it is essential to make this the first criteria for short-listing of projects.

Auer et al (2006) in their paper discussed analogy based cost estimation in contest with optimal project feature weights. The proposed strategy utilizes broad pursuit to discover ideal project feature weight for a relationship base cost estimation. The methodology says each peculiarity has a unique impact on the quest for a comparable feature within a chronicled featured database. This wipes out the requirements in favor of specialist to assign and adjust the weight physically, taking into account their own particular experience. Straightforward strategy for the dataset of high ends dispense with those peculiarities that are unrealistic to impact the quality of estimation. Discussed approach outperforms existing strategies concerning ordinarily utilized estimation quality metrics.

Analogy Based Estimation method has been widely used for developing software effort estimation models based upon retrieval by similarity, Azzeh et al (2010). Analogy Based Estimation method involves four primary steps;

(1) Select k nearest analogies using Euclidean distance function as depicted in equation 2.1.

(2) Reuse efforts from the set of nearest analogies to find out effort of the new project.

(3) Adjust the retrieved efforts to bring them closer to the new project.

(4) Retain the estimated project in the repository for future prediction.

….. (2.1)

Where is the Euclidean distance between projects x and y across m predictor features.

Kocaguneli et al (2011) discovered that in spite of Analogy Based Estimation generates better accuracy than other well-known prediction methods; it still requires adjusting the retrieved estimates to reflect the structure of nearest analogies on the final estimate. Practically, the key factor of successful Analogy Based Estimation method is finding the appropriate number of k analogies. Li et al (2009) and Azzeh et al (2010) in their various work recommended using a fixed number of analogies starting from k=1 and increase this number until no further improvement on the accuracy can be obtained. Azzeh et al (2010) discovered that this approach is somewhat simple, but not necessarily accurate, and relies heavily on the estimator intuitions.

In this direction, Kirsopp et al (2003) proposed making predictions from the k=2 nearest cases as it was found the best value for their investigated datasets. They have increased their accuracy values with case and feature subset selection strategies. The conclusion drawn from their empirical studies is that the same k number has been used for all datasets irrespective of their size and feature types (i.e. numerical, categorical and ordinal features).

Azzeh et al (2010) carried out an extensive replication study on various linear and non-linear adjustment strategies used in Analogy Based Estimation in addition to finding the best k number for these strategies. He found that k=1 was the most influential setting for all adjustment strategies over all datasets under investigation. They proposed a fuzzy similarity approach that can select the best analogies for which their similarity degrees are greater than the predefined threshold. This approach could ignore some useful projects which might contribute better when similarity between selected and unselected cases is negligible. Also, the determination of the threshold value is a challenge on its own and needs expert intuition.

Li et al (2007) focused on k analogies identification in the context of Analogy Based Estimation. They proposed a new model of Analogy Based Estimation called AQUA which consists of two main phases: learning and prediction. During the learning phase, the model attempts to learn the k analogies and best similarity threshold by performing cross-validation on all training projects. The obtained k is then used during second phase to make prediction for different test projects. In the study they performed rigorous trials on actual and artificial datasets and they observed various effects of k values.

Recently, Azzeh et al (2012) attempted to learn the k value from the dataset characteristic. The k-medoid is a classical partitioning (breaking the dataset up into groups) technique of clustering that clusters the data set of n objects into k clusters known a priori. A medoid can be defined as the object of a cluster whose average dissimilarity to all the objects in the cluster is minimal. i.e. it is a most centrally located point in the cluster. They applied the Bisecting k-medoid clustering algorithm on the historical datasets without using adjustment techniques or feature selection. The main observation was that while there is no optimum static k value for all datasets, there is definitely a dynamic k values for each dataset. However, the proposed approach has a significant limitation in which they used the un-weighted mean effort of the train projects of the leaf cluster whose medoid is closest to the test project to estimate the effort for that test project. Using such cluster does not ensure that all project in it are nearest analogies. In this paper we solve that problem by proposing a more robust approach in which in this study we focus mainly on discovering the optimum set of analogies rather than guessing only number of nearest analogies for each test project.

Azzeh et al (2017) investigated that whether the obtained set of analogies works well with different kinds of adjustment techniques. So they chose three well known adjustment techniques from the literature besides mean effort adjustment to investigate the potential improvements of using their model on the adjustment techniques. The techniques investigated in this study are:

1. Similarity based adjustment: This kind of adjustment aims to calibrate the retrieved effort values based on their similarity degrees with a target project. The general form of this technique involve sum of product of the normalized aggregated similarity degrees with retrieved effort value as shown in equation 2.2.

……….. (2.2)

Where and are the estimated effort and effort of *i*th source project respectively. SM is the normalized similarity degree between two projects (SM = 1-d, where d is the normalized Euclidean distance obtained from equation 2.1, and k is the number of analogies.

1. Genetic Algorithm (GA) based Adjustment: This adjustment strategy uses GA to optimize the coefficient αj for each feature distance based on minimizing MMRE as shown in equation 2.3. The main challenge of this technique is that it needs too many parameter configurations and user interactions such as chromosome encoding, mutation and crossover which makes replication a somewhat difficult task.

….. (2.3)

Where f*xj* is the j*th* feature value of the target project. f*ij* is the j*th* feature value of the nearest project y*i*.

### 2.3 Constructive Cost Model (COCOMO)

### 2.3.1 Overview of COCOMO Model

The Constructive Cost Model (COCOMO) cost estimation model is used by thousands of software project managers and is based on a study of hundreds of software projects. Unlike other cost estimation models, COCOMO is an open model, so all of the details are published, including: the underlying cost estimation equations, every assumption made in the model, every definition (e.g. the precise definition of the Product Design phase of a project), the costs included in an estimate are explicitly stated (e.g. project managers are included, secretaries are not). COCOMO is well defined, and it doesn't rely upon proprietary estimation algorithms: It can be calibrated to reflect a software development environment, and to produce more accurate estimates.

The most fundamental calculation in the COCOMO model is the use of the Effort Equation to estimate the number of Person-Months required to develop a project.

### 2.3.2 Source Lines of Code

The COCOMO calculations are based on the estimates of a project's size in Source Lines of Code (SLOC). SLOC is defined such that only Source lines that are delivered as part of the product are included -- test drivers and other support software is excluded. Source lines are created by the project staff, code created by applications generators is excluded. One SLOC is one logical line of code, declarations are counted as SLOC and comments are not counted as SLOC.

The original COCOMO 81 model was defined in terms of Delivered Source Instructions, which are very similar to SLOC. The major difference between Delivered Source Instruction (DSI) and SLOC is that a single Source Line of Code may be several physical lines. For example, an "if-then-else" statement would be counted as one SLOC but might be counted as several DSI.

### 2.3.3 The Scale Drivers

In the COCOMO II model, some of the most important factors contributing to a project's duration and cost are the Scale Drivers. Each Scale Driver is set to describe the project; these Scale Drivers are used to determine the exponent in the Effort Equation. The five Scale Drivers are:

1. Precedentedness
2. Development Flexibility
3. Architecture / Risk Resolution
4. Team Cohesion
5. Process Maturity

It is however to be noted that the Scale Drivers have replaced the Development Mode of COCOMO 81. The first two Scale Drivers, Precedence and Development Flexibility actually describe much the same influences that the original Development Mode did.

### 2.3.4 Cost Drivers

COCOMO II has 17 cost drivers. The cost drivers are multiplicative factors that determine the effort required to complete the software project. For example, if the project will develop a software that controls an airplane's flight, the Required Software Reliability (RELY) cost driver would be set to Very High. That rating corresponds to an effort multiplier of 1.26, meaning that the project will require 26 more effort than a typical software project. COCOMO II defines each of the cost drivers, and the Effort Multiplier associated with each rating.

### 2.3.5 COCOMO II Effort Equation

The COCOMO II model makes its estimates of required effort (measured in Person-Months - PM) based primarily on the estimate of the software project's size (as measured in thousands of SLOC, KSLOC)) as show in equation 2.4

…. (2.4)

Where

EAF Is the Effort Adjustment Factor derived from the Cost Drivers

E is an exponent derived from the five Scale Drivers

As an example, a project with all Nominal Cost Drivers and Scale Drivers would have an EAF of 1.00 and exponent, E, of 1.0997. Assuming that the project is projected to consist of 8,000 source lines of code, COCOMO II estimates that 28.9 Person-Months of effort is required to complete it:

= 28.9 Person-Months

### 2.3.6 Effort Adjustment Factor

The Effort Adjustment Factor in the effort equation is simply the product of the effort multipliers corresponding to each of the cost drivers for the project. For example, if the project is rated Very High for Complexity (effort multiplier of 1.34), and Low for Language & Tools Experience (effort multiplier of 1.09), and all of the other cost drivers are rated to be Nominal (effort multiplier of 1.00), the EAF is the product of 1.34 and 1.09.

Effort Adjustment Factor = EAF = 1.34 \* 1.09 = 1.46

### 2.3.7 COCOMO II Schedule Equation

The COCOMO II schedule equation predicts the number of months required to complete the software project. The duration of a project is based on the effort predicted by the effort as shown in equation 2.5

…. (2.5)

where

Effort is the effort from the COCOMO II effort equation

SE is the schedule equation exponent derived from the five Scale Drivers

Continuing the example, and substituting the exponent of 0.3179 that is calculated from the scale drivers, yields an estimate of just over a year, and an average staffing of between 3 and 4 people:

Duration = 3.67 \* = 12.1 months

Average staffing = (42.3 Person-Months) / (12.1 Months) = 3.5 people

### 2.3.8 The SCED Cost Driver

The COCOMO cost driver for Required Development Schedule (SCED) is unique and requires a special explanation. The SCED cost driver is used to account for the observation that a project developed on an accelerated schedule will require more effort than a project developed on its optimum schedule. A SCED rating of Very Low corresponds to an Effort Multiplier of 1.43 (in the COCOMO II.2000 model) and means that you intend to finish the project in 75 of the optimum schedule (as determined by a previous COCOMO estimate). Continuing the example used earlier, but assuming that SCED has a rating of Very Low, COCOMO produces these estimates:

Duration = 75 \* 12.1 Months = 9.1 Months

Effort Adjustment Factor (EAF) = = 2.09

Effort = 2.94 \* (2.09) \* = 60.4 Person/Months

Average staffing = (60.4 Person-Months) / (9.1 Months) = 6.7 people

It is important to note that the calculation of duration is not based directly on the effort (number of Person-Months) instead it is based on the schedule that would have been required for the project assuming it had been developed on the nominal schedule. Remember that the SCED cost driver means "accelerated from the nominal schedule".

Benediktsson et al (2005) present a quantitative analytical schema for displaying effort-boxed development request to reveal the impacts on the general development effort and the potential influence that can be obtained from incremental delivery in such activities. The paper utilizes models that envisage product size as an exponential function of the development effort to investigate the connections in the middle of effort and the number of increments, along these lines giving new bits of knowledge into the financial effect of incremental methodologies to effort-boxed software projects. The authors of the article talked about the failure rate of software and proposed some ideas to improve the success rate particularly for Agile, rapid and incremental software development.

Table 2.1 Effort Estimation Models with Inverted Forms - Benediktsson et al (2005)

|  |  |  |
| --- | --- | --- |
| Model | Original Formula | Inverted Form |
| Halstead | PM=0.7KLOC1.50 | KLOC = 1.27 PM0.667 |
| Basic COCOMO organic | PM = 2.4 KLOC1.05 | KLOC = 0.43 PM0.952 |
| Basic COCOMO semidetached | PM = 3.0 KLOC1.12 | KLOC = 0.37 PM0.893 |
| Basic COCOMO intermediate | PM = 3.6 KLOC1.20 | KLOC = 0.34 PM0.833 |
| COCOMO II.2000 | PM = 2.9 KLOC1.10 | KLOC = 0.38 PM0.909 |
| Walston-Felix | PM = 5.2 KLOC0.91 | KLOC = 0.16 PM1.10 |
| Bailey-Basil | PM = 5.5 KLOC1.16 | KLOC = 0.23 PM0.862 |
| Doty for (KLOC.9) | PM = 5.2880 KLOC1.047 | KLOC = 1.27 PM0.674 |
| Albrecht and Gaffney | PM = -13.39 +0.0545 FP | FP = 245.7 +18.35 PM |
| Kermerer | PM = 60.62 \* 7.728 \* 10-8FP3 | FP = 59.76 PM0.33 |

Paper rework on the estimation model to reflect size in terms of the required effort in equation 2.6

…. (2.6)

The required effort can be likened to the project resource envelope or the effort box. The paper considers the results of taking a shot at projects with a fixed effort level where the number of augmentations and their sequencing may be used to acquire extra, serve as a major influence for project managers. The model assumes that the size of a project can be estimated in thousands of delivered source instruction and then uses a nonlinear equation to determine the effort for the project.

### 2.4 Putnam Model

The Putnam Model is an empirical software effort evaluation model. Lawrence H. Putnam in 1978 is seen as pioneering work in the field of Software Process Modeling. The Putnam model uses two equations the software manpower equation and the software productivity equation. This model is of the form:

*Technical constant* C = size \* B1/3 \* T4/3  …… (2.11)

*Total Person Months* B = 1/T4 \* (size/C)3  …… (2.12)

Here, C is a development environment dependent parameter on the and it is calculated using the historical data of the past projects, Size is estimated in LOC and T= development time required in years

For understanding, the membership function can be represented mathematically as below

According to Mahesh (2014) Putnam model is much related to development time and the person months needed for development can be greatly increased if the development time is reduced.

Narendra et al (2012) discovered a problem with the Putnam model, he argued in his article that the model was based on knowing or being able to estimate accurately the size (in lines of code) of the software to be developed. There is often great uncertainty in the software size hence resulting in the imprecision of cost estimation. An advantage of this model is that data such as size, duration and effort can be easily collected for the previous project by most of the software companies regardless of the maturity level.

Table 2.2 - Comparison of Putnam Model vs SEER-SEM

|  |  |  |  |
| --- | --- | --- | --- |
| No | Estimation Model | Advantages | Disadvantages |
| 1 | Putnam model | Based on two variables (size and duration –time) | All aspect of the SDLC aren’t considered |
| 2 | SEER-SEM | Based on the size of the software only | More parameter increases the project’s complexity, also difficulty in discovering non-linear relationships between input and corresponding output parameters. |

### 2.5 Function Point Analysis

Function Point Analysis is a standard technique for problem-solving. A function point is the unit of measurement. It is the estimate of the amount of business functionality provided to the user by an information system.

According to Khatibi et al (2010), the cost of a single unit is calculated from past projects and counting the functional points should be pre-planned. The count of distinct format or processing logic types decides the number of function points. A linear combination of five basic software components – external input and outputs, logic internal files, external inquiries and external interfaces, each being either simple, average or complex, leads to arrival of basic raw function counts.

Multiplication of an adjustment factor to the raw function count leads to the final count. 14 aspects of processing complexity are considered to arrive at this adjustment factor. The multiplication modifies the original function count by +35 or -35.

### 2.6 Soft Computing Based Effort Evaluation Techniques

The limitations of algorithmic models led to the exploration of the non-algorithmic techniques which are soft computing based. These include artificial neural network, evolutionary computation, fuzzy logic models, case-based reasoning, and combinational models and so on. Some of these techniques are reviewed in subsequent sections.

### 2.6.1 Overview of fuzzy logic

According to Kumar (2014), the term "fuzzy" refers to the fact that the logic involved can deal with concepts that cannot be expressed as the "true" or "false" but rather as "partially true". Fuzzy logic has the advantage over conventional Boolean logic in that the solution to the problem can be cast in terms that human operators can understand, so that their experience can be used in the design of the controller. This makes it easier to mechanize tasks that are already successfully performed by humans.

The development of software has always been characterized by parameters that possess a certain level of fuzziness. The application of fuzzy logic is able to overcome some of the problems which are inherent in existing effort evaluation techniques. Fuzzy logic is not only useful for effort prediction but that it is essential in order to improve the quality of current estimating models. Fuzzy logic enables linguistic representation of the input and output of a model to tolerate imprecision. It is particularly suitable for effort evaluation as many software attributes are measured on nominal or ordinal scale type which is a particular case of linguistic values, however, in many cases, it would be uncertain in making a deterministic decision of where an instance attribute belong, the fuzzy set concept would be required to justify that an instance or object belongs to a set. This set reflects an uncertain knowledge.

### 2.6.1.1 Fuzzy Set

True and False is sometimes insufficient when describing human reasoning, A fuzzy set is a set that allows its members to have different degrees of membership, called membership functions having intervals between 0 and 1, mathematically the set is bounded from below and above, i.e. [0, 1]. A fuzzy set defines value between 0 and 1, unlike the crisp set whose value is defined either as 0 and 1 or yes or no. ‘Programmers’ ability’, ‘experience’ are all examples of vague meanings in the parameters of the COCOMO model, fuzzy logic focuses only on vague predicates that are measurable. ‘Programmers’ ability’, ‘experience’ are measurable. A measurable vague predicate P can be represented by a curve (a function), fP. The curve determines the degree to which a given object satisfies certain property or not. Graded properties are represented by fuzzy sets.

A fuzzy set over a universe X is defined by its generalized membership function, usually denoted by µP(x), representing the compatibility to attribute the property P to a given element x or, in other interpretation, the possibility to soundly attribute P to x. In this latter sense, as Zadeh (1996) noted in his work, it is said that a vague predicate is characterized by its possibility distribution, i.e., by the specification of the degree to which each element of the universe is compatible with the meaning of the predicate. By way of example, in the universe U = [0, 10], the vague predicate ‘small’ can be represented by the following possibility distribution:

.... (2.13)

i.e., the compatibility of 0 with the predicate ‘small number’ is 1 or, in an alternative way, the possibility of calling 0 a ‘small number’ is 1, …, and so on.

Consider the difference between the representation of a vague predicate and a crisp one. For example, in the universe U = [0, 10], the crisp predicate P= greater or equal to five has only one possible use and the curve (membership function, f) representing it will be figure 2.1

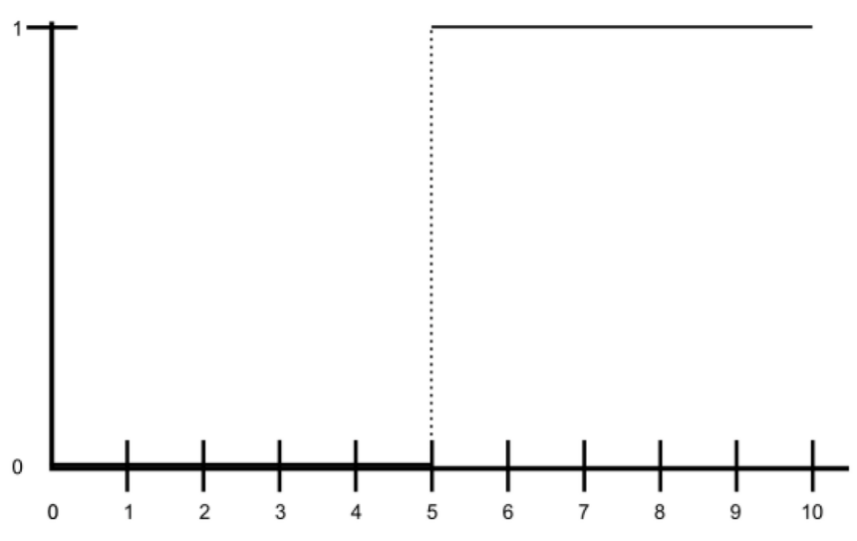


Figure 2.1 Representation of the predicate greater or equal to five.

i.e,

configuring a two-step function.

In contrast, in the universe U = [0, 100] the vague predicate P’=old can be used or interpreted slightly differently. Figure 2.2 shows three different curves that can be consistently used attending to the meaning of ‘old’

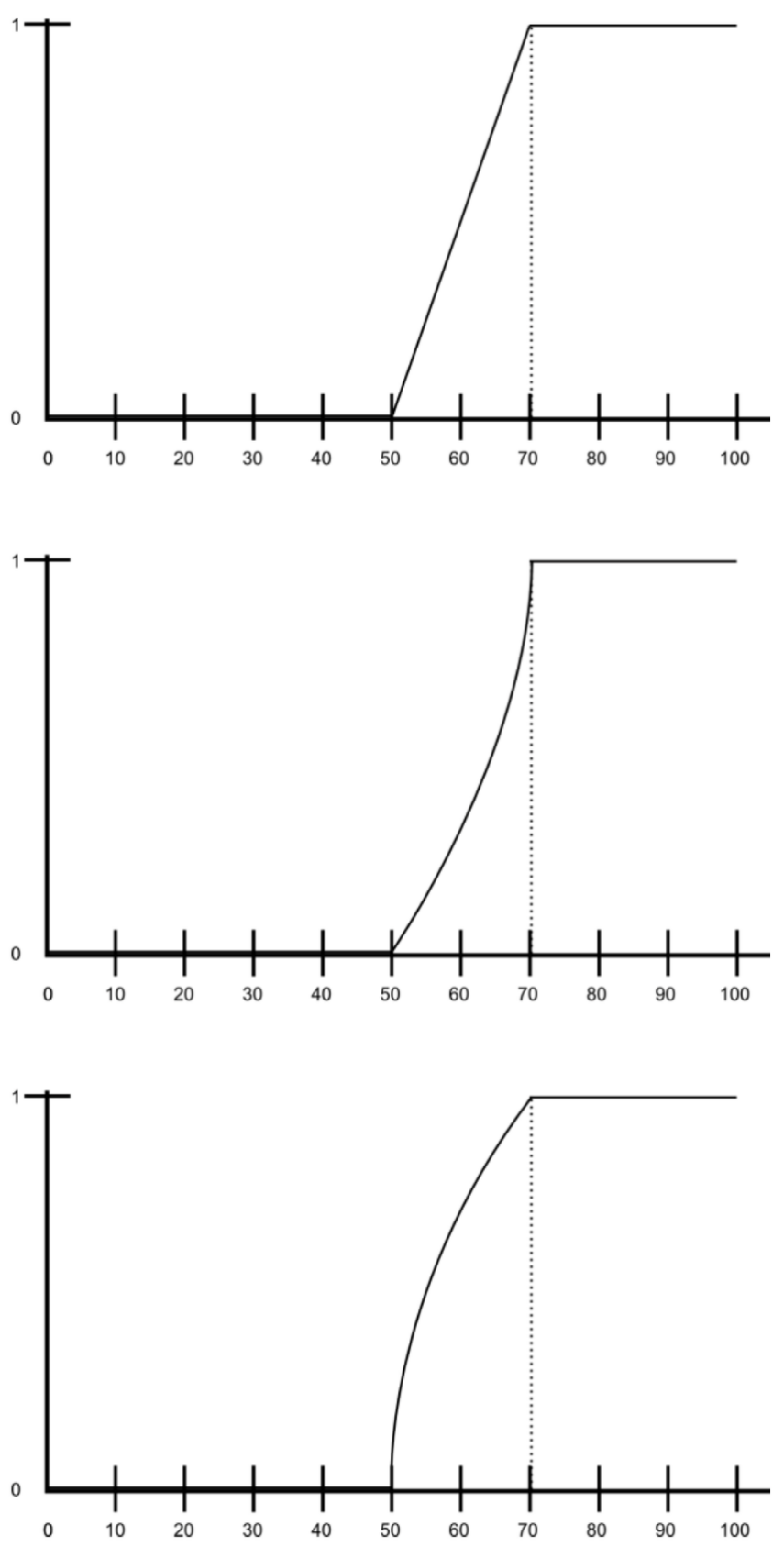


Figure 2.2 Representations of the predicate old.

#### **2.6.1.2 Fuzzy Numbers**

A fuzzy number is a quantity whose value is imprecise, rather than exact as is the case with “ordinary” (single-valued) numbers. Tomaszewska (2014) stated that the fuzzy number is an extension of a regular number in the sense that it does not refer to one single value but rather to a connected set of possible values, where each possible value has its own weight between 0 and 1. In fuzzy logic, a membership function provides a measure of the degree of similarity of an element to a fuzzy set. A membership function represents the degree of truth as an extension of valuation. The membership function fully defines the fuzzy set. Rezvani (2015) stated a generalized Fuzzy number A as a Fuzzy subset of the real line R, whose membership function *µA* satisfies the following conditions,

(a) *µA* is a continuous mapping from R to the closed interval in [0, 1].

(b) *µA*(x) = 0, −∞ < x ≤ a.

(c) *µA*(x) = L(x) is strictly increasing on [a, b],

(d) *µA*(x) = w, b ≤ x ≤ c,

(e) *µA*(x) = R(x) is strictly decreasing on [c, d],

(f) *µA*(x) = 0, d ≤ x < ∞

Where 0 < w ≤ 1 and a, b, c, and d are real numbers. We call this type of generalized fuzzy number a trapezoidal fuzzy number, and it is denoted by

…. (2.14)

A = (a, b, c, d; w) is a fuzzy set of the real line R whose membership function *µA*(x) is defined as

*µA*(x) = …. (2.15)

#### **2.6.1.3 Fuzzy control system**

A fuzzy control system is a control system based on fuzzy logic—a mathematical system that analyzes analog input values in terms of logical variables that take on continuous values between 0 and 1, in contrast to classical or conventional logic, which operates on discrete values of either 1 or 0 (true or false, respectively).

#### **2.6.1.4 Fuzzy control**

Fuzzy controllers are very simple conceptually. They consist of an input stage, a processing stage, and an output stage. The input stage maps sensor or other inputs, such as switches, thumbwheels, and so on, to the appropriate membership functions and truth values. The processing stage invokes each appropriate rule and generates a result for each, then combines the results of the rules. Finally, the output stage converts the combined result back into a specific control output value.

The processing stage is based on a collection of logic rules in the form of IF-THEN statements, where the IF part is called the "antecedent" and the THEN part is called the "consequent". Typical fuzzy control systems have dozens of rules.

Ahmad (2010) in his work considered a rule for a thermostat:

IF (temperature is "cold") THEN (heater is "high")

The rule uses the truth value of the "temperature" input, which is some truth value of "cold", to generate a result in the fuzzy set for the "heater" output, which is some value of "high". This result is used with the results of other rules to finally generate the crisp composite output. Obviously, the greater the truth value of "cold", the higher the truth value of "high", though this does not necessarily mean that the output itself will be set to "high" since this is only one rule among many. He noted that in practice, the fuzzy rule sets usually have several antecedents that are combined using fuzzy operators, such as AND, OR, and NOT, though again the definitions tend to vary: AND, in one popular definition, simply uses the minimum weight of all the antecedents, while OR uses the maximum value. There is also a NOT operator that subtracts a membership function from 1 to give the "complementary" function.

There are several ways to define the result of a rule, but one of the most common and simplest is the "max-min" inference method, in which the output membership function is given the truth value generated by the premise. Rules can be solved in parallel in hardware, or sequentially in software. The results of all the rules that have fired are "defuzzified" to a crisp value by one of several methods. There are dozens, in theory, each with various advantages or drawbacks.

The "centroid" method is very popular, in which the "center of mass" of the result provides the crisp value, Cherifi et al (2011). Another approach is the "height" method, which takes the value of the biggest contributor. The centroid method favors the rule with the output of greatest area, while the height method obviously favors the rule with the greatest output value.

Fuzzy control system design is based on empirical methods, basically a methodical approach to trial-and-error. The general process is as follows:

i. Document the system's operational specifications and inputs and outputs.

ii. Document the fuzzy sets for the inputs.

iii. Document the rule set.

iv. Determine the defuzzification method.

v. Run through test suite to validate system, adjust details as required.

vi. Complete document and release to production.

#### **2.6.1.5 Fuzzy Expert System**

Put as simply as possible, a fuzzy expert system is an expert system that uses fuzzy logic instead of Boolean logic. In other words, a fuzzy expert system is a collection of membership functions and rules that are used to reason about data. Unlike conventional expert systems, which are mainly symbolic reasoning engines, fuzzy expert systems are oriented toward numerical processing.

The rules in a fuzzy expert system are usually of a form similar to the following:

if x is low and y is high then z = medium

where x and y are input variables (names for known data values), z is an output variable (a name for a data value to be computed), low is a membership function (fuzzy subset) defined on x, high is a membership function defined on y, and medium is a membership function defined on z. The part of the rule between the "if" and "then" is the rule's premise or antecedent. This is a fuzzy logic expression that describes to what degree the rule is applicable. The part of the rule following the "then" is the rule's conclusion or consequent. This part of the rule assigns a membership function to each of one or more output variables. Most tools for working with fuzzy expert systems allow more than one conclusion per rule.

A typical fuzzy expert system has more than one rule. The entire group of rules is collectively known as a rule base or knowledge base.

#### **2.6.1.6 The Inference Process**

With the definition of the rules and membership functions in hand, we now need to know how to apply this knowledge to specific values of the input variables to compute the values of the output variables. This process is referred to as inferencing. In a fuzzy expert system, the inference process is a combination of four sub processes: fuzzification, inference, composition, and defuzzification. The defuzzification sub process is optional.

#### **2.6.1.7 Fuzzification**

In the fuzzification subprocess, the membership functions defined on the input variables are applied to their actual values, to determine the degree of truth for each rule premise. The degree of truth for a rule's premise is sometimes referred to as its alpha. If a rule's premise has a nonzero degree of truth (if the rule applies at all...) then the rule is said to fire.

#### **2.6.1.8 Inference**

In the inference subprocess, the truth value for the premise of each rule is computed and applied to the conclusion part of each rule. This results in one fuzzy subset to be assigned to each output variable for each rule.

#### **2.6.1.9 Composition**

In the composition sub process, all of the fuzzy subsets assigned to each output variable are combined together to form a single fuzzy subset for each output variable. There are two composition rules: MAX composition and SUM composition. In MAX composition, the combined output fuzzy subset is constructed by taking the pointwise maximum over all of the fuzzy subsets assigned to the output variable by the inference rule. In SUM composition the combined output fuzzy subset is constructed by taking the point wise sum over all of the fuzzy subsets assigned to the output variable by the inference rule. Note that this can result in truth values greater than one! For this reason, SUM composition is only used when it will be followed by a defuzzification method, such as the CENTROID method, that doesn't have a problem with this odd case.

#### **2.6.1.10 Defuzzification**

Sometimes it is useful to examine the fuzzy subsets that are the result of the composition process, but more often, this fuzzy value needs to be converted to a single number, which is a crisp value. This is what the defuzzification process does.

Saade et al (2004) did a short paper that compared defuzzification methods. Two of the more common techniques are the CENTROID and MAXIMUM methods. In the CENTROID method, the crisp value of the output variable is computed by finding the variable value of the center of gravity of the membership function for the fuzzy value. In the MAXIMUM method, one of the variable values at which the fuzzy subset has its maximum truth value is chosen as the crisp value for the output variable. There are several variations of the MAXIMUM method that differ only in what they do when there is more than one variable value at which this maximum truth value occurs. One of these, the AVERAGE-OF-MAXIMA method, returns the average of the variable values at which the maximum truth value occurs.

According to Rijwani et al (2014), Fuzzy logic systems are mainly categorized into three types: pure fuzzy logic systems, Takagi and Sugeno’s fuzzy system and fuzzy system with fuzzifier and defuzzifier. Fuzzifier and defuzzifier which were first proposed by Mamdani are the most widely used fuzzy logic systems, they also stated that these systems have been applied and tested on different types of consumer and end products.

### 2.6.2 Neural Networks

Neural networks are nets of processing elements that are able to learn the mapping existent between input and output data. The neuron computes a weighted sum of its inputs and generates an output if the sum exceeds a certain threshold. This output then becomes an excitatory (positive) or inhibitory (negative) input to other neurons in the network. The process continues until one or more outputs are generated. The nodes are termed simulated neurons as they attempt to imitate the functions of biological neurons. The nodes are connected together via links. This can be compared with axon synapse dendrite connections in the human brain.

The idea of developing artificial neural network (ANN) was inspired by the biological neural network that is found in the human brain. Since the introduction of simplified neurons, research has been carried out to investigate the role of the neurons in the human brain. Neurons are single units in the brain that transfer the information to the other neurons in the complex nerve networks. Using the brain as a model, computer scientists try to design and develop new platform and network that can perform computational tasks as the neurons do. The earliest research into ANNs was used to understand and model the information processing in the brain. After several promising developments in the ANN research, this machine learning technique has been applied to solve other real life problems.

The Neural Network is initialized with random weights and gradually learns the relationships implicit in a training data set by adjusting its weights when presented with these data. The network generates effort by propagating the initial inputs through subsequent layers of processing elements to the final output layer. Each neuron in the network computes a nonlinear function of its inputs and passes the resultant value along its output. Artificial neural networks are typically specified using three things.

1. Architecture: The architecture specifies what variables are involved in the network and their topological relationships—for example the variables involved in a neural network might be the weights of the connections between the neurons, along with activities of the neurons
2. Activity Rule: Most neural network models have short time-scale dynamics: local rules define how the activities of the neurons change in response to each other. Typically the activity rule depends on the weights (the parameters) in the network.
3. Learning Rule The learning rule specifies the way in which the neural network's weights change with time. This learning is usually viewed as taking place on a longer time scale than the time scale of the dynamics under the activity rule. Usually the learning rule will depend on the activities of the neurons. It may also depend on the target values supplied by a teacher and on the current value of the weights.

Arora et al (2018) in their work proposed a neural network model of 20 inputs in which 15 are effort multipliers and 5 scale factors of the COCOMO Model. They initialized bias as 1.00, weight w = 1.08, and learning rate = 0 and used identity activation function to calculate the desired output of the network. In the net diagram below, F1 – F15 are effort multipliers and F16 – F20 are scale factors.

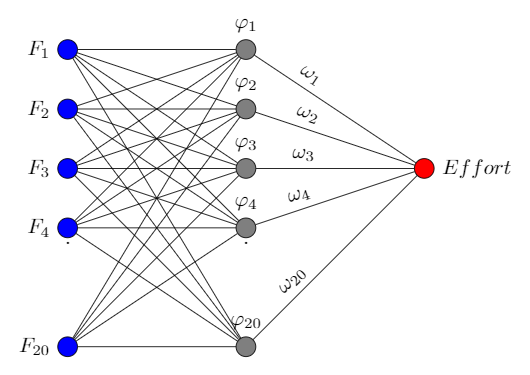


Fig 2.1 An artificial neural network

#### **2.6.2.1 Network Function**

If we consider a network system with three layers then the first layer has input neurons which send data via synapses to the second layer of neurons, and then via more synapses to the third layer of output neurons. More complex systems will have more layers of neurons, some having increased layers of input neurons and output neurons. The synapses store parameters called "weights" that manipulate the data in the calculations. An ANN is typically defined by three types of parameters:

1. The interconnection pattern between the different layers of neurons

2. The learning process for updating the weights of the interconnections

3. The activation function that converts a neuron's weighted input to its output activation.

Aberśek B (2017) in his book defined a neuron's network function f(x) as a composition of other functions gi(x), which can further be defined as a composition of other functions. This can be conveniently represented as a network structure, with arrows depicting the dependencies between variables. A widely used type of composition is the nonlinear weighted sum, where

….(2.8)

and K is commonly referred to as the activation function. It will be convenient for the following to refer to a collection of functions gi as simply a vector g = (g1, g2, …, gn).

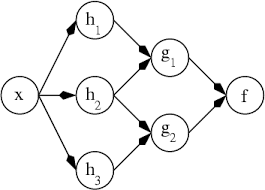


Fig 2.2 ANN Dependency Graph

This figure 2.2 depicts such a decomposition of *f*, with dependencies between variables indicated by arrows. These can be interpreted in two ways.

The first view is the functional view: the input x is transformed into a 3-dimensional vector h, which is then transformed into a 2-dimensional vector g, which is finally transformed into f. This view is most commonly encountered in the context of optimization

The second view is the probabilistic view: the random variable ) depends upon the random variable , which depends upon , which depends upon the random variable X. This view is most commonly encountered in the context of graphical models.

The two views are largely equivalent. In either case, for this particular network architecture, the components of individual layers are independent of each other (e.g., the components of g are independent of each other given their input h). This naturally enables a degree of parallelism in the implementation. Networks such as the previous ones are commonly called feed forward, because their graph is a directed acyclic graph. Networks with cycles are commonly called recurrent. Such networks are commonly depicted in the manner shown at the top of the figure, where f is shown as being dependent upon itself. However, an implied temporal dependence is not shown. For example, a neural network for handwriting recognition is defined by a set of input neurons which may be activated by the pixels of an input image. After being weighted and transformed by a function (determined by the network's designer), the activations of these neurons are then passed on to other neurons. This process is repeated until finally, the output neuron that determines which character was read is activated.

According to Divya et al (2012), proposed a multi-layer feed forward neural network model that was based on Satyanada et al (2009) architecture. He stated that the performance of a neural network is based on its architecture and their parameter settings. The neural network model proposed used the identity function at the input layer and was defined by

*f(x) = x …(2.9)*

The hidden and the output layer use a unipolar sigmoid function defined by

*f(x) = 1/ (1 + e-x) …(2.10)*

### 2.6.3 Particle Swarm Optimization (PSO) Algorithm

Particle Swarm Optimization (PSO) simulates the behaviors of bird flocking. Suppose the following scenario: a group of birds are randomly searching food in an area. There is only one piece of food in the area being searched. All the birds do not know where the food is. But they know how far the food is in each iteration. So what's the best strategy to find the food? The effective one is to follow the bird which is nearest to the food. PSO learned from the scenario and used it to solve the optimization problems. In PSO, each single solution is a "bird" in the search space, sometimes called "particle". All of particles have fitness values which are evaluated by the fitness function to be optimized, and have velocities which direct the flying of the particles. The particles fly through the problem space by following the current optimum particles.

PSO is initialized with a group of random particles (solutions) and then searches for optima by updating generations. In every iteration, each particle is updated by following two "best" values. The first one is the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called pbest. Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the population. This best value is a global best and called gbest. When a particle takes part of the population as its topological neighbors, the best value is a local best and is called lbest.

After finding the two best values, the particle updates its velocity and positions with following equation (a) and (b).  
  
…(2.11)

...(2.12)

v is the particle velocity, present is the current particle (solution), pbest[] and gbest[] are defined as stated before. rand () is a random number between (0,1). c1, c2 are learning factors. Usually c1 = c2 = 2.

#### **2.6.3.1 Particle Swarm Optimization Algorithm**

Assume that a particle i has a vector position of xij and velocity of vij,. The formula for calculating new positions of the velocity vector is shown by the following:

…(2.13)

The initial position for every particle should be shared with other particles reasonably so that they can easily stay in the group. The initial value of the velocity begins is 0 (zero), so now, v(t) = 0 The particle’s position can now be updated by the equation below.

…(2.14)

The variable is the current position of particle i, is the new position of particle i, is the current velocity, is the moving velocity, is the personal best experience of the particle, is the global best value, is the weighting function, and and are two positional random vectors and generate values between 0 and 1. The acceleration variables and are the personal acceleration and acceleration coefficient parameters, and can approximately be set to 2 for both. PSO solution space ranges within [-x, x]. Although v*i*can be set to any possible solution values, it depends on the lower bound [0, v*min*] and upper bound [0, v*max*] of the decision variable range. Kumar et al 2014 described PSO using a data flow chart. The data flow chart is shown in figure 2.3

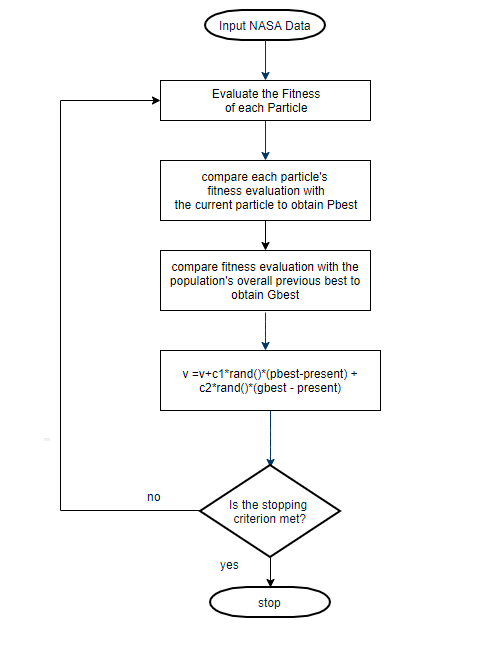


Figure 2.3 PSO Data Flow Chart

#### **2.6.3.1 Cost Constructive Model and PSO**

There are various previous works that attempted to optimize and improve the accuracy and calibrate the parameter’s value of COCOMO.

Sarno et al (2015) investigated the roles of Effort Multiplier (EM) and Line of Code (LOC) to utilize effort estimation. They applied the Gaussian Membership Function (GMF) to COCOMO II to represent the EM. They discovered that GMF could create a smoother transition, resulting to a more accurate Effort Multipliers. The proposed Neural Network (NN) model showed a major improvement, compared to pure Fuzzy model or basic COCOMO model.

Baiquni et al (2015) proposed a model based on Fuzzy Logic, Local Calibration, and Tabu Search. They tried to improve accuracy by fuzzifying cost drivers in Fuzzy Logic with Gaussian Membership Functions (GMF) to redesign the Effort Multiplier. Local Calibration as Calico and Tabu Search were used to update the value of parameters which formed new value for the calculation parameters of the COCOMO II model. The new value is able to improve accuracy and decrease error significantly.

Parkash J. et al (2014) proposed PSO technique to optimize COCOMO II model coefficients with NASA dataset. They tried to optimize coefficients a, b, c, and d of the COCOMO II model. They did an experiment using Turkey Industry dataset with 15 data points. They performed optimization on four parameters, instead of effort estimation in Post Architectural Model which requires only parameter A and B. They discovered that PSO efficiently solved the optimization problem, reduced the uncertainties, and gave better results when compared to using regular value of coefficients.

Kumar et al. (2013) analyzed the optimization of PSO with linear regression and Fuzzy Logic. They implemented this on COCOMO model with NASA18 dataset. The PSO based method gave an appropriate process in optimizing the prediction of the effort. They also compared their result with the result of linear regression method. It was observed that the later generated fairly high results with much time consumption during each process.

### 2.7 ANFIS (Adaptive Neuro-Fuzzy Inference System)

ANFIS is a framework of neuro-fuzzy model which adapts itself through learning. Its architecture consists of two types of nodes: fixed and adaptable as shown in Figure 2.9. To better understand ANFIS, the two fuzzy if-then rules are considered here.

*Rule 1: if x is A1 and y is B1 then f = p1x + q1y +r1*

*Rule 2: if x is A2 and y is B2 then f = p2x + q2y +r2*

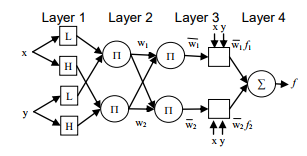
****

Figure 2.9: ANFIS architecture

The above-mentioned rules can also be rewritten in terms of membership degrees as:

*Rule 1: if µ A1(x) and µ B1(y) then f = p1x + q1y +r1*

*Rule 2: µ A2(x) and µ B2(y) then f = p2x + q2y +r2*

where Ai and Bi are MFs; and , , are the parameters of the consequent part of fuzzy rules. The nodes of layer 1 (MFs layer) and layer 4 (consequent layer) are adaptable, whereas the nodes of layer 2 (product layer) and layer 3 (normalization layer) are fixed. The five-layer architecture of ANFIS is explained as following: Layer 1: Each node i of this layer is a parameterized MF i.e., Triangle, Trapezoidal, Gaussian, or generalized Bell function. The parameters of MFs are referred to as premised parameters.

O1i = *µ Ai(x) i =1,2 (2.19)*

O1i = *µ Bi-2(y) i = 3,4 (2.20)*

For Gaussian shape of MFs (in Equation 2.21), the two parameters {c,σ} are premise parameters, which are trained during the process of learning.

*guassian(x;c,σ)* =  *(2.21)*

Layer 2: The nodes of this layer are product ∏which calculate the firing strength of a rule.

O2i = wi *µ Ai(x) µ Bi(y) i = 1, 2*

The auto-generated rules, using grid partitioning, are mn where m is the number of MFs in each input and n is the total number of inputs. Layer 3: Each node, represented as N, normalizes the firing strength of a rule by calculating the ratio of the ith rule’s firing strength to the sum of all rules’ firing strength.

O3i = i = wi/(w1 + w2) *i = 1 ,2 (2.22)*

where *wi , w1, w2*, and are ith rule’s firing strength, firing strength of first rule, firing strength of second rule, and the normalized firing strength of ith rule, respectively.

Layer 4: These nodes of this layer represent consequent part of a fuzzy rule with node function

*fi = pix + qiy +ri (2.23)*

O4i = = i (*pix + qiy +ri*) *i = 1, 2 (2.24)*

wherei is the normalized firing strength of *i*th rule, and { *pi , qi , ri* } is a first order polynomial of *i*th rule’s consequent part. The parameters { *pi , qi , ri* } are identified during the training process of ANFIS.

Layer 5: This node only does the summation of outputs of all the rules from the previous layer.

O5i = = *(2.25)*

#### **2.7.1 ANFIS Effort Based Models**

Nanda et al (2016) in their work proposed a model based on ANFIS Optimization with PSO to determine software effort estimation. They studied the 63 projects, COCOMO 81 dataset and discovered a relatively strong positive correlation (0.449) between the 15th cost drivers - Database Size (DATA) - was found in factor with the actual effort. They analyzed and obtained the result using the Pearson Correlation. The coefficient 'r' or simply put the Pearson Correlation Coefficient was calculated by the following equation:

…. (2.26)

Where x is the cost factor and y is the actual effort. is the average value of the cost factor of 63 data and is the average value of the actual effort. Similarly is the average actual effort of 63 Data.

This made them suggest that if the size 220 of the database increases, the value of effort also increases. Likewise, they also found linear associations at factor cost MODP, RELY, and TURN to the actual effort. They confirmed this using one-way ANOVA to test whether it is true for other variables. They did this to determine whether there are important differences between the means of two unrelated groups, namely the cost factor and the actual group effort.

To further get a better result, eight dominant features (DATA, MODP, RELY, TURN ACAP, AEXP, PCAP and LEXP) were selected and processed for training. Their proposed model ANFIS uses Particle Swarm Optimization (PSO) to adjust the parameters of the membership function (triangular and Gaussian).

In the first stage of the overall algorithm, they conducted ANFIS with data from the previous stage. The training process allowed the system to adjust the parameters as input/output are included and the training process was stopped when the number of epoch was reached or the number of error-rates achieved. In the second stage, a vector containing the number of parameters of membership functions with the number of dimensions N (number of membership functions) was created and was further optimized by the PSO algorithm.

Huang et al (2007) stated the use of neural network in their neuro-fuzzy COCOMO model to automatically tune the fuzzy rules from the numerical project data; fuzzy logic was used to encode expert knowledge directly using linguistic terms and neuro-fuzzy sub-model were used to calibrate the parameters of COCOMO model. They used the parameters of standard COCOMO models to initialize the neuro-fuzzy model.

#### **2.7.1.1 Triangular Membership Result**

For standard ANFIS models, training results indicate over fitting if using110 epoch. Best fitting was found in the number of training10 epoch. Error training is lowest at 4.94. While ANFIS-PSO training results showed a decrease in error until 2.22 in the epoch to 107. Since the model used is Partition Grid, then the number of rules generated is 256 rules. In the model of ANFIS Triangular, the average MSE ANFIS standards using back propagation optimization is 22.61 while the average MSE with ANFIS modification with optimization PSO is 9.30.

#### **2.7.1.2 Gaussian Membership Result**

Training results indicate over fitting if using 500 epochs. Best fitting was found in the number of training 20 epoch. Error lowest at training is 8.15. While on ANFIS models PSO error rate can still be decreased after the epoch to 500. The lowest error when training is 7.04. Because the model used is Partition Grid, then the number of rules generated is 256 rules.

In ANFIS Gaussian models, the average MSE ANFIS standards using back propagation optimization is 12:48 while the average MSE with ANFIS modification with optimization PSO is 10.63.

#### **2.7.1.3 Subtractive Clustering Result**

To ANFIS standards, training results indicate over fitting if using 150 epochs. Best fitting for standard ANFIS was recorded at exactly 2 epoch number. Error training is lowest at 0:52. As for the best-fitting ANFIS PSO was found in the number of training 150 epoch. Lowest error when training is 0:07. 40 Rule were generated by the model using Subtractive Clustering.

In Subclustering ANFIS models, the average MSE ANFIS standards using back propagation optimization is 9.31 while the average MSE with ANFIS modification with optimization PSO is 4.09.

The third stage was to determine the initiation parameter PSO. Parameters randomly initiated in the first phase and then updated by PSO algorithm. The fourth stage and final stage Parameter produced PSO then extracted to output ANFIS. The output is an effort prediction of PSO-ANFIS approach.

Ghose et al (2011) in their work use standard dataset as proposed by Lopez-Martin (2008) whose work conducted an experiment with one hundred and five small programs that were developed by thirty programmers. They used the sets of system development projects, where the Development Time (DT), Dhama Coupling (DC), McCabe Complexity (MC) and the Lines of Code (LOC) metrices were registered for 41 modules. Since all the programs were written in Pascal, the module categories mostly belong to procedures and functions. The development time of each of the forty-one modules were registered including five phases: requirements understanding, algorithm design, coding, compiling and testing. Table 2.1 shows the dataset used for carrying out experimentation. All these models were trained with first 31 inputs from the standard dataset and later 10 inputs from the same dataset were used to test the models. Already, Bhatnagar et al (2010) with the same dataset proved the effectiveness of neural network model over the regression analysis model. From these programs, three Fuzzy Logic Models were generated to estimate the effort in the development of twenty programs by seven programmers.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Module description | MC | DC | LOC | DT (m) | |
| 1 | Calculates t value | 1 | 0.25 | 4 | 13 |
| 2 | Inserts a new element in a linked list | 1 | 0.25 | 10 | 13 |
| 3 | Calculates a value according to normal distribution equation | 1 | 0.333 | 4 | 9 |
| 4 | Calculates the variance | 2 | 0.083 | 10 | 15 |
| 5 | Generates range square root | 2 | 0.111 | 23 | 15 |
| 6 | Determines both minimum and maximum values from a stored linked list | 2 | 0.125 | 9 | 15 |
| 7 | Turns each linked list value into its z value | 2 | 0.125 | 9 | 16 |
| 8 | Copies a list of values from a file to an array | 2 | 0.125 | 14 | 16 |
| 9 | Determines parity of a number | 2 | 0.167 | 7 | 16 |
| 10 | Defines segment limits | 2 | 0.167 | 8 | 18 |
| 11 | From two lists (X and Y), returns the product of all xi and yi values | 2 | 0.167 | 10 | 15 |
| 12 | Calculates a sum from a vector and its average | 2 | 0.167 | 10 | 15 |
| 13 | Calculates q values | 2 | 0.167 | 10 | 18 |
| 14 | Generates the sum of a vector components | 2 | 0.2 | 10 | 13 |
| 15 | Calculates the sum of a vector values square | 2 | 0.2 | 10 | 14 |
| 16 | Calculates the average of the linked list values | 2 | 0.2 | 10 | 15 |
| 17 | Counts the number of lines of code including blanks and comments | 2 | 0.2 | 15 | 13 |
| 18 | Prints values non zero of a linked list | 2 | 0.25 | 10 | 12 |
| 19 | Stores values into a matrix | 2 | 0.25 | 10 | 12 |
| 20 | Generates range square root | 3 | 0.08 | 3 | 17 |
| 21 | Returns the number of elements in a linked list | 3 | 0.125 | 11 | 19 |
| 22 | Calculates the sum of odd segments (Simpson’s formula) | 3 | 0.125 | 15 | 18 |
| 23 | Calculates the sum of pair segments (Simpson’s formula) | 3 | 0.125 | 15 | 19 |
| 24 | Generates the standard deviation of the linked list values | 3 | 0.143 | 13 | 21 |
| 25 | Returns the sum of square roots of a list values | 3 | 0.143 | 14 | 20 |
| 26 | Prints a matrix | 3 | 0.143 | 14 | 21 |
| 27 | Calculates the sum of odd segments (Simpson’s formula) | 3 | 0.143 | 15 | 19 |
| 28 | Calculates the sum of pair segments (Simpson’s formula) | 3 | 0.143 | 15 | 20 |
| 29 | Calculates the average of linked list values | 3 | 0.167 | 13 | 15 |
| 30 | Returns the sum of a list of values | 3 | 0.167 | 14 | 13 |
| 31 | Generates the standard deviation of linked list values | 3 | 0.2 | 18 | 19 |
| 32 | Prints a linked list | 3 | 0.25 | 9 | 13 |
| 33 | Calculates gamma value (G) | 3 | 0.25 | 12 | 12 |
| 34 | Calculates the average of vector components | 3 | 0.25 | 17 | 12 |
| 35 | Calculates the range standard deviation | 4 | 0.077 | 16 | 21 |
| 36 | Calculates beta 1 value | 4 | 0.077 | 31 | 21 |
| 37 | Returns the product between values of two vectors and the number of these pairs | 4 | 0.111 | 16 | 19 |
| 38 | Counts commented lines | 4 | 0.2 | 24 | 18 |
| 39 | Reduces final matrix (according to Gauss method) | 5 | 0.143 | 22 | 24 |
| 40 | Reduces a matrix (according to Gauss method) | 5 | 0.143 | 22 | 25 |
| 41 | Counts blank lines | 5 | 0.2 | 22 | 18 |

Ghose et al work used the standard dataset from that experiment to find out the Development Time (DT) by applying the Feed Forward Back Propagation Neural Network model, Cascaded Feed Forward Back Propagation Neural Network model, Elman Back Propagation Neural Network model, Layer Recurrent Neural Network model and Generalized Regression Neural Network model present in the Neural Network toolbox of Matlab 7.5. They obtained Development Time (DT’) for each of the trained neural network models and finally carried out a comparative analysis was based on the standard performance criterions like Magnitude of Relative Error (MRE), Mean Magnitude of Relative Error (MMRE), Balanced Relative Error (BRE) and Prediction Pred(25).

Mewada (2013) in their paper suggested a new approach for estimating of software project development time. They considered Adaptive Neuro Fuzzy Inference model and three membership functions i.e. Gaussian MF, Triangular MF and Trapezoidal. They observed that Neuro Fuzzy model using Trapezoidal membership function gives better results than all other models. They compared the following neural network models; Feed Forward Back Propagation neural network, Cascaded Feed Forward Back Propagation neural network, Layer recurrent neural network, ANFIS Model using GMF, ANFIS Model using Tri MF, ANFIS Model using Trap MF Model with using parameters MMRE (), Pred (25), and BRE ()

#### **2.7.2 Pseudo-code of ANFIS Evaluation**

Begin:

Step I; Determine the inputs of the model;

Collect a data set;

Divide the data into two sets: Train data set, and the other one for evaluating the validity of the estimated model, called the test data set.

Step II; Generate ANFIS model;

[Define no of Membership functions] numMFs;

[define type of Membership functions] mfType;

[define no epoch] epoch\_n;

[Generate Fuzzy Inference System structure from data using grid partition] in\_fis=genfis1 (trnData, numMFs, mfType);

[Training routine for Sugeno-type Fuzzy Inference System (uses a hybrid learning algorithm)] out\_fis= anfis (trnData,in\_fis,epoch\_n);

Step III; Evaluate the value of Development Time;

For each individual test data

For i=1 to total test data

[Evaluate the value of Development Time]

dt(i)= evalfis(inpData,fis);

Next i;

Step IV; Evaluate the Value of MRE from result obtained by step III;

For MRE of each individual test data

For i = 1: to total test data

mre(1,i)=abs((Actual dt(i)-dt(i))/Actual

dt(i));

Next i;

Step V; Evaluate the Value of MMRE and PRED from result obtained by step IV;

For MMRE and PRED of each individual test data

Initialize mmre =0, pred =0;

For i = 1: to total test data

mmre = mmre+mre(i);

IF (mre (i) <=.25)

pred =pred+1;

EndIF

Next i;

MMRE= (mmre/ (total test data))\*100;

PRED=pred/ (total test data);

Step VI; Evaluate the Value of BRE from result obtained by step III;

For BRE of each individual test data

For i = 1: to total test data

bre(1,i)=abs((Actual dt(i)-dt(i))/min(Actual dt(i), dt(i)));

Next i;

END

#### **2.7.3 Performance Evaluation Metrics**

Performance evaluation metrics are adapted to assess and evaluate the performance of the effort estimation models. A model which gives lower BRE is better than that which gives higher BRE. A model which gives higher Pred (n) is better than that which gives lower Pred (n). A model which gives lower MMRE is better than that which gives higher MMRE.

MMRE is better than that which gives higher MMRE.

#### **2.7.3.1 Magnitude of Relative Error (MRE)**

….(2.26)

#### **2.7.3.2 Mean Magnitude of Relative Error (MMRE)**

….(2.27)

The MMRE calculates the mean for the sum of the MRE of n projects. Specifically, it is used to evaluate the prediction performance of an estimation model.

#### **2.7.3.3 Prediction Level (PRED)**

The prediction level metrics is defined by the equation below

..…(2.28)

where l is the maximum MRE of a selected range, n is the total number of projects, and k is number of projects in a set of n projects whose MRE <= l. PRED calculates the ratio of projects’ MREs that falls into the selected range (l) out of the total projects. For example,.if n = 100, k =80, where L= MRE <= 30: PRED (30) =80/100=80).

#### **2.7.3.4 Balanced Relative Error (BRE)**

..…(2.29)

Where E= estimated effort and E’=actual effort

# CHAPTER THREE

## RESEARCH METHODOLOGY AND DESIGN

### 3.1 Description of the Existing System

In this chapter, the existing and the proposed system are analyzed, the architecture of the proposed system are presented in details. This entails having an in-depth knowledge about problems of the existing system with the purpose of improving its operations. The existing system (Neuro-Fuzzy Constructive Cost Model) carries some of the desirable features of a neuro-fuzzy approach, such as learning ability and good interpretability, while maintaining the merits of the COCOMO model. The model deals effectively with imprecise and uncertain input and enhances the reliability of software cost estimates. In addition, it allows input to have continuous rating values and linguistic values, thus avoiding the problem of similar projects having large different estimated costs.

The inputs for this model are the software size and ratings of 22 cost drivers including 5 scale factors (*SFRi*) and 17 effort multipliers (*EMRi*). The output is the software development effort estimation. Ratings of cost drivers can be continuous numerical values or linguistic terms such as ‘‘low’’, ‘‘nominal’’ and ‘‘high’’. The parameters in this model are calibrated by learning from industry project data. This system covers all important dimension of software evaluation through the integration of different technologies.

NF1

NF2

*SFR1*

**…**

*SFR2*

NF5

*SFR5*

NF6

*EMR1*

**…**

NF22

*EMR17*

*SF1*

*SF2*

*SF5*

*EM1*

*EM17*

*COCOMO II MODEL*

***Effort***

*Size*

Figure 3.1: Model of an Existing System (Kaur, 2018)

### 3.2 The Neuro-Fuzzy based Evaluation of Software System.

The neuro-fuzzy model for software development effort estimation is shown in Figure 3.1. The model is a combination of neural network and Fuzzy Inference System. The input for this model is the software size and ratings of 22 cost drivers including 5 scale factors (SFRi) and 17 effort multipliers (EMRi). The output is the optimized cost driver ratings which is used to estimate a new software development effort. From the COCOMO based dataset, ratings of cost drivers can be continuous numerical values or linguistic terms.

Each cost driver represents one factor that contributes to the development effort, such as application domain experience and product complexity. Six qualitative rating levels are used to evaluate the contribution. When expressed in linguistic terms, these six rating levels are very low (VL), low (L), nominal (N), high (H), very high (VH) and extra high (XH). Each rating level of every cost driver relates to a value called a multiplier value, which is a quantitative value used in the COCOMO model. The parameters in this model are calibrated by learning from industry project data. The result of the model was found to have good generalization capability.

There are two major components in the neuro-fuzzy model namely;

Set of Neuro fuzzy sub models (NF’s).

b. One cost driver to each sub model.

a. Set of Neuro fuzzy sub models (NF’s).

b. One cost driver to each sub model.

There are twenty-two sub-models NF*i*. For each sub-model, the input is the rating value of a cost driver, and the output is the corresponding effort multiplier value, which is used as the input of the COCOMO II model. Each NF sub model translates linguistic terms of cost drivers into qualitative multiplier values by using fuzzy sets. They are described as membership functions.

Each NF is represented by Adaptive Neuro Fuzzy inference system (ANFIS). ANFIS technique is a combination of Neural Network (NN) and Fuzzy Inference System (FIS) that considers input nodes as fuzzy sets and provides learning process for fuzzy modeling to deal with various data sets.

### 3.2.1 Sub-models

There are 22 cost drivers in the neuro-fuzzy model.

Sub-model *NFi* is used to translate the qualitative rating of a cost driver into a quantitative multiplier value and to calibrate these relations using industry project data, however, not all six rating levels are valid for all cost drivers. A natural way to represent linguistic terms is to use fuzzy sets. A fuzzy set is defined for each linguistic term of every cost driver, i.e. ‘‘very low’’, ‘‘low’’, ‘‘nominal’’, ‘‘high’’, ‘‘very high’’. The membership functions used is the triangular functions, and the universe of discourse is defined on a range of interval. The number of linguist variables were limited to 5 as most of the expert comment on extra high were insignificant.

### 3.2.2 Advantages of the Existing System

(a) Learning ability: The model has the learning/adaptation capability to model highly complex nonlinear relationships between software development effort and cost drivers.

(b) Robust to imprecise and uncertain input: The model can effectively deal with imprecise and uncertain input information, while remaining insensitive to imprecise and uncertain input such as ratings of the cost drivers. Because our model allows for continuous rating values (e.g. between one and six), it avoids the problem of similar projects with large different effort estimations

(c) Good interpretability: Although the neural network approach provides a powerful tool to model complex sets of relationships and learns from previous data, it has an inherent shortcoming: neither is it easy to understand nor is it easy to explain its decision process. However, our neurofuzzy model is clear to users during the whole decision process and its learning parameters EMi and SFi can be interpreted and validated by experts.

(d) Knowledge integration: We can integrate expert knowledge with concrete numerical project data in our model using fuzzy rules. For example, we can integrate monotonic constraints that reflect expert knowledge of cost drivers into our model to guarantee that the calibration results are reasonable.

### 3.2.3 Problem of the Existing System

The following problems were identified in the system after careful analysis.

1. The existing system only addressed effort prediction based on the existing Neuro fuzzy model.
2. The effort was accurate to an extent because COCOMO II model parameters were not optimized enough
3. The existing system considered only 18 project data set.

### 3.2.4 System Design

The system is designed as the integration of components that will enable prepossessing of the dataset, clustering of the preprocessed dataset, creation of initial Fuzzy Inference System (FIS) from the dataset, optimization of the FIS parameters by Particle Swarm Optimization algorithm, and as well as subjecting the result to the Neuro Fuzzy Model to predict the final effort required to develop a software. The architecture of the system is as shown in Figure 3.3.

The major components of the system are as follows:

1. Dataset
2. Data Preprocessor
3. Conversion using Rosetta Stone
4. Fuzzy Logic System
5. Particle Swarm Optimization (PSO) Optimizer
6. Neuro Fuzzy Model



**(ADDS 2 EFFORT MULTIPLIERS, 5 SCALE FACTORS, KSLOC)**

**PCA COMPONENT**

**UPDATE ROSETTA OUTPUT**

**15 COST DRIVERS**

**NASA DATASET – COCOMO I**

**User Manager**

**COCOMO II INPUT**

**COCOMO II INPUT**

**ANFIS**

**FUZZY INFERENCE SYSTEM**

**EVALUATION**

**HYBRID LEARNING**

**DEFFUZIFICATION**

Figure 3.3: Architecture for the proposed system

### 3.3.1 Description of the proposed system

The frame work of the proposed system is an improved work of Kaur (2018). It integrates the different components of the entire system and clearly stated their functions. We introduced a productivity coefficient into the COCOMO model to reduce the effect of productivity among team members. Hanchate (2015) stated that the productivity of employee or team is given by the ratio scope ℓ of project and performance η of the employee: … (3.1)

The productivity by COCOMO-II is given by the ratio of effort and Team Size (TS).

… (3.2)

By equations 3.1 and 3.2, we can introduce the effect of team productivity from effort by subtract the difference of the inverse of . By this, we have an improved model,

NF1

NF2

*SFR1*

**…**

*SFR2*

NF5

*SFR5*

NF6

*EMR1*

**…**

NF22

*EMR17*

*SF1*

*SF2*

*SF5*

*EM1*

*EM17*

*COCOMO II MODEL*

*-*

***Effort***

*Size*

Figure 3.2: COCOMO II Model with Team Size Productivity

The descriptions of the various components are shown in the following section.

### 3.3.2 Dataset

The dataset subsystem consists of related, discrete items from National Aeronautics and Space Administration (NASA) projects from different centers using the COCOMO I model. However, due to life cycle processes and paradigm (reuse-driven approaches, commercial off-the-shelf (COTS) life cycle developments, component software engineering approaches, use of Object-Oriented methods, etc) that have become popular since the advent of the COCOMO I model, it is necessary to convert the COCOMO I dataset to COCOMO II model using the Rosetta Stone tool. The tool was developed at IBM research in order to make COCOMO estimates functional with COCOMO II model.

### 3.3.2.1 Data gathering

It is obvious that without knowledge of the past, it is impossible to predict what may happen on future projects. (Even with knowledge of the past, there is still no guarantee that the future can be predicted.) A corollary is that if an organization wants to improve its cost estimation process, it must gather relevant data on previous projects.

Virtually all the organizations surveyed recognized the benefits that could be gained by gathering historical data to use in estimating. However, very few Software Cost Estimation organizations had an effective means of gathering data on their processes and their projects; even fewer organizations were able to apply the data gathered to improve their estimation accuracy.

The simplest way to gather data is to have a stable work force so that project and process data are maintained in the memory of the individuals of the organization. The individuals can then use this information to estimate costs of other projects. However, relying on individuals’ imperfect memories is barely sufficient for small projects; for large projects it is completely inadequate.

To overcome the limitations of relying on individuals’ memories to record project data it is necessary to have a more rigorous approach to data collection. A few organizations tried to write a summary report of each project upon completion of the project. These reports would contain a summary of the product and the process used, and any “lessons learned” associated with the project. The few organizations that attempted this type of historical record found that the record was very infrequently referred to by future project managers, and so there was little incentive to actually write these records upon completion of the project.

J

1. Usefulness of metrics: Many organizations were not clear on how to use the metrics gathered. One had instituted an extensive and expensive metric gathering program, but as yet had no idea how these metrics were to be applied to future projects. It seems that before beginning a metrics program, organizations should ask how the metrics are to be used, rather than gathering the metrics and then trying to determine what to do with them.
2. How to gather metrics: Depending on which metrics are gathered and how they are gathered, metrics can be expensive. Organizations with successful metrics programs had found a straightforward and inexpensive means of gathering the metrics through the use of automated tools. These included intelligent use of time sheets, automatic recording of problem reports and their status, coding conventions which allow SLOC and change histories to be computed automatically, etc.

### 3.3.2.3 Description of Attributes

The description of the attributes used for the prediction of software development effort is presented in Table 3.5 and Table 3.6. They are divided into domain (NASA) attributes and COCOMO attributes, and modes of development constants. The COCOMO software cost model measures effort in calendar months of 152 hours (and includes development and management hours). COCOMO assumes that the effort grows more than linearly on software size;

i.e. …3.3

Size is the estimated size in KLOC, scale factor is the combined process factors, and Effort multiplier is the combined effort factors. The constant and scale factor are domain-specific parameters and Effort Multiplier is the product of over a dozen "effort multipliers". Below is a description of the effort multiplier (attributes) and scale factor (attributes) in the NASA dataset used.

1. The effort multipliers(attributes) fall into three groups: those that are positively correlated to more effort; those that are negatively correlated to more effort; and a third group containing just schedule information. A positive correlation is a relationship between variables where all the subset variables increase or decrease together. The rate of increase may not be the same but they increase or decrease together somewhat. A negative correlation exists when subset variables have different behavior. The attributes in each of the categories are presented in Tables 3.1, 3.2, and 3.3.

Table 3.1 Positively Correlated Effort Multipliers

|  |  |
| --- | --- |
| **Effort Attribute** | **Attribute Definition** |
| Acap | Analysts capability |
| Pcap | Programmers capability |
| Aexp | application experience |
| Modp | modern programing practices |
| Tool | use of software tools |
| Vexp | virtual machine experience |
| Lexp | language experience |

Table 3.6 Effort Multiplies with no correlation

|  |  |
| --- | --- |
| **Effort Attribute** | **Attribute Definition** |
| Sced | schedule constraint |

Table 3.2 Negatively Correlated Effort Multiplies

|  |  |
| --- | --- |
| **Effort Attribute** | **Attribute Definition** |
| Stor | Memory constraint |
| Data | Database Size |
| Time | Time constraint for CPU |
| Turn | Turnaround time |
| Virt | Machine volatility |
| Cplx | Process complexity |
| Rely | Required software reliability |

b) The scalar factor is the sum of five "scale factors" which modeled issues such as Precedentedness, Flexibility, Significant risks eliminated, Team interaction process and Process maturity. They are described in Table 3.3

Table 3.3 Scalar Factors

|  |  |
| --- | --- |
| **Scalar Attribute** | **Attribute Definition** |
| Precedentedness (PREC) | Reflects the previous experience of the organization. |
| Development Flexibility (FLEX) | Reflects the degree of flexibility in the development process. |
| Risk Resolution (RESL) | Reflects the extent of risk analysis carried out. |
| Team Cohesion (TEAM) | Reflects how well the development team knows each other and work together. |
| Process Maturity (PMAT) | Reflects the process maturity of the organization. |

Table 3.4: Description of NASA based attributes

|  |  |
| --- | --- |
| **NASA ATTRIBUTES** | **DESCRIPTION** |
| Uniqueid | Defined as numbers to identify each unique project. The numbers are not continuous since the records are a subset of another NASA database. |
| Forg | States whether it is a flight or ground system |
| Center | Used to define the NASA center. Examples include 1, 2,3, 4, 5, 6 |
| year | Defines the year of development |
| Mode | Defines the development mode, whether it is embedded, organic or semidetached |

Table 3.5: Description of COCOMO attributes

|  |  |
| --- | --- |
| **COCOMO ATTRIBUTES** | **DESCRIPTIONS** |
| rely | Defined as the reliability of the system |
| Data | Defined as the database size |
| Cplx | Measures Process complexity |
| Time | Measures Time constraint for CPU |
| Stor | Measures Memory constraint |
| Virt | Measures Machine volatility |
| Turn | Measures Turnaround time |
| Acap | Measures analysts capability |
| Aexp | Measures application experience |
| Pcap | Defines programmer’s capability |
| Vexp | Measures virtual machine experience of the developers |
| Lexp | Measures language experience of the programmers |
| Modp | Measures modern programing practices of the programmer |
| Tool | Measures the programmer’s use of software tools |
| Sced | Measures schedule constraint |

Table 3.6: The numeric values of the effort multipliers (COCOMO ATTRIBUTES)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ATTRIBUTE** | **VERY LOW** | **LOW** | **NOMINAL** | **HIGH** | **VERY HIGH** | **EXTRA HIGH** | **PRODUCTIVITY**  **RANGE** |
| ACAP | 1.46 | 1.19 | 1.00 | 0.86 | 0.71 |  | 2.06 |
| PCAP | 1.42 | 1.17 | 1.00 | 0.86 | 0.70 |  | 1.67 |
| AEXP | 1.29 | 1.13 | 1.00 | 0.91 | 0.82 |  | 1.57 |
| MODP | 1.24 | 1.10 | 1.00 | 0.91 | 0.82 |  | 1.34 |
| TOOL | 1.24 | 1.10 | 1.00 | 0.91 | 0.83 |  | 1.49 |
| VEXP | 1.21 | 1.10 | 1.00 | 0.90 |  |  | 1.34 |
| LEXP | 1.14 | 1.07 | 1.00 | 0.95 |  |  | 1.20 |
| SCED | 1.23 | 1.08 | 1.00 | 1,04 | 1.10 |  | E |
| STOR |  |  | 1.00 | 1.06 | 1.21 | 1.56 | -1.21 |
| DATA |  | 0.94 | 1.00 | 1.08 | 1.16 |  | -1.23 |
| TIME |  |  | 1.00 | 1.11 | 1.30 | 1.66 | -1.30 |
| TURN |  | 0.87 | 1.00 | 1.07 | 1.15 |  | -1.32 |
| VIRT |  | 0.87 | 1.00 | 1.15 | 1.30 |  | -1.49 |
| RELY | 0.75 | 0.88 | 1.00 | 1.15 | 1.40 |  | -1.86 |
| CPLX | 0.70 | 0.85 | 1.00 | 1.15 | 1.30 | 1.65 | -1.87 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mode | a | b | c | d |
| Organic | 2.4 | 1.05 | 2.5 | 0.38 |
| Semi-detached | 3 | 1.12 | 2.5 | 0.35 |
| Embedded | 3.6 | 1.20 | 2.5 | 0.32 |

### 3.3.3 Data Preprocessing

The software development effort evaluation process extracts knowledge from original data and it begins with the original database from which the knowledge will be extracted. This original data will be used for the whole evaluation process. Before the data undergoes mining, they must be prepared in a preprocessing step that removes or reduces noise and handles missing values. Relevance analyses for omitting unnecessary and redundant data, as well as data transformation, are needed for generalizing the data to higher-level concepts.

### 3.3.4 Principal Component Analysis (PCA)

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components. The NASA dataset passes the 22 input to a COCOMO model to get the effort required to develop a software. However, many of these features measure related properties and so will be redundant. There is therefore need to construct some new characteristics from the existing dataset that turn out to summarize the dataset. PCA finds the best possible value for an input that summarize the data as well as only possible (among all linear combinations of attributes).

The dimension reduction will be on the COCOMO I set from NASA before conversion. This means the 15 Effort multipliers in the COCOMO I set will be reduced from 15. For design sake, we will use the first two data points for reduction. The mathematical principle behind dimension reduction is to find a vector that defines the surface to which we wish to project our data.

### 3.3.4.1 PCA Algorithm

1. Reduce data n-dimension to k-dimension where n>k>0

(a) Compute Covariance matrix

Let the covariance matrix be given as sigma ∑, an n by n matrix.

4

Where is a 15 by 1 matrix which represents a point in the COCOMO I set, and is a 1 by 15 matrix. The later is the transpose of the X matrix. This is achieved by making the row column and the column rows. The product of a 15 x 1 matrix with a 1 by 15 matrix is a 15 by 15 matrix.

b) Compute Eigen vector of sigma: The first output of the eigen vector is also an n by n matrix and that is the space that reduces the dimension.

The first eleven data point in the dataset were used to illustrate the workings behind PCA is given in Table 3.7.

Table 3.7 15 COCOMO I sample for PCA

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1.15 | 0.94 | 1.15 | 1.00 | 1.00 | 0.87 | 0.87 | 1.00 | 1.00 | 1.00 | 1.00 | 0.95 | 0.91 | 1.00 | 1.08 |
| 2 | 1.15 | 0.94 | 1.15 | 1.00 | 1.00 | 0.87 | 0.87 | 1.00 | 1.00 | 1.00 | 1.00 | 0.95 | 0.91 | 1.00 | 1.08 |
| 3 | 1.15 | 0.94 | 1.15 | 1.00 | 1.00 | 0.87 | 0.87 | 1.00 | 1.00 | 1.00 | 1.00 | 0.95 | 0.91 | 1.00 | 1.08 |
| 4 | 1.15 | 0.94 | 1.15 | 1.00 | 1.00 | 0.87 | 0.87 | 1.00 | 1.00 | 1.00 | 1.00 | 0.95 | 0.91 | 1.00 | 1.08 |
| 5 | 1.15 | 0.94 | 1.15 | 1.00 | 1.00 | 0.87 | 0.87 | 1.00 | 1.00 | 1.00 | 1.00 | 0.95 | 0.91 | 1.00 | 1.08 |
| 6 | 1.15 | 0.94 | 1.15 | 1.00 | 1.00 | 0.87 | 0.87 | 1.00 | 1.00 | 1.00 | 1.00 | 0.95 | 0.91 | 1.00 | 1.08 |
| 7 | 1.15 | 0.94 | 1.15 | 1.00 | 1.00 | 0.87 | 0.87 | 1.00 | 1.00 | 1.00 | 1.00 | 0.95 | 0.91 | 1.00 | 1.08 |
| 8 | 1.15 | 0.94 | 1.15 | 1.00 | 1.00 | 0.87 | 0.87 | 1.00 | 1.00 | 1.00 | 1.00 | 0.95 | 0.91 | 1.00 | 1.08 |
| 9 | 1.15 | 0.94 | 1.15 | 1.66 | 1.56 | 0.87 | 1.07 | 0.86 | 0.91 | 0.86 | 1.00 | 0.95 | 0.91 | 0.91 | 1.00 |
| 10 | 1.00 | 0.94 | 1.15 | 1.00 | 1.00 | 0.87 | 0.87 | 0.86 | 0.82 | 0.70 | 1.00 | 0.95 | 1.00 | 1.00 | 1.00 |
| 11 | 1.00 | 0.94 | 1.15 | 1.00 | 1.00 | 0.87 | 0.87 | 0.86 | 0.82 | 0.86 | 1.00 | 0.95 | 1.00 | 1.00 | 1.00 |

The mean of the above input is given in Table 3.8. It is calculated by using equation

Table 3.8: Mean (X-X̅) of the 15 Cost Drivers



Table 3.9 shows the transpose of the Matrix Table 3.8, the transpose changed from a 11x15 matrix to a 15x11 matrix.

Table 3.9 Transpose Matrix of Dataset



To estimate of the covariance matrix between each cost driver variable of the COCOMO dataset in terms of the observation vectors, the sample covariance is given in Table 3.10

Table 3.10: Covariance of the dataset matrix 

To calculate the correlation matrix, we must divide the Covariance by the Standard deviation of the related matrix. Mathematically, the correlation matrix is given in equation 3.7

is the standard deviation or how each point in the set varies from one another. Si is usually denoted by . Mathematically, the standard deviation format is written as

The computed correlation matrix is given in Table 3.11

Table 3.11: Correlation Set Matrix



From the above correlation matrix, all the values on the main diagonal are 1 because the variances have been standardized. For design sake, the eigen values of the correlation matrix calculated is given in Table 3.12.

Table 3.12: Eigen Values of Correlation Matrix.



The Eigen vector is given in Table 3.13

Table 3.13: Eigen Vector of the Correlation Matrix.



The coefficients of the eigenvectors serve as the regression coefficients of the 15 principal components. For example, the first principal component can be expressed by

We can go ahead and compute the PC using the equation PC= XTXS, i.e. the product of the X transpose and the standardized X. However, for dimension reduction, we will quickly arrange the eigen vector in their descending order to see how each variance was accounted for by the eigen vector and some important information in Table 3.14.

Table 3.14: Eigen value summary

|  |  |  |
| --- | --- | --- |
| **Eigen Value** | **Percentage Composition** | **Cumulative Percentage** |
| 5.1849 | 34.57% | 34.57% |
| 3.4028 | 22.69% | 57.26% |
| 2.5140 | 16.76% | 74.02% |
| 1.1960 | 7.97% | 81.99% |
| 0.4576 | 3.05% | 85.04% |
| 0.2837 | 1.89% | 86.93% |
| -0.0997 | -0.67% | 86.26% |
| 0.0606 | 0.40% | 86.66% |
| 2.0000 | 13.33% | 99.99% |
| 6.04E-16 | 4.03e-15% | 99.99% |
| -2.43E-16 | -1.62e-15% | 99.99% |
| -3.14E-17 | -2.09e-16% | 99.99% |
| -1.11E-17 | -7.4e-17% | 99.99% |
| 3.85E-18 | 2.57e-17% | 99.99% |
| -1.21E-32 | -8.07e-32 | 99.99% |
| **Total = 15** |  |  |

The graph for the data in Table 3.14 is shown in Figure 3.4

Figure 3.4: Variable Reduction Series from Eigen Values

From the above chart we can see that the principal component determinant which is the Eigen vector favors only the rely, data, cplx, time, and aexp based on their percentage composition. So, we reduce the dimensions from 15 to 5 in the original COCOMO I file.

### 3.3.5 Rosetta Stone

The Rosetta Stone is a tool that permit update of original COCOMO 81 files so that they can be used with COCOMO II model. The tool translates files to a form that is compatible with COCOMO II. Table 3.15 – Table 3.18 illustrates how to convert factors in the COCOMO equations (i.e., the exponent, the size estimate and the ratings for the cost drivers) from COCOMO I to the COCOMO II model. The following four steps are required to convert COCOMO I to the COCOMO II model:

1. **Update size**: Table 3.15 provides guidelines for converting size in Delivered Source Instruction (DSI) to Source Line of Code (SLOC) for their use with the COCOMO II model.

Table 3.15 Converting Size Estimates

|  |  |
| --- | --- |
| **COCOMO 81** | **COCOMO II** |
| - DSI  **-** 2nd generational Languages  - 3rd generational languages  - 4th generational languages  - object oriented languages | - SLOC3  - reduce DSI by 35%  - reduce DSI by 25%  - reduce DSI by 40%  - reduce DSI by 30% |

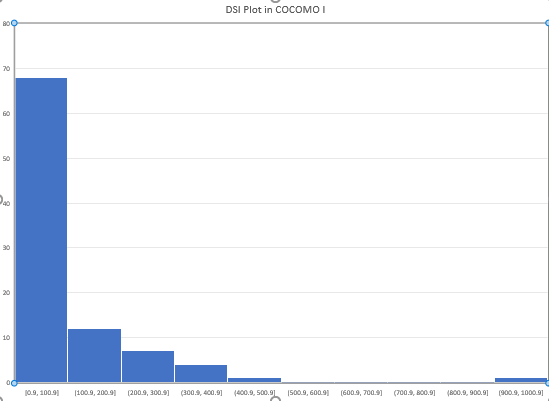


Figure 3.5: DSI plot in COCOMO I Dataset

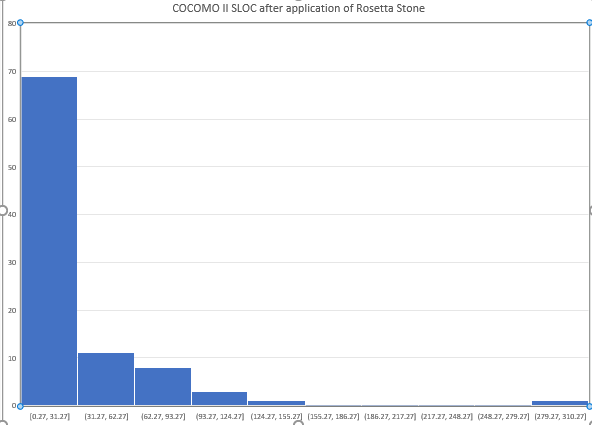


Figure 3.6: SLOC in COCOMO II after conversion.

1. **Convert Exponent:** The exponent variable in COCOMO II are five scale factors which are absent in COCOMO I. They are Precedentedness (PREC), Development Flexibilty (FLEX), Architecture/Risk Resolution (RESL), Team Cohesion (TEAM), and Process Maturity (PMAT). Table 3.16 presents values used for the five scale factors that determines the exponent of COCOMO II.

Table 3.16 Mode/Scale Factor Conversion Ratings

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MODE/SCALE FACTORS** | **ORGANIC** | **SEMI-DETACHED** | | **EMBEDDED** |
| Precedentedness (PREC) | XH | | H | L |
| Development Flexibilty (FLEX) | XH | | H | L |
| Architecture/Risk Resolution (RESL) | XH | | H | L |
| Team Cohesion (TEAM) | XH | | VH | N |
| Process Maturity (PMAT) | MODP | | MODP | MODP |

From the above, PMAT replaces the Modern Programming Practice (MODP) cost driver in COCOMO I model. A Modern Programming Practices (MODP) cost driver from Table 3.2 with ratings VL or L translate into a PMAT rating of VL, or a low level on the SEI CMM scale. A MODP rating of N translates into a PMAT rating of L, or a high Level 1. A MODP rating of H or VH translates into a PMAT rating of N or CMM Level 2. As with the other factors, if you know that the project’s actual rating was different from the one provided by the Rosetta Stone, use the actual value. This was achieved in excel using this formula

*=IF (OR (T2="vl”, T2="l"),"vl”, IF(OR(T2="h",T2="vh"),"n",IF(T2="n","l")))*

Where T2 is the MODP cost driver.

1. Rate Cost Drivers: The Rosetta Stone guidelines in Table 3.17 are used to convert COCOMO I cost drivers to COCOMO II.

Table 3.17 Cost Driver Conversion

|  |  |  |
| --- | --- | --- |
| **COCOMO 81 DRIVERS** | **COCOMO II DRIVERS** | **CONVERSION FACTORS** |
| RELY | RELY | Same as actual |
| DATA | DATA | Same as actual |
| CPLX | CPLX | Same as actual |
| TIME | TIME | Same as actual |
| STOR | STOR | Same as actual |
| VIRT | PVOL | Same as actual |
| TURN |  | Use values in Table 3.4 |
| ACAP | ACAP | Same as actual |
| PCAP | PCAP | Same as actual |
| VEXP | PEXP | Same as actual |
| AEXP | AEXP | Same as actual |
| LEXP | LTEX | Same as actual |
| TOOL | TOOL | Use values in Table 3.4 |
| MODP | Adjust PMAT settings | IF MODP is rated VL or L, set PMAT to VL, else if N, set PMAT to L, else if H or VH, set PMAT to N |
| SCED | SCED | Same as actual |
|  | RUSE | Set to N, or actual if available |
|  | DOCU | If mode = organic, set to L, if semi-detached, set to N, if mode = embedded, set to H |
|  | PCON | Set to N, Same as actual if available |
|  | SITE | Set to H, Same as actual if available |

Table 3.18 TURN and TOOL Adjustments

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| COCOMO II MULTIPLIER/COCOMO 81 RATING | VL | L | N | H | VH |
| TURN |  | 1.00 | 1.15 | 1.23 | 1.32 |
| TOOL |  |  | 1.24 | 1.10 | 1.00 |

### 3.4 Adaptive Neuro-Fuzzy Inference System

The Adaptive Neuro-Fuzzy Inference System (ANFIS) is adopted for each NFi. The first five cost drivers are those denoted by CDi = SFi for 1i5 and the remaining 17 cost drivers are denoted by CDi+5 = EMi, i = 1, 2, ..., 17. The input of NFi is the rating value CDRik of the ith cost driver CDi, and the output is the corresponding multiplier value CDi.

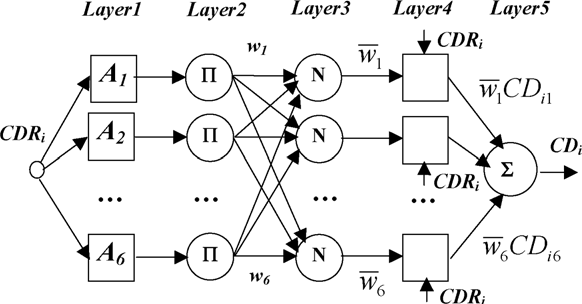


Figure 3.7: ANFIS Structure of sub-model NF*i*

Figure 3.6 shows the ANFIS structure of sub-model NFi for a 1-input-1- output system that is functionally equivalent to a Takaki and Sugeno’s type of fuzzy system. The ANFIS model super imposes Fuzzy Inference System on a Neural Network. It has Five Layers.

**The Fuzzification layer:** During fuzzification the crisp values are given as a vector for the importance of each cost driver by the domain expert. This vector of importance is also called the fuzzy set. In our case the domain expert is the COCOMO ratings.

The outputs of layer 1 are the fuzzy membership grade of the inputs. For example, the output of layer 1 for node RELY (1,1) is the membership function for the RELY (1,1) node attribute. This value shows the degree to which given data point satisfies the fuzzy set {very low}. Since the first fuzzy set is very low.

we represented mathematically by:

for i = 1, 2,3, 4,5 … (3.10)

where x is the input to the node k, and Aik is the linguistic label (high, low, etc.) associated with this node function

Expanding the for loop gives

, etc

is a triangular membership function.

With the above equation, we can compute all the membership functions for RELY inputs using a and b as boundaries in Table 3.1. We denote the degree of importance of cost driver i belonging to the importance level k as . Each node in the layer 1 is an adaptive node. For example; the first data input in the network is RELY. The input range for RELY is given mathematically as

The values of fuzzy importance for RELY, and CPLX is given in Table 3.19 and 3.20 respectively

Table 3.19: Range of Values for RELY fuzzy set.

|  |  |
| --- | --- |
| **Range of Values** |  |
| Very Low | 0 – 0.82 |
| Low | 0.82 - 0.92 |
| Nominal | 0.92 - 1.00 |
| High | 1.00 – 1.10 |
| Very High | 1.10 – 1.26 |

Table 3.20: Range of Values for CPLX fuzzy set

|  |  |
| --- | --- |
| **Range of Values** |  |
| Very Low | 0 – 0.73 |
| Low | 0.73 - 0.87 |
| Nominal | 0.87 - 1.00 |
| High | 1.00 – 1.17 |
| Very High | 1.17 – 1.34 |

Table 3.21: Range of Values for ACAP fuzzy set

|  |  |
| --- | --- |
| **Range of Values** |  |
| Very Low | 1.19 – 1.42 |
| Low | 1.00 - 1.19 |
| Nominal | 0.85 - 1.00 |
| High | 0.71 - 0.85 |
| Very High | 0 – 0.71 |

**The Membership Function Layer:** 22 Scale Factor rating from the dataset are fuzzified and passed into the Membership function layer. Here, five levels of importance (l = 5) are represented by five fuzzy sets; The MF for the input reliability is shown in the Figure 3.7.

Fig 3.8: RELY Triangular Membership Function

**Rule Based Layer:** The rules in this layer have two part namely the antecedent part and the consequent part. The number of rules is the number of neurons in the Rules Layer. The domain expert says an increase of ACAP, PCAP, and LTEX will decrease effort.

Table 3.22: Rule Formulation for ACAP, PCAP, and LTEX

|  |  |  |
| --- | --- | --- |
| **ACAP** | **PCAP** | **LTEX** |
| 1 | 1 | 1 |
| 1 | 1 | 2 |
| 1 | 1 | 3 |
| 1 | 1 | 4 |
| 1 | 1 | 5 |
| **…** | | |
| 5 | 1 | 1 |
| 5 | 1 | 2 |
| **…** | **…** | **…** |
| 5 | 5 | 5 |

For five linguistic variables, we considered the highest number obtainable when three inputs are summed since the problem is to minimize effort (number of persons per month). It gives 14 and 15. So we make rules with accepted combinations. Table 3.23 illustrates the rules formulation machine.

Table 3.23: Rules Machine Table for ACAP, PCAP and LTEX

|  |  |  |  |
| --- | --- | --- | --- |
| **ACAP** | **PCAP** | **LTEX** | **Accept/Reject** |
| 1 | 4 | 1 | Reject |
| 5 | 5 | 5 | Accept |
| 5 | 5 | 4 | Accept |
| 5 | 4 | 5 | Accept |

So, applying the following accepted rules in Table 3.4 we can create rules for ACAP, PCAP and LTEX

*R1 if (acap is very high) AND (pcap is very high) AND (ltex is very high) then effort is reduced.*

*R2if (acap is very high) AND (pcap is very high) AND (ltex is high) then effort is reduced.*

*R3if (acap is very high) AND (pcap is high) AND (ltex is very high) then effort is reduced.*

*R4if (acap is high) AND (pcap is very high) AND (ltex is very high) then effort is reduced.*

Transposing the rules into weights forms the output of the Rule Layer of ANFIS. The membership function acts like an activation function in a conventional neural network. To calculate the respective neuron weight for the rules, the three attributes above represent input 15, 16 and 20 respectively. So, we compute membership function for MF (15,5), MF (15,4), MF (16,5), MF (16,4), MF (20,5) and MF (20,4)

From the dataset, row 1 = 1. In Table 3.3 a and c is given as an interval [1.00 1.19], b = (a+c)/2 which is the mid-point of the interval. The value of b = 1.095, since x =1 lies between a and b, we compute the membership function using the first equation

The same way we computed the remaining membership functions and use in the implication AND which means the minimum of the sets to compute the value of the neuron. The node in this layer is a circle node labelled multiplies the incoming signals and outputs the product. The output of each node represents the firing strength of a rule.

**Normalization Layer:** This layer checks that the activation function or firing strength derived in the previous layer are always normalized. The kth (normalization) node calculates the ratio of the kth rule’s firing strength to the sum of all rules’ firing strengths:

… (3.11)

**Defuzzification Layer:** Every node k in this layer is a square node with a node function:

…. (3.12)

where is the output of layer 3, and {} is the parameter set. Parameters in this layer are referred to as consequent parameters.

Layer 5: The single node in this layer is a circle node labelled S that computes the overall output as the summation of all incoming signals, i.e.

… (3.13)

In summary, the overall output of sub-model NF is:

… (3.14)

Where = , … (3.15)

### 3.5 Fuzzy Inference System (FIS)

The Fuzzy Inference System (FIS)

Fuzzy input

Defuzzification Interface

Output

Fuzzification Interface

Crisp Input

Processing Rules

Membership Functions

Knowledge base

Database

Rule base

Inference Engine

Fuzzy Crisp Output

Fig. 3.4: Fuzzy Logic Model for Software Development Effort Evaluation

### 3.5.1 Description of Fuzzy Logic Model

The fuzzy Logic model for software development effort evaluation can be envisioned as involving a knowledge base and a processing stage. The knowledge base provides membership functions and the fuzzy rules needed for the process. Numerical crisp variables are the input of the system in the processing stage. These variables are passed through a fuzzification stage where they are transformed to linguistic variables, which become the fuzzy input of the inference engine. This fuzzy input is transformed by rules of the inference engine to fuzzy output. These linguistic results are then changed by a defuzzification stage into numerical values that becomes the output of the system.

### 3.5.2 Inference Process of Fuzzy Expert Model

A fuzzy expert system is an expert system that uses fuzzy logic instead of Boolean logic. In other words, a fuzzy expert system is a collection of membership functions and rules that are used to reason about data.

A typical fuzzy expert system has more than one rule. The entire group of rules is collectively known as a rule-based or knowledge base. With the definition of the rules and membership functions in hand, we now need to know how to apply this knowledge to specific values of the input variables to compute the values of the output variables. This process is referred to as inferencing. In a fuzzy expert system, the inference process is a combination of four subprocesses: fuzzification, inference, composition, and defuzzification. The defuzzification subprocess is optional.

### 3.5.3 Fuzzification

In the fuzzification subprocess, the membership functions defined on the input variables are applied to their actual values, to determine the degree of truth for each rule premise. The degree of truth for a rule's premise is sometimes referred to as its alpha. If a rule's premise has a nonzero degree of truth (if the rule applies at all...) then the rule is said to fire.

### 3.5.4 Inferencing

In the inference subprocess, the truth value for the premise of each rule is computed and applied to the conclusion part of each rule. This results in one fuzzy subset to be assigned to each output variable for each rule.

# CHAPTER FOUR

**RESULTS AND DISCUSSION**

## 4.1 Results

In this dissertation, Principal Component Analysis (PCA) is used to reduce the input feature vector which is then fed to Adaptive neuro-fuzzy inference system. In the next section, complete results of PCA are discussed which includes the scores generated by PCA against each output and the explained variance of these scores. The reduced feature vector consists of those minimum features which have maximum contribution in overall accuracy of the system. These features are then provided to ANFIS for further prediction. These results are presented in Sections 4.1.1, and 4.1.2. The discussion of the result and system evaluation is presented in Section 4.2.

### 4.1.1 Feature Extraction (PCA) Result

In order to eradicate redundant input from feature vector and sort out the best features that contains most applicable pieces of information from the given dataset, Principal component analysis was used. In this study, a total of 23 features were extracted in the initial stage. These features were then fed to PCA to extract most valuable features. The results show that out of the 23 features, 17 features that are effort multipliers have less than 2% explained variance and 6 features that are scalar multipliers have above 2% explained variance. The 6 features have cumulatively 94.1% explained variance. The cumulative explained variance value is presented in Table 4.1 and the scree plot for the cumulative explained variance is depicted in Figure 4.1.

Table 4.1: Effect of dimension on the cumulative explained variance

|  |  |
| --- | --- |
| **Feature** | **Cumulative Explained Variance (%)** |
| RELY | 63.3385 |
| DATA | 16.4827 |
| CPLX | 5.8788 |
| RUSE | 3.4049 |
| DOCU | 2.7814 |
| TIME  STOR  PVOL  ACAP  PCAP  PCON  AEXP  PEXP  LTEX  TOOL  SITE  SCED  PREC  FLEX  RESL  TEAM  PMAT  SLOC | 2.23903  1.60532  1.43192  0.75292  0.60549  0.56600  0.45995  0.29948  0.15349  5.89E-28  2.52E-33  1.01E-34  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000 |

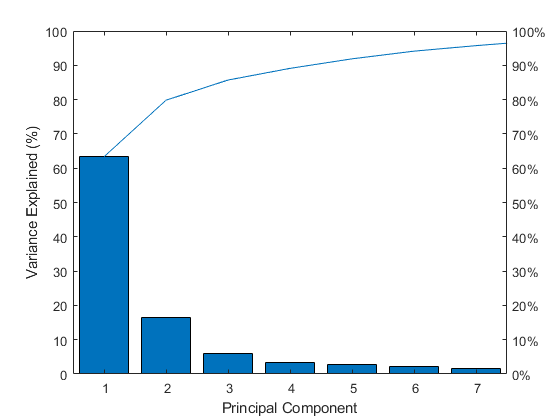


Figure 4.1: Cumulative scree plot of the original dataset

From the results shown in Figure 4.1, we have selected first six principal components as new feature vector because these six principal components have total cumulative variance of 94.1. Hence the new feature vectors are RELY, DATA, CPLX, RUSE, DOCU, and TIME. The algorithm of PCA calculates these scores by making use of covariance matrix, eigenvector, and eigenvalues from the original feature vector. The number of dimensions used was six (6) because six (6) is the minimum of the dimension that explains more than 94.1% variance in the original dataset. Hence the new (or reduced) dataset had a dimension of six (6). The PCA algorithm achieves this by calculating the covariance matrix, eigenvector, and eigenvalues of the original dataset and used this information to reduce the dimension in the original dataset without losing much of the dataset’s original information. The covariance matrix produced by PCA is presented in Table 4.2, while the eigenvector and the eigenvalues is presented in Table 4.3 and Table 4.4 respectively.

Table 4.2: 5 by 5 Extraction of the 23 by 23 PCA covariance matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **RELY** | **DATA** | **CPLX** | **RUSE** | **DOCU** | **TIME** |
| 0.0461 | 0.1744 | 0.2002 | -0.0136 | 0.0920 | 0.2485 |
| -0.0177 | 0.2132 | -0.3926 | -0.5570 | -0.3759 | -0.0440 |
| 0.0586 | 0.3906 | 0.7778 | -0.1118 | -0.2005 | -0.2869 |
| -1.4E-17 | -5.6E-17 | 0.0E+00 | -6.9E-17 | -5.6E-17 | 5.6E-17 |
| 0.0275 | 0.1145 | 0.0689 | 0.0244 | 0.0920 | -0.0391 |

Table 4.3: A 4 by 4 Extraction of the 23 by 23 PCA eigenvector

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **RELY** | **DATA** | **CPLX** | **RUSE** | **DOCU** | **TIME** |
| -6.4E-16 | -4.1E-16 | 3.7E-17 | -1.9E-01 | 2.1E-01 | 1.2E-01 |
| -4.0E-17 | 1.9E-16 | -4.2E-18 | 4.6E-02 | 1.9E-02 | -3.0E-01 |
| 2.5E-16 | -1.3E-16 | -6.0E-18 | -3.4E-02 | -3.6E-03 | 6.8E-02 |
| -6.2E-03 | -3.6E-02 | -1.0E+00 | -1.7E-18 | -3.5E-18 | -1.4E-17 |
| 3.6E-15 | 1.7E-15 | -5.0E-17 | 8.3E-01 | 7.2E-02 | -8.7E-02 |

Table 4.4: PCA eigenvalues

|  |  |
| --- | --- |
| **Variable Name** | **Eigenvalue** |
| Required Software Reliability (RELY) | 0.2626 |
| Database Size (DATA) | 0.0683 |
| Process Complexity (CPLX)  Required Reusability (RUSE) | 0.0244  0.0141 |
| Documentation Match to life-cycle needs (DOCU) | 0.0115 |
| Execution Time (TIME) | 0.0093 |
| Main Storage Constraint (STOR) | 0.0115 |
| Platform Volatility (PVOL)  Analyst Capability (ACAP)  Programmer Capability (PCAP)  Personnel Continuity (PCON)  Applications Experience (AEXP)  Platform Experience (PEXP)  Language & Tool Experience (LTEX)  Use of Software Tools (TOOL)  Multisite Development (SITE)  Required Development Schedule (SCED) | |  | | --- | | 0.0031 | | 0.0025 | | 0.0023 | | 0.0019 | | 0.0012 | | 0.0006 | | 2.4E-30 | | 1.0E-35 | | 4.2E-37 | |

The eigenvalues in Table 4.4 can be visualized in Figure 4.2.

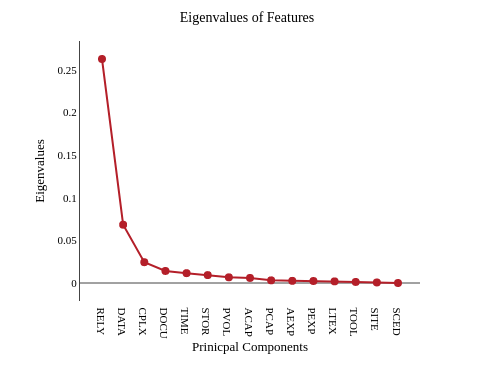


Figure 4.2: PCA Eigenvalues

PCA feature selection is done by selecting components with the maximum eigenvalue. In this case RELY, DATA, CPLX, RUSE, DOCU and TIME are chosen. This implies that RELY is components 1, DATA is component 2, CPLX is component 3, RUSE is component 4, DOCU is component 5 and TIME is component 6.

The amount of information in the original dataset retained by the reduced dataset is presented in Table 4.5.

Table 4.5: Explained Variance

|  |  |
| --- | --- |
| **PCA component** | **Explained Variance** |
| Required Software Reliability (RELY) | 63.34 |
| Database Size (DATA) | 16.48 |
| Process Complexity (CPLX)  Require Reusability (RUSE)  Documentation Match to life-cycle needs (DOCU)  Execution Time (TIME) | 5.87  3.41  2.78  2.23 |

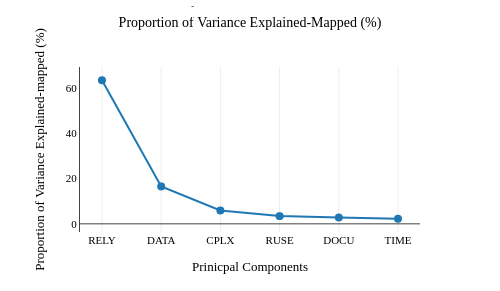
The variance explained by the six (6) PCs presented in Table 4.5 can be visualized in Fig 4.3

Figure 4.3: Proportion of variance explained by RELY, DATA, CPLX, RUSE, DOCU, and TIME

The cumulative amount of information in the original dataset retained by the six (6) Principal Components (PC) arranged in their ascending order is presented in Table 4.6.

Table 4.6: Cumulative explained variance

|  |  |
| --- | --- |
| **PCA Component Name** | **Cumulative Explained Variance** |
| Execution Time (TIME)  Documentation Match to life-cycle needs (DOCU)  Required Reusability (RUSE) | 2.23  5.02  8.43 |
| Process Complexity (CPLX)  Database Size (DATA)  Required Software Reliability (RELY) | 14.31  30.79  94.13 |
|  |  |

The cumulative variance in Table 4.6 is presented graphically in Figure 4.4.

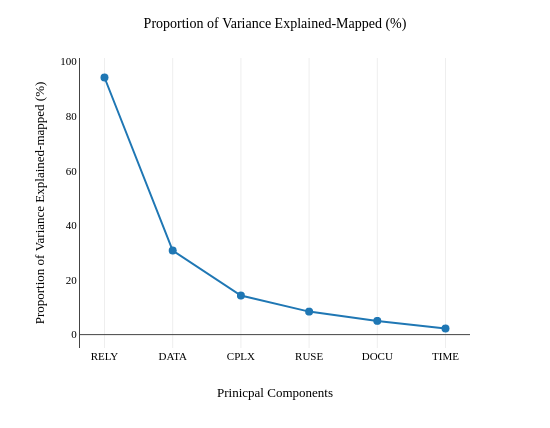


Figure 4.4: Cumulative Explained Variance

A section of the reduced dataset obtained from the PCA algorithm is presented in Table 4.8.

Table 4.8: Reduced Dataset

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **RELY** | **DATA** | **CPLX** | **RUSE** | **DOCU** | **TIME** | **EFFORT** |
| 0.32563 | -0.23076 | 0.09741 | -0.01397 | -0.00464 | 0.12038 | 117.60 |
| 0.32563 | -0.23076 | 0.09741 | -0.01397 | -0.00464 | 0.12038 | 117.60 |
| 0.32563 | -0.23076 | 0.09741 | -0.01397 | -0.00464 | 0.12038 | 31.20 |
| 0.32563 | -0.23076 | 0.09741 | -0.01397 | -0.00464 | 0.12038 | 36.00 |
| 0.32563 | -0.23076 | 0.09741 | -0.01397 | -0.00464 | 0.12038 | 25.20 |
| 0.32563 | -0.23076 | 0.09741 | -0.01397 | -0.00464 | 0.12038 | 8.40 |
| 0.32563 | -0.23076 | 0.09741 | -0.01397 | -0.00464 | 0.12038 | 10.80 |
| 0.32563 | -0.23076 | 0.09741 | -0.01397 | -0.00464 | 0.12038 | 352.80 |
| 0.20292 | 0.48064 | -0.14989 | 0.20560 | 0.07411 | 0.13506 | 72.00 |
| 0.29609 | -0.19916 | -0.01736 | 0.14423 | -0.07429 | -0.15543 | 72.00 |
| 0.30265 | -0.20419 | -0.00412 | 0.10326 | -0.07164 | -0.11354 | 24.00 |
| 0.29609 | -0.19916 | -0.01736 | 0.14423 | -0.07429 | -0.15543 | 360.00 |
| 0.31592 | -0.20464 | 0.00893 | 0.07567 | -0.00225 | -0.13245 | 36.00 |
| 0.32419 | -0.17448 | -0.04938 | 0.02493 | 0.21014 | -0.26430 | 215.00 |
| 0.30265 | -0.20419 | -0.00412 | 0.10326 | -0.07164 | -0.11354 | 48.00 |
| 0.32649 | -0.21192 | 0.04413 | 0.04151 | 0.06658 | -0.11177 | 360.00 |
| 0.30561 | 0.02908 | -0.09809 | 0.40753 | -0.02801 | 0.00539 | 324.00 |
| 0.30503 | -0.20790 | 0.00889 | 0.08767 | -0.06131 | -0.09223 | 60.00 |
| 0.30265 | -0.20419 | -0.00412 | 0.10326 | -0.07164 | -0.11354 | 48.00 |
| 0.32110 | 0.01532 | -0.05861 | 0.31000 | -0.01238 | 0.11049 | 60.00 |
| 0.32563 | -0.23076 | 0.09741 | -0.01397 | -0.00464 | 0.12038 | 60.00 |
| 0.32563 | -0.23076 | 0.09741 | -0.01397 | -0.00464 | 0.12038 | 300.00 |
| 0.32563 | -0.23076 | 0.09741 | -0.01397 | -0.00464 | 0.12038 | 120.00 |
| 0.31719 | -0.16810 | -0.15346 | -0.03586 | 0.13152 | -0.01484 | 90.00 |
| 0.32151 | -0.17720 | -0.00229 | -0.03064 | 0.08536 | -0.07036 | 210.00 |
| 0.30694 | -0.26105 | -0.15454 | -0.01028 | 0.11024 | -0.04643 | 48.00 |
| 0.06905 | 0.18354 | -0.22508 | -0.11729 | -0.23380 | -0.00623 | 70.00 |
| 0.06905 | 0.18354 | -0.22508 | -0.11729 | -0.23380 | -0.00623 | 239.00 |
| 0.06905 | 0.18354 | -0.22508 | -0.11729 | -0.23380 | -0.00623 | 82.00 |
| 0.06905 | 0.18354 | -0.22508 | -0.11729 | -0.23380 | -0.00623 | 62.00 |
| 0.06905 | 0.18354 | -0.22508 | -0.11729 | -0.23380 | -0.00623 | 170.00 |
| 0.06905 | 0.18354 | -0.22508 | -0.11729 | -0.23380 | -0.00623 | 192.00 |
| 0.32563 | -0.23076 | 0.09741 | -0.01397 | -0.00464 | 0.12038 | 18.00 |
| 0.32563 | -0.23076 | 0.09741 | -0.01397 | -0.00464 | 0.12038 | 50.00 |
| 0.32563 | -0.23076 | 0.09741 | -0.01397 | -0.00464 | 0.12038 | 60.00 |
| 0.33215 | -0.18859 | 0.03325 | -0.09834 | 0.10572 | 0.00806 | 42.00 |
| 0.32754 | -0.20603 | 0.01323 | -0.09698 | 0.09652 | -0.01679 | 60.00 |
| 0.16915 | -0.19291 | 0.01045 | -0.06801 | 0.07126 | -0.02532 | 444.00 |
| 0.32754 | -0.20603 | 0.01323 | -0.09698 | 0.09652 | -0.01679 | 42.00 |
| 0.33317 | -0.13052 | -0.00570 | -0.12120 | 0.10860 | -0.01004 | 114.00 |
| 0.32151 | 0.05562 | -0.12764 | -0.11214 | -0.13286 | -0.00168 | 1248.00 |
| -0.91227 | -0.12935 | 0.169262 | -0.00707 | 0.156412 | 0.017672 | 2400.00 |
| 0.149581 | -0.24266 | -0.346 | -0.1356 | 0.114599 | -0.0506 | 1368.00 |
| 0.300033 | -0.24826 | -0.30911 | -0.08691 | 0.027755 | -0.00282 | 973.00 |

### 4.1.2 Adaptive Neuro-Fuzzy (ANFIS) Result

The Adaptive Neuro-Fuzzy Interface System model is trained using the reduced feature vector generated by principal component analysis. The structure of the ANFIS model is presented in Figure 4.5.

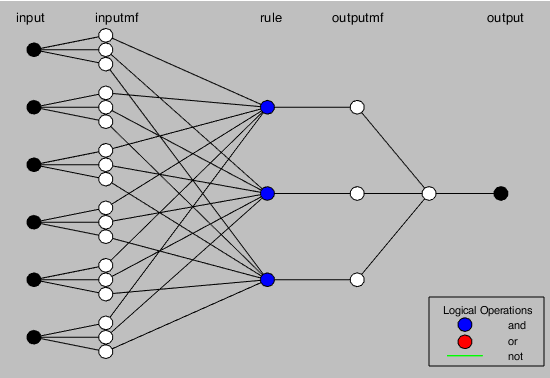


Figure 4.5: The ANFIS structure

The ANFIS model was able to successfully learn from the data and produce an optimal Fuzzy Inference System (FIS). This FIS is composed of the linguistic variables, membership functions, and auto-generated rule-base. The components of the FIS generated by the ANFIS model are presented in Figure 4.6. The membership function plots of RELY, DATA, CPLX, RUSE, DOCU and TIME are shown in Figure 4.7, 4.8, 4.9, 4.10, 4.11 and 4.12 respectively.

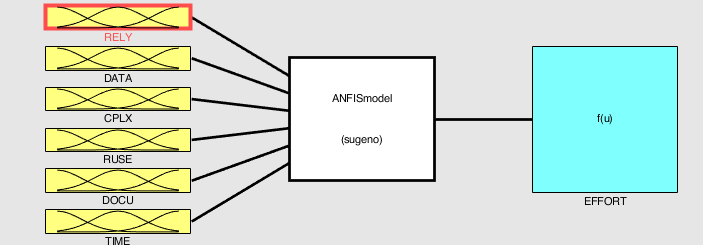


Figure 4.6: Optimal FIS structure

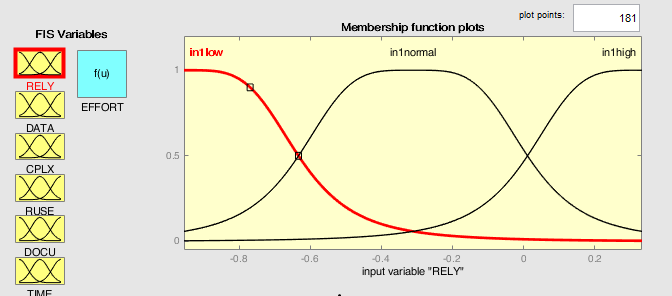


Figure 4.7: PC2-RELY membership function

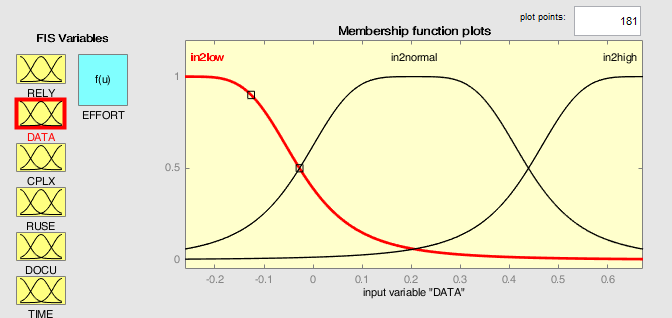


Figure 4.8: PC1-DATA membership function

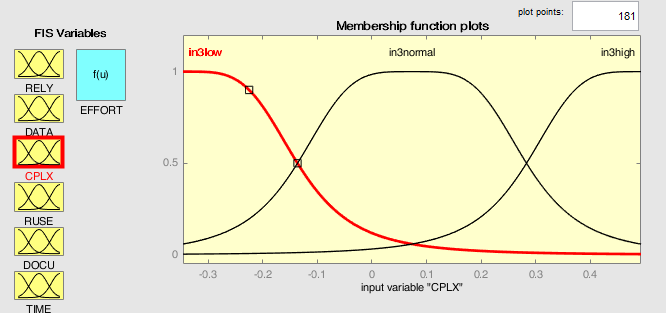


Figure 4.9: PC3-CPLX membership function

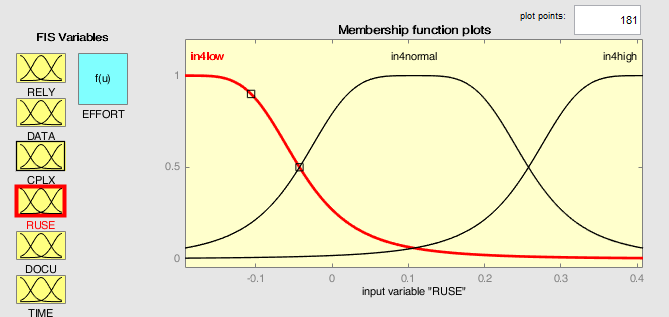


Figure 4.10: PC4-RUSE membership function

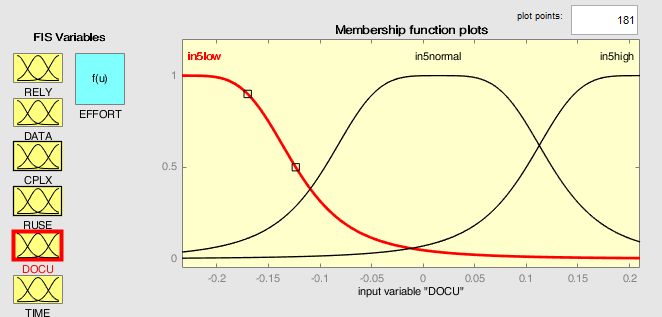


Figure 4.11: PC5-DOCU membership function

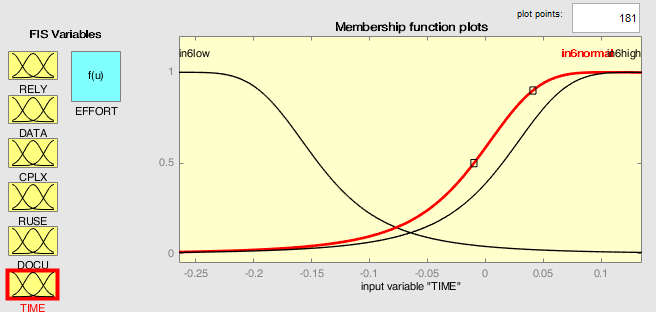


Figure 4.12: PC6-TIME membership function

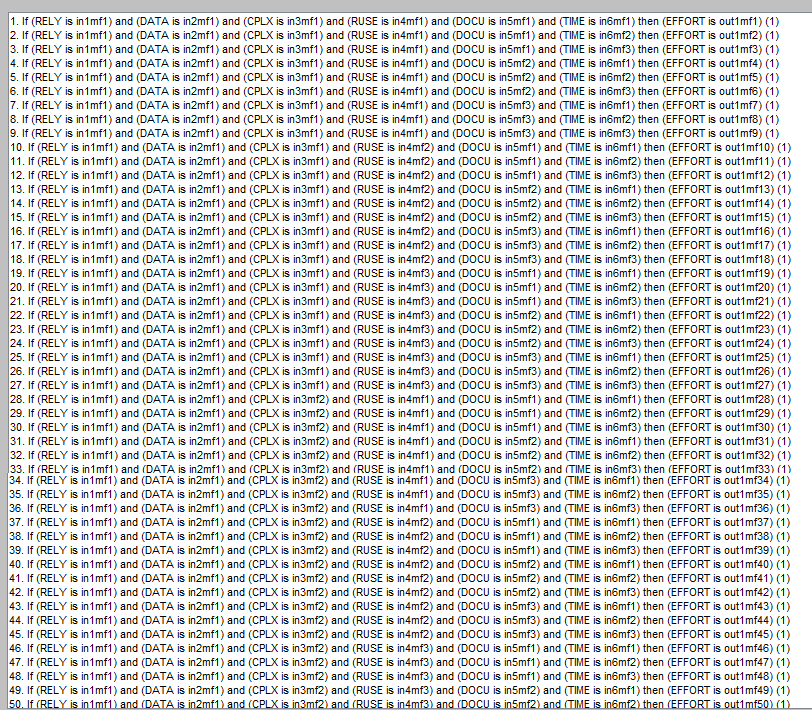


Figure 4.13: auto-generated rule-base

### 4.1.2.1 ANFIS Training

The model was trained with 70% of the entire dataset and tested with 30% of the dataset.

The graph below shows the distribution of training dataset.

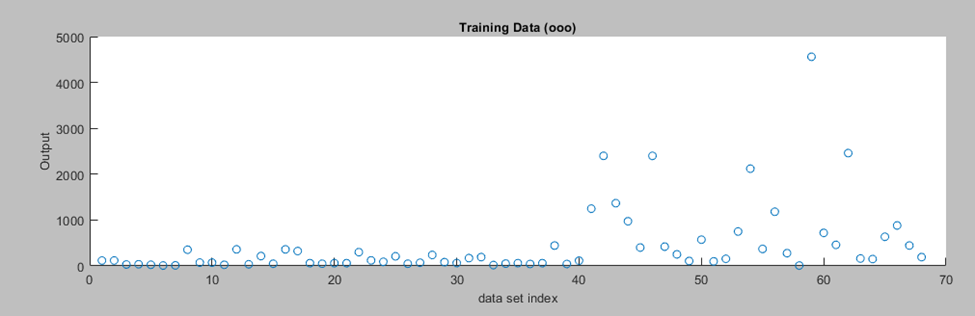
****

Figure 4.14: ANFIS training dataset

The model training was carried out on different number of epochs i.e. 10 epochs, 20 epochs, 25 epochs, 30 epochs, 50 epochs, 70 epochs, and 100 epochs. The ANFIS training result with different epoch values are presented in Figure 4.15 – Figure 4.19.

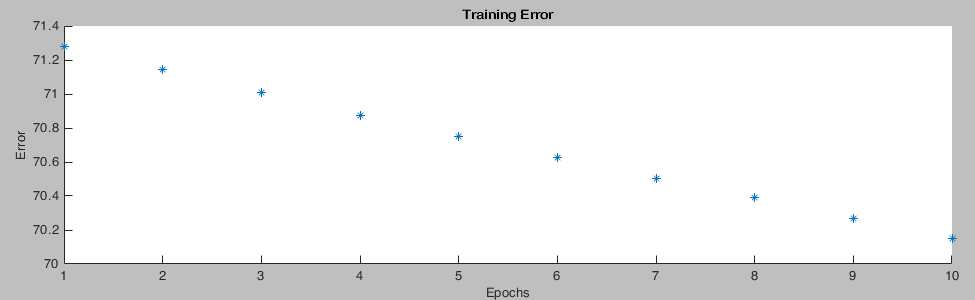


Figure 4.15: Training Results for Epoch 10

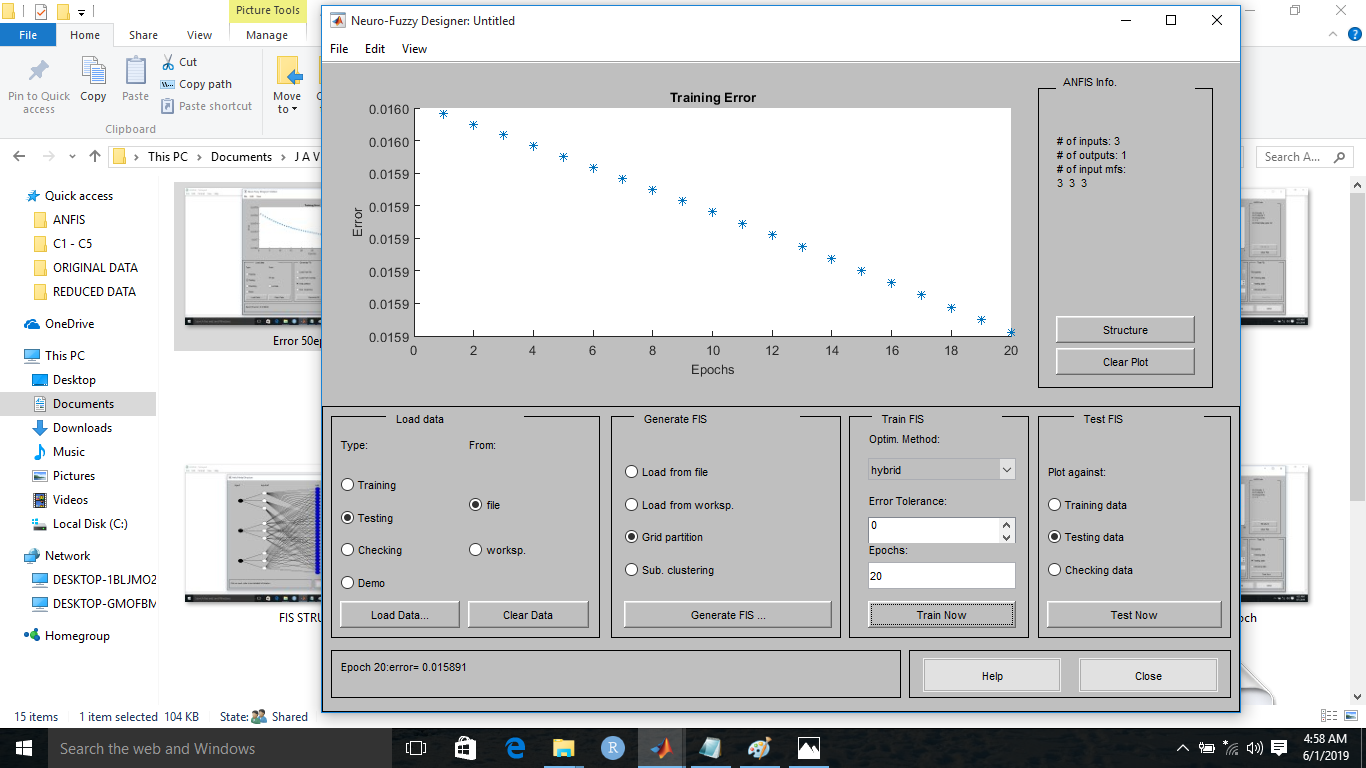


Figure 4.16: Training result with 20 epoch

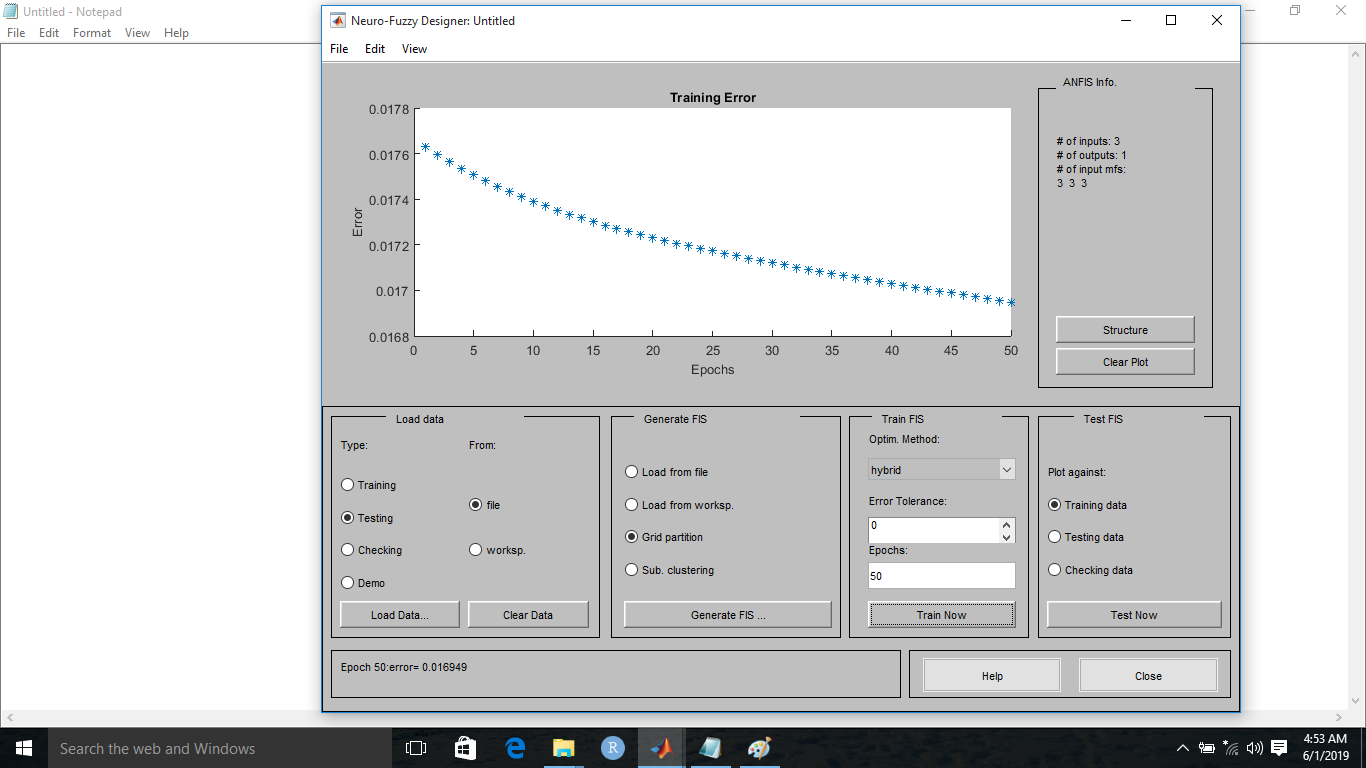


Figure 4.17: Training result with 50 epoch

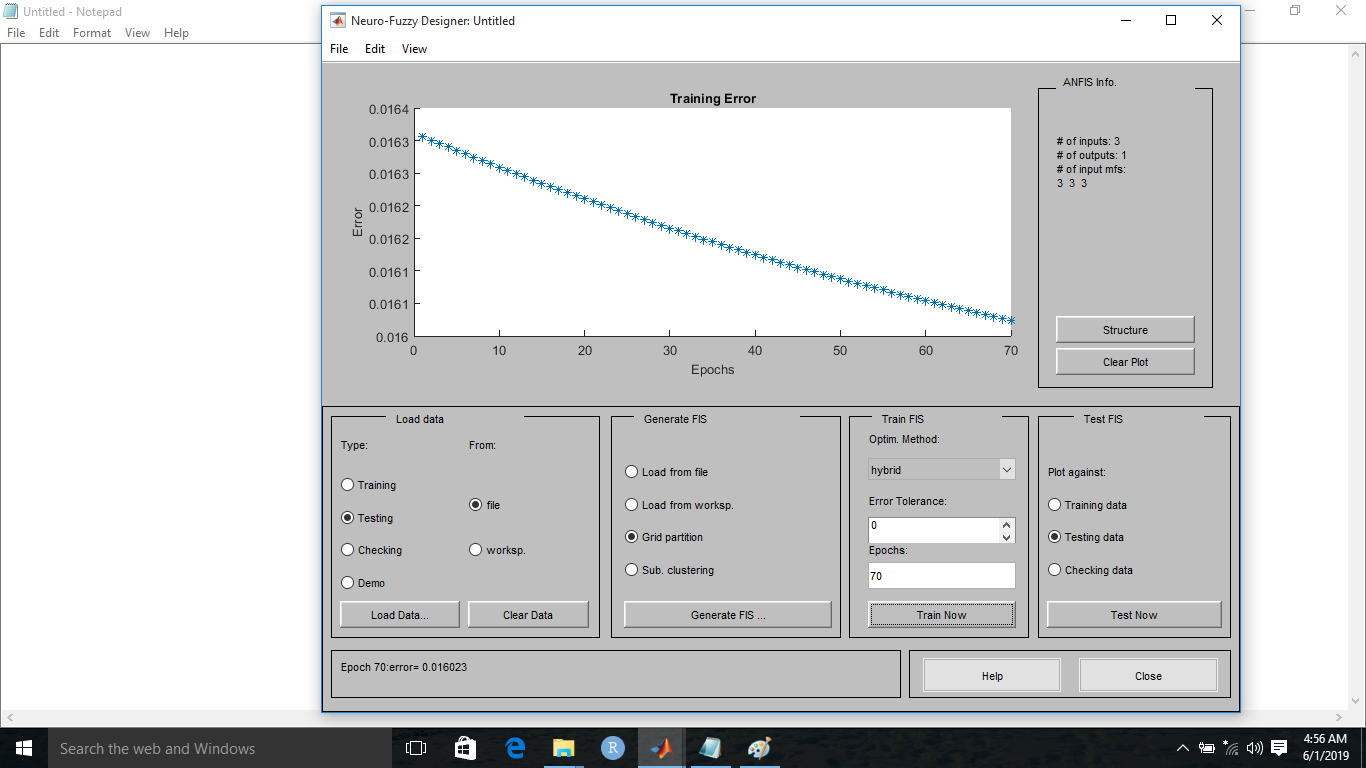


Figure 4.18: Training result with 70 epoch

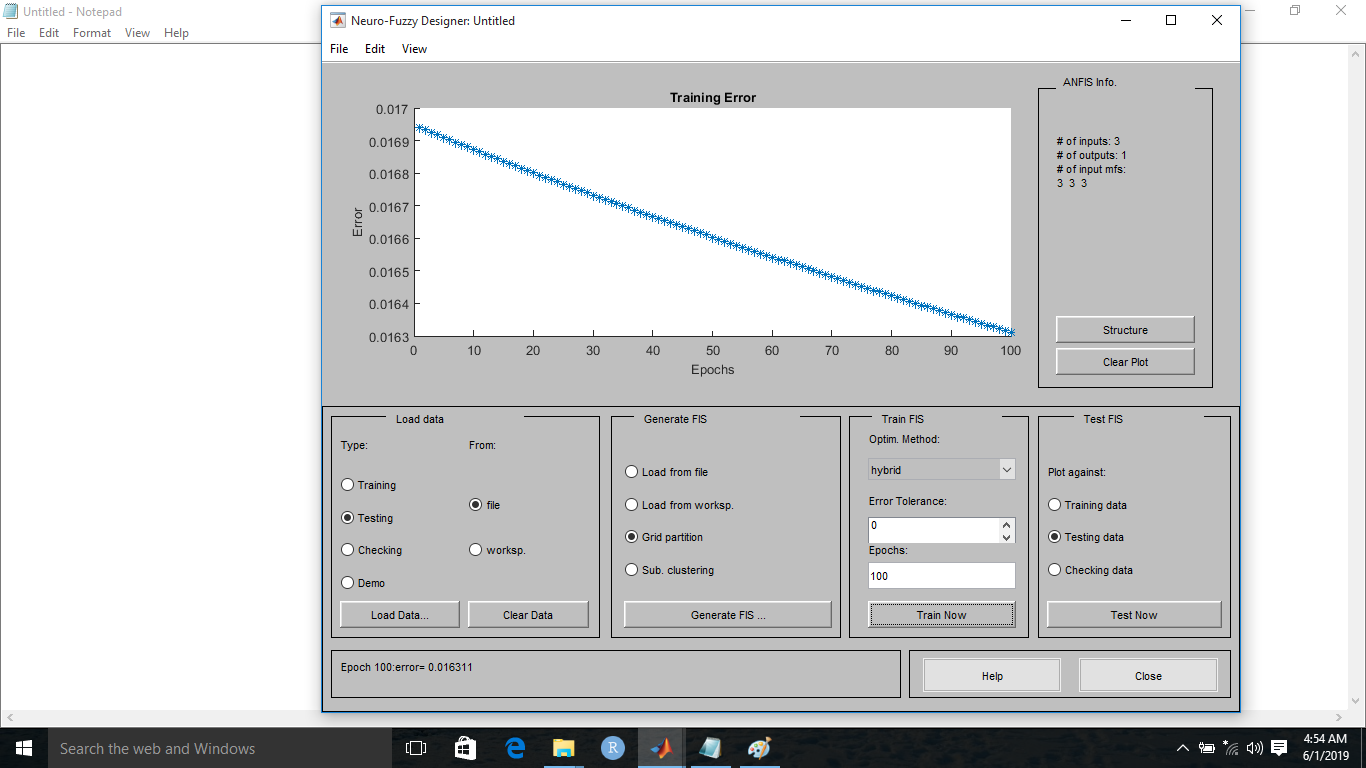


Figure 4.19: Training result with 70 epochs

The effect of epoch on the training error is presented in Table 4.9.

Table 4.9: ANFIS Training Error

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Metric** | **Values** | | | | |
| Epoch | 10 | 20 | 50 | 70 | 100 |
| Training Error | 0.015849 | 0.015514 | 0.016949 | 0.016023 | 0.016311 |

The training error can be visualized using Figure 4.11.

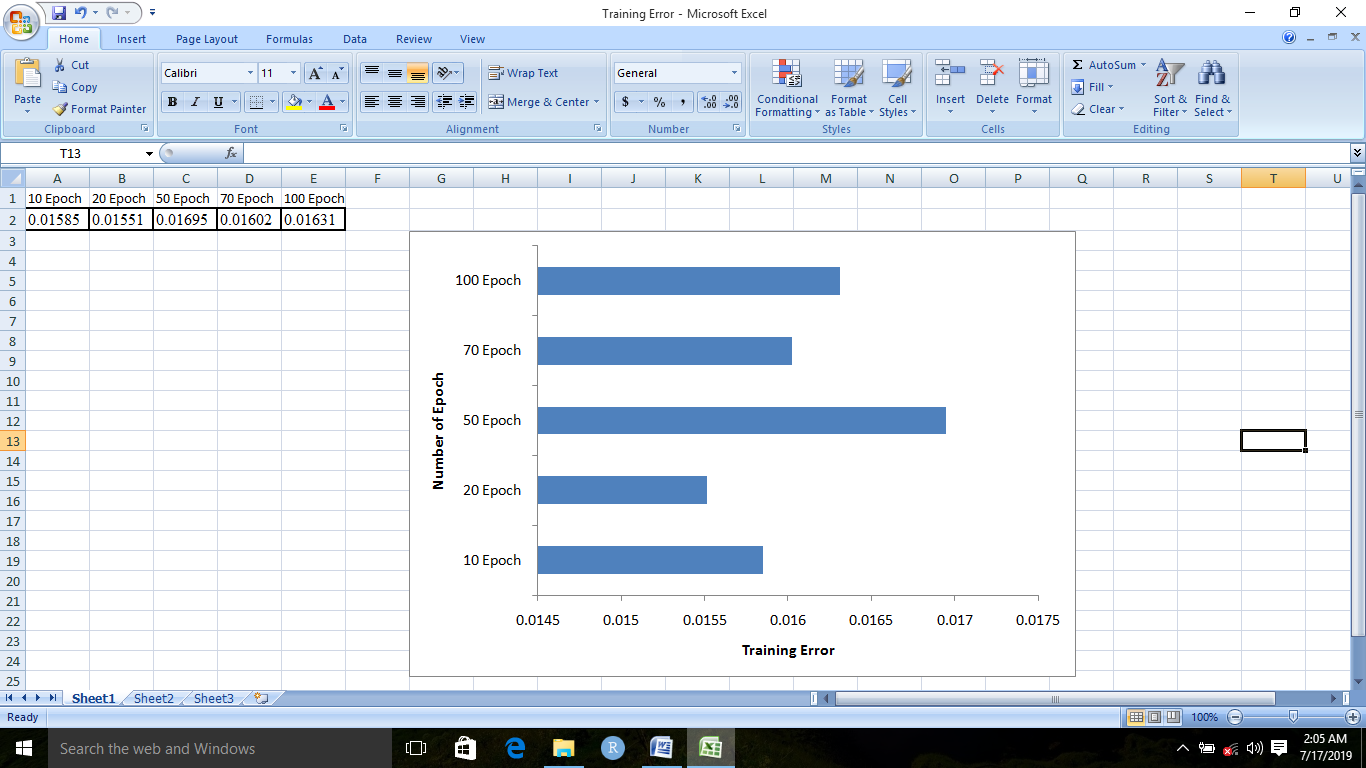


Figure 4.20: ANFIS training error

In order to test how accurate, the ANFIS model is, 30% of the reduced dataset was used for model testing. The testing result based on the different epochs is presented in Figures 4.13, 4.14, 4.15, 4.16 and 4.17.

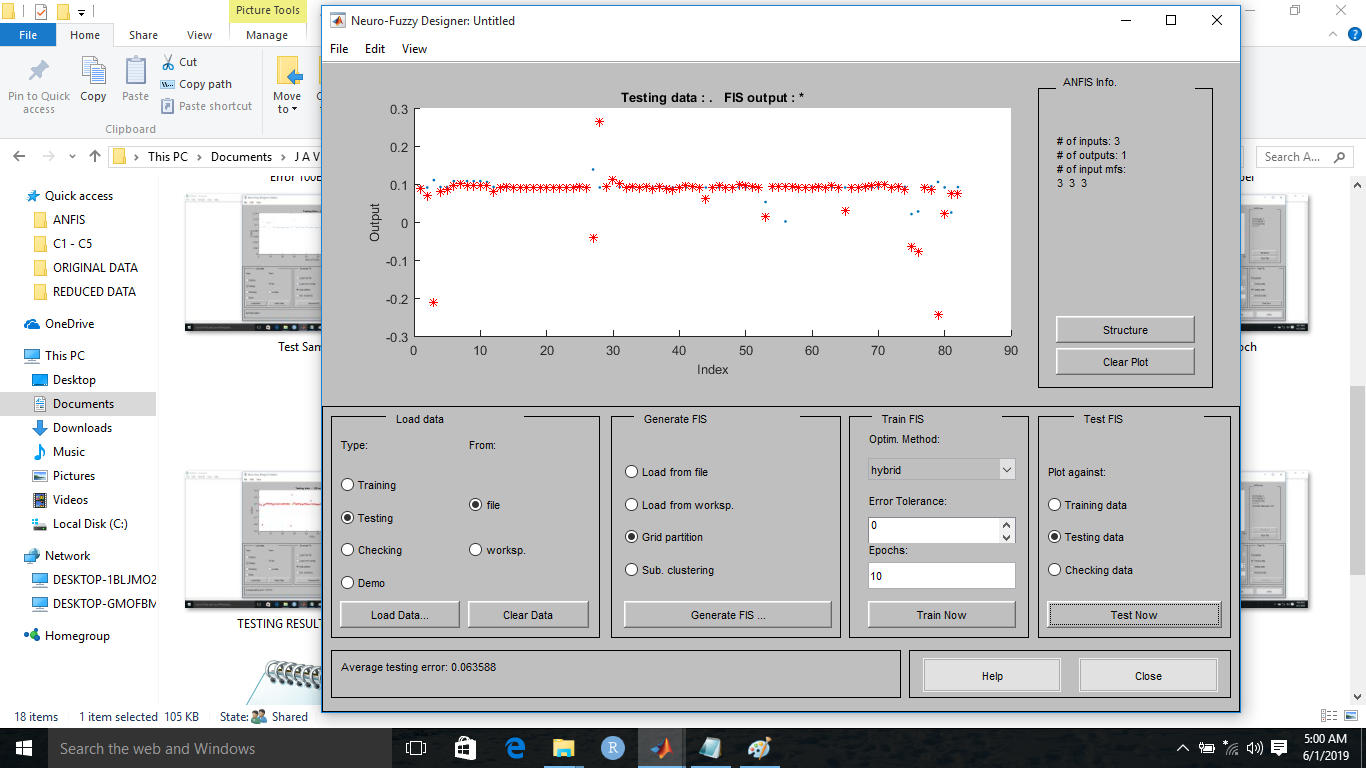


Figure 4.21: Testing result with 10 epochs model

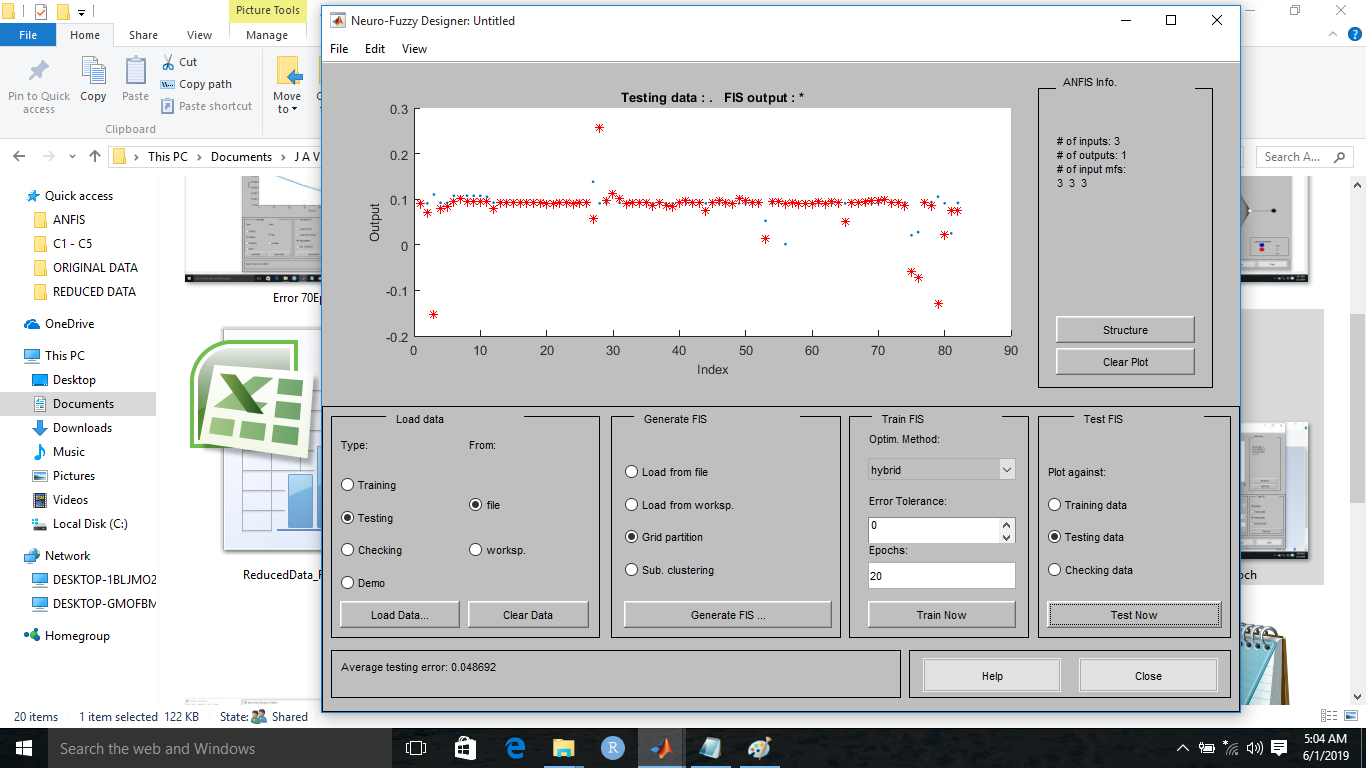


Figure 4.22: Testing result with 20 epochs model

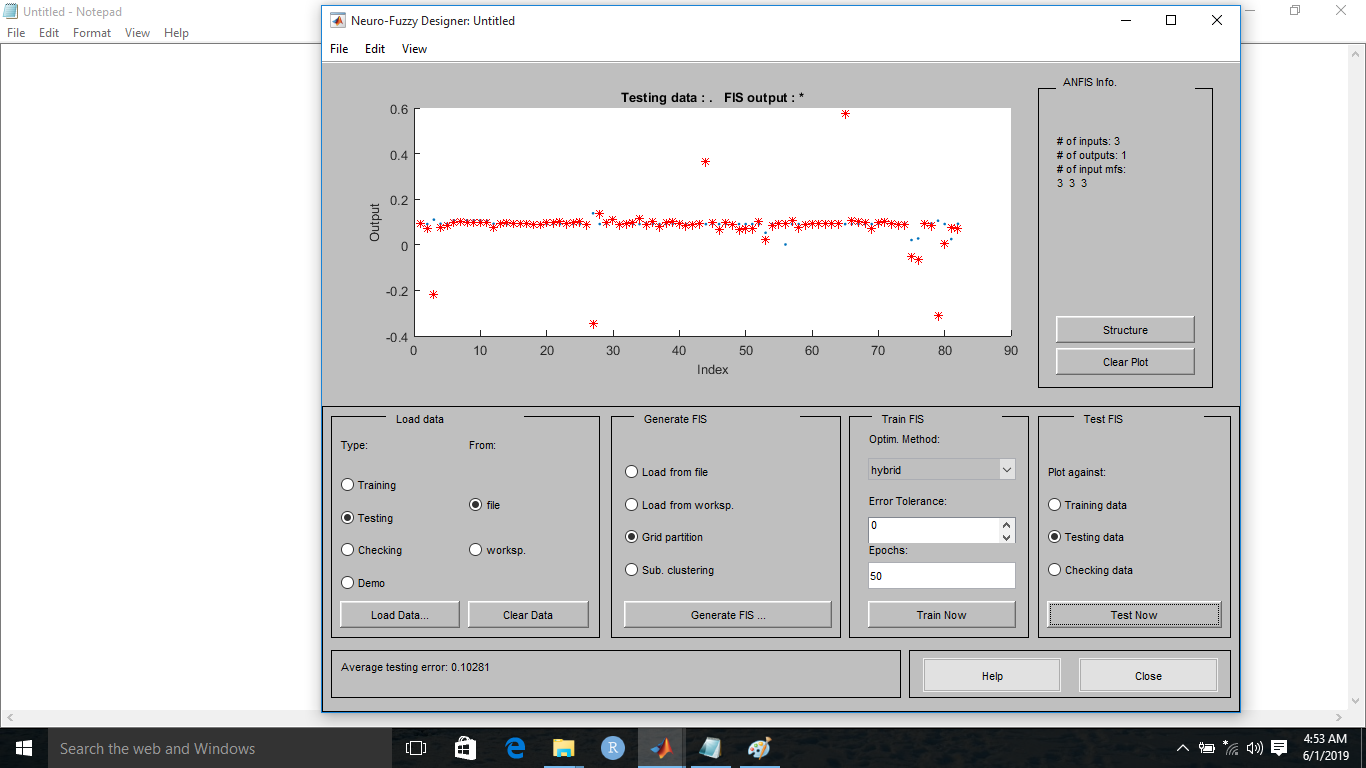


Figure 4.23: Testing result with 50 epochs model

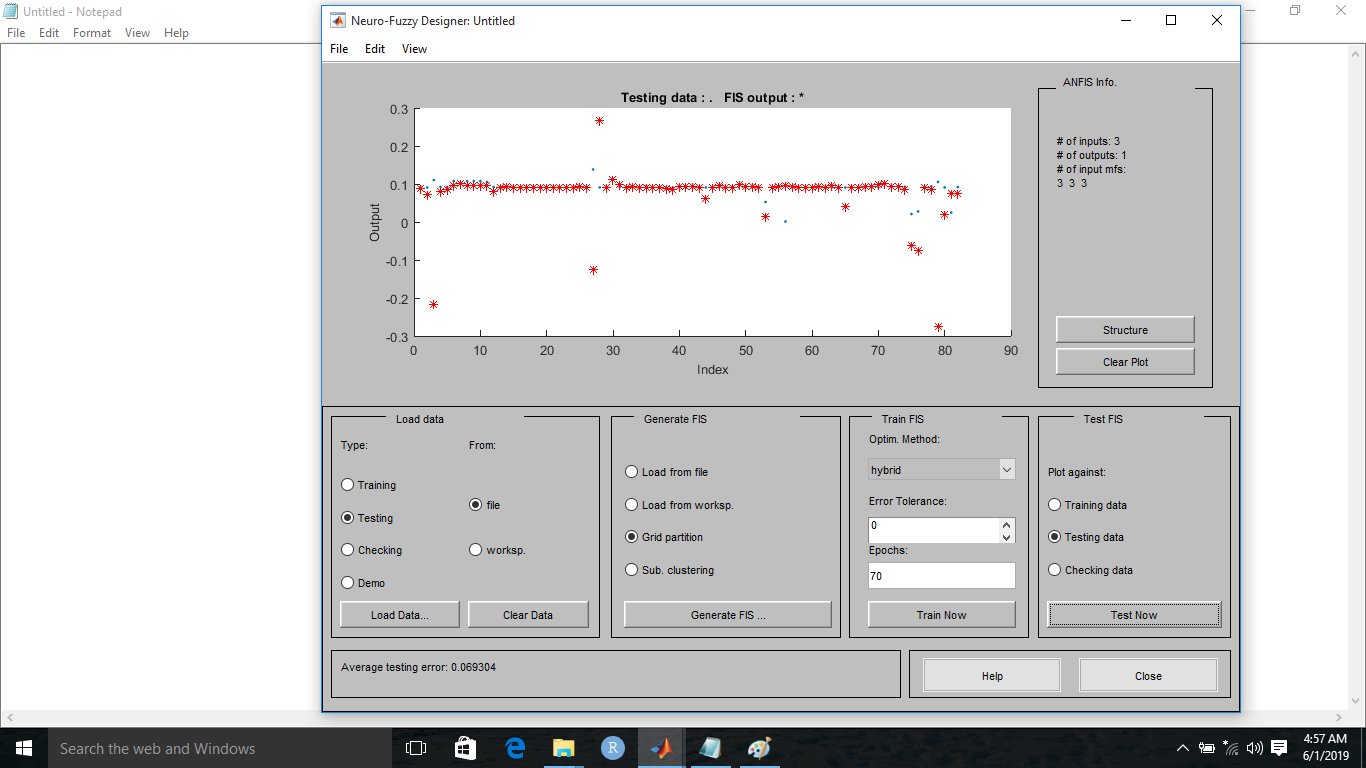


Figure 4.24: Testing result with 70 epochs model

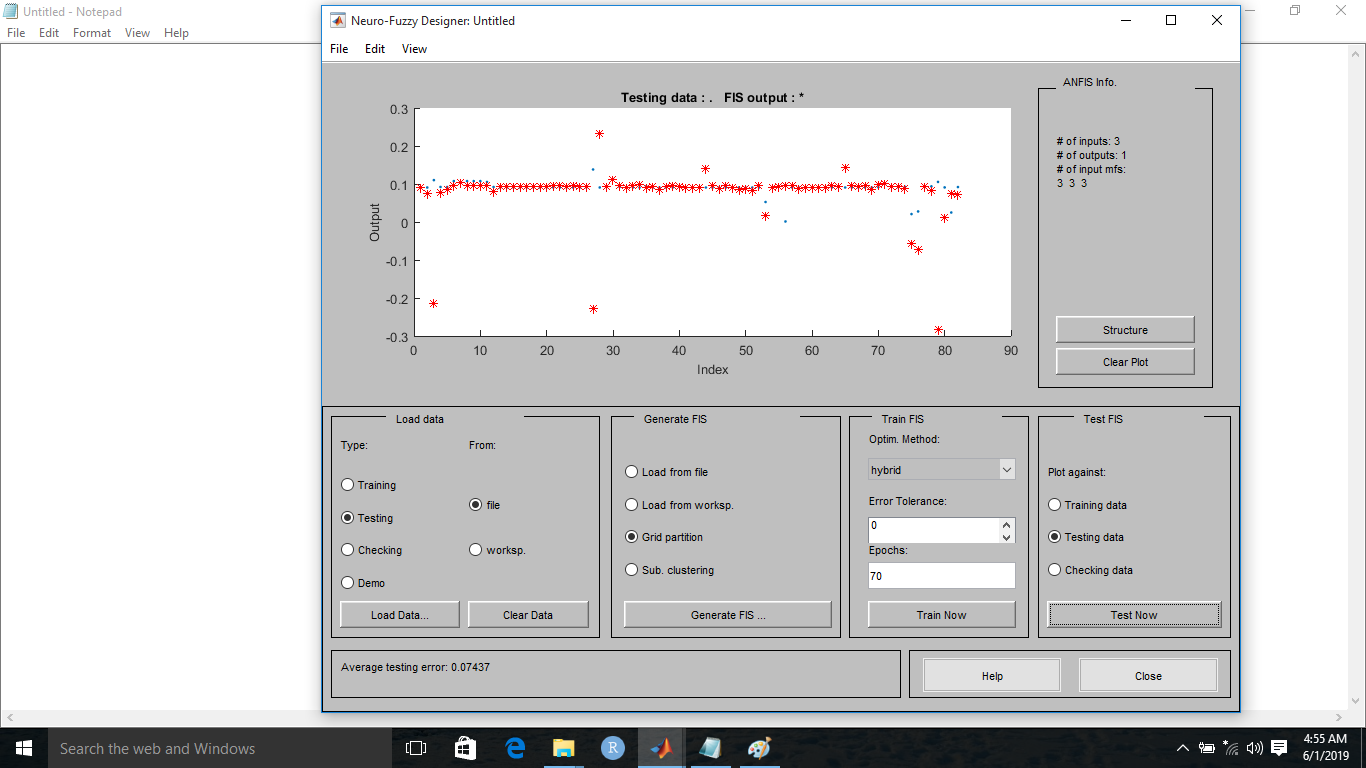


Figure 4.25: Testing result with 70 epochs model

The effect of the epoch on the model testing is depicted in Table 4.10.

Table 4.10: ANFIS Testing Error

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Metric** | **Values** | | | | |
| Epoch | 10 | 20 | 50 | 70 | 100 |
| Testing Error | 0.063588 | 0.048692 | 0.10281 | 0.069304 | 0.07437 |

The graph in Figure 4.18 compares the ANFIS training error and that of the testing error.

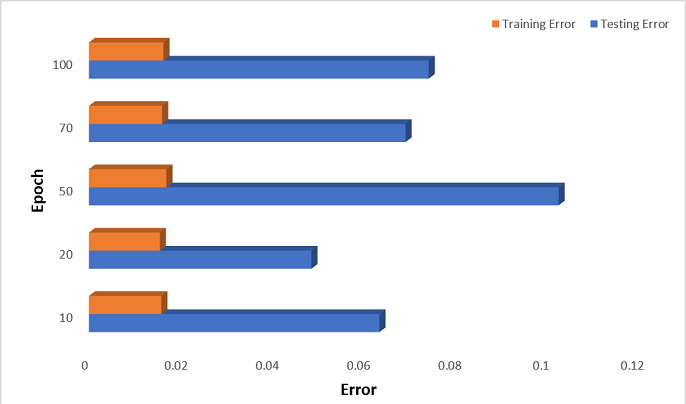


Figure 4.26: ANFIS training and Test error

The resulting relationship among the ANFIS input variables can be visualized using the surface plots presented in Figures 4.26, 4.27, 4.28, 4.29 4.30, 4.31, 4.32, 4.33, 4.34, 4.35,4.36, 4.37, and 4.38

In Figure 4.26, Person-Hours (EFFORT) is fixed and non-increasing if there is no database to implement and if the reliability requirement of the system is minimal. The plot shows that a high software reliability requirement and database size increases the number of person required to develop the software.

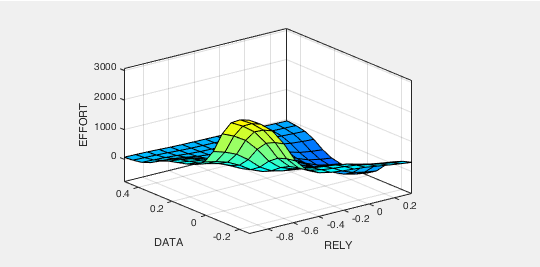


Figure 4.26: Effect of PC1-RELY on PC2-DATA

In Figure 4.27, The number of person-month is high as the process complexity of the program to be developed increases. It can also be depicted that the reliability requirement has more effect in the person-month (EFFORT) because the process complexity gets low as the reliability requirement (RELY) decreases.

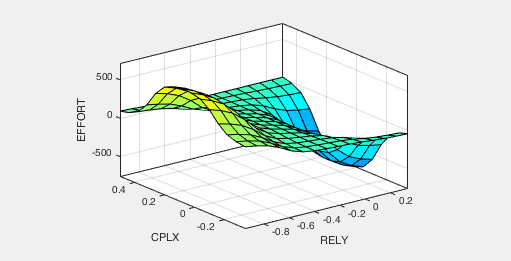


Figure 4.27: Effect of PC1-RELY on PC3-CPLX

In Figure 4.28, when the require reusability feature of a software is high and the required reliability is low, it affects the number of person-hours required to develop the software. Also, an increasing reusability constraint can also increase the reliability requirement and number of persons-hour required to develop such software.

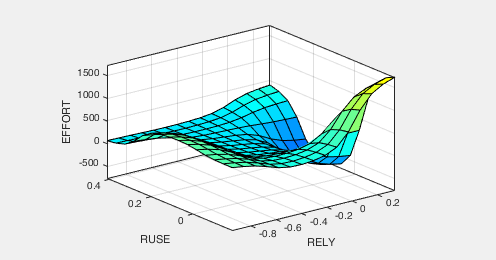


Figure 4.27: Effect of PC1-RELY on PC4-RUSE

In Figure 4.28, The Documentation match to life-cycle (DOCU) can increase to a high extent the number of person-hours because the more the documentation the more functions and processes complexity of the system. However, even though detailed documentations contribute to a good software but when the required reliability constraint is low, the number of person-months reduces. It can also be seen from the plot that the effort remains fixed as the required reliability consttraint is below or equal 0 (not constrained).

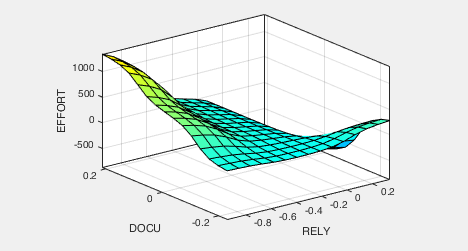


Figure 4.28: Effect of PC1-RELY on PC4-DOCU

In Figure 4.29, it can be seen that the person-month required to develop a software increases as the process complexity of the software and database size (DATA) are required constraints but as the process complexity constraint reduces and database size reduces, the effort required is fixed to 120 and not zero. The 120 person-month minimal effort in the plot accounts for the time and man power required to create table and schema of the database regardless of the complexity of the system.

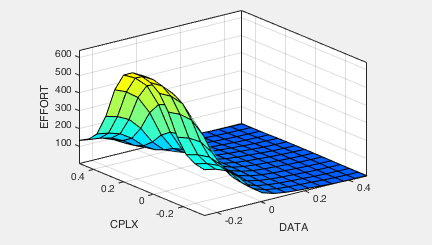


Figure 4.29: Effect of PC2-DATA on PC3-CPLX

In Figure 4.30, as the required reusability reduces and the database size increases, the number of person-month (effort) required to develop a software increases from 100 person-months to 395 person-month but when the database size becomes significant for large projects for example, and the resusability constraint is 0.2 – 0.4, effort increases to 600 person-month.

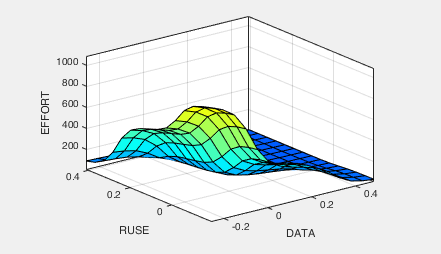


Figure 4.30: Effect of PC2- DATA on PC4-RUSE

In Figure 4.31, the number of person per month required to develop a software is at 600 persons-month when Documentation match to life-cycle (DOCU) is detailed and when the database size is large at 0.4. Even when the database size is large and the Documentation match to life-cycle (DOCU) small (less than zero), the number of person-month required to develop the software is minimal.

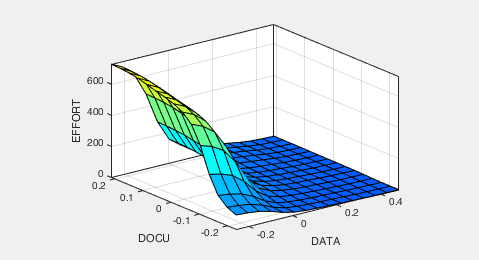


Figure 4.31: Effect of PC2- DATA on PC5-DOCU

In Figure 4.32, it can be seen that the person-month required to develop a software increases as the execution time (TIME) and database size (DATA) of the software are required constraints. However, as the execution time constraint reduces and database size reduces, the effort decreases to a non-zero minimum (120 person-month) and not zero. The 120 person-month minimal effort in the surface accounts for the man hours required to create table and schema of the database.

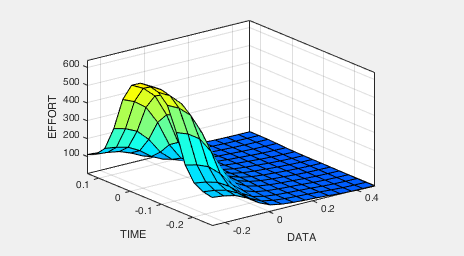


Figure 4.32: Effect of PC2- DATA on PC6-TIME

Figure 4.33 shows the person-hours required to develop a software increases as the process complexity increases. The effect of the required reusability constraint compared to that of the process complexity is infinitesimal. Also, when the complexity constraint reaches the peak value 0.4 and the reusability requirement constraint is negative, the effort remains at minimum value of 30 person-hours.

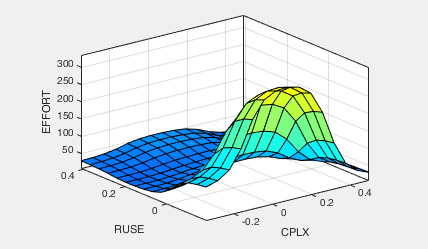


Figure 4.33: Effect of PC3- CPLX on PC4-RUSE

Figure 4.34 shows the effect of process complexity (CPLX) and Documentation match to life-cycle (DOCU) on the Effort. The surface shows that a more detailed documentation and increase the effort and complexity.

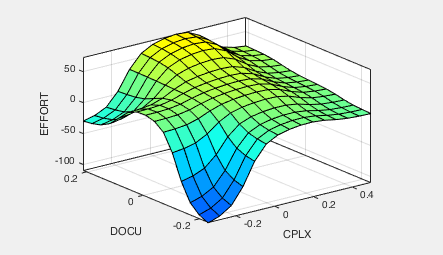


Figure 4.34: Effect of PC3- CPLX on PC5-DOCU

### 4.1.3 Performance Evaluation Result

Based on experiment conducted in 4.1.1 and 4.1.2, it can be concluded that ANFIS model using Back Propagation optimization results better estimate the level RMSE and MSE than the Least Square Estimation ANFIS approach. In addition, the number of epochs used during training are also shown for the two algorithms and their hybrid. It can be concluded that the previous model of ANFIS is more efficient and stable in terms of reduced error during training. The performance evaluation results for Grid Partitioning and Subclustering FIS Generation options are presented in Table 4.11 and Table 4.12 respectively.

Table 4.11: Performance Evaluation Result using Grid Partitioning FIS Generation Option

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Best Epoch** | **Test Error** | **RMSE** | **MSE** |
| Back Propagation | 20 | 0.048692 | 0.54297 | 0.2948112 |
| Hybrid (Back Propagation + Least Square) | 150 | 0.07 | 3.09384171 | 0.0040061 |

Table 4.12: Performance Evaluation Result using Subclustering FIS Generation Option

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Best Epoch** | **Test Error** | **RMSE** | **MSE** |
| Back Propagation | 20 | 0.1026 | 1.2421 | 0.6421 |
| Hybrid (Back Propagation + Least Square) | 150 | 0.117 | 5.4271 | 0.0121 |

## 4.2 Discussion

Table 4.1 displays the effect of number of dimensions on the cumulative explained variance computed by the principal component analysis (PCA) algorithm. From Table 4.1, we can see that the higher the number of dimensions the higher the cumulative explained variance. The total dimension in Table 4.1 is twenty-three (23), which is the total number of dimensions in the original dataset. To select relevant features from the original dataset, a dimension is chosen in order to achieve 94.1% of the original information retained. In the case of this system, the best minimum number of dimensions is Feature 6 with a cumulative explained variance of 2.33%. Figure 4.1 presents in graphical form, the effect of number of dimensions on the cumulative explained variance.

Table 4.2 to Table 4.4 presents the covariance matrix, eigenvector, and the eigenvalues produced by the PCA algorithm. The eigenvalues are used by the PCA algorithm to determine the relevance of each feature in the original dataset. Based on the number of dimensions chosen for the reduced dataset (in this dissertation work, six (6)), PCA chooses principal components with the maximum eigenvalues in descending order.

The principal components together with their eigenvalues are visualized in Figure 4.2. From Figure 4.2, RELY has the highest eigenvalue, followed by DATA, CPLX, REUSE, DOCU, and TIME in their original ordering. Figure 4.3 presented the principal component in descending order sorted by the eigenvalue. The amount of information retained by each of the six (6) chosen principal component is presented in Table 4.5. This table shows that RELY contributes to the highest amount of variance (63.34) retained by the reduce dataset, followed by DATA with the variance of 16.48, followed by CPLX with the variance of 5.87, RUSE with the variance of 3.4, DOCU with the variance of 2.78 and TIME with the variance of 2.24. The proportion of the variance retained by each of this component as presented in table 4.5 is visualized using Figure 4.3. The total amount of variance retained by the reduce dataset is presented in Table 4.6 and visualized in Figure 4.4. Figure 4.4 shows that the total variance retained by the reduced dataset is 94.1 which within the acceptable range.

Table 4.8 presents the reduced dataset produced by the PCA algorithm. The reduced dataset is made up of six (6) principal components (features) and one (1) predictor variable called “Effort”. Figure 4.5 presents the structure of the ANFIS model which is comprised of the crisp input (represented by the input linguistic variables), input membership functions, rules, output membership functions, and the crisp output. The model is made up of six inputs (principal components). Each input variable is made up of three membership functions (low, moderate, and high). The number of rules generated by the ANFIS model is 729 computed as the number of linguistic terms (T) to the power of the number of linguistic variables (V) - TV when using Grid partitioning. The number of rules generated when using subtractive clustering is 3 (three). In subtractive clustering data points are used as the candidates for cluster centres. This means that the computation is proportional to the problem size instead of the problem dimension. (Rama P., et al). However, this work focuses on allowing software managers use quality attributes deduced from requirement document in determining software development effort. Following this, the results are only presented in Table 4.11 and 4.12 and the remain discussion focuses on Grid Partitioning option.

Figure 4.6 shows the fuzzy inference system (FIS) generated by the ANFIS model. The FIS structure is made up of three (3) parts – the input, software development effort predictor (which is made of fuzzifier, inference engine, and defuzzifier), and the crisp output. Figure 4.7 and Figure 4.8 are the graphical membership functions of the first and second principal component respectively.

Figure 4.9 and Figure 4.10 are the membership function of the 3rd and 4th principal components respectively. These membership functions are auto-generated by the ANFIS system. The rule generated by the ANFIS model is presented in Figure 4.13. It comprises of the antecedent and the consequent parts. The rules are used by the fuzzy inference engine to reason on the given input value in order to produce a crisp output.

Figure 4.14 presents the training dataset used in the ANFIS model. The ANFIS model was trained with different values of epoch (i.e. 10epoch, 20epoch, 50epoch, 70epoch, 100epoch) and presented in Figure 4.15, 4.16, 4.17, 4.18, and Figure 4.19. Figure 4.16 shows the ANFIS training result with 10epoch giving a training error of 0.015849, Figure 4.8 shows the ANFIS training with 20epoch giving a training error of 0.015514, Figure 4.18 is the ANFIS training result with 50epoch giving a training error of 0.016949, Figure 4.19 and Figure 4.20 presents the training results with epoch of 70 and 100 giving a training error of 0.016023 and 0.016311 respectively.

The effect of the training epoch on the training error is presented in Table 4.9 and visualized using the bar graph in Figure 4.20. Figure 4.20 shows that the best training performance can be achieved when the ANFIS model is trained with the epoch of 20.

Figure 4.22 shows the ANFIS testing result with epoch 10 giving a testing error of 0.063588 while Figure 4.22, 4.23, 4.24, 4.25 and 4.26 shows the ANFIS testing error on the models trained with 20, 50, 70, and 100 epochs giving a testing error of 0.048692, 0.10281, 0.069304, and 0.07437 respectively. Table 4.10 presents the ANFIS testing error on the 5 different models trained using different epochs. The testing errors in Table 4.9 and Table 4.10 are visualized using the bar chart in Figure 4.16. Figure 4.16 shows that the ANFIS model trained with an epoch value of 20 achieves the overall best performance evaluated using both the training error and the testing error value.

The relationships between each of the input variables (PC1-RELY, PC2-DATA, PC3-CPLX, PC4-RUSE, PC5-DOCU, PC6-TIME) are presented in Figure 4.26, 4.27, 4.28, 4.29, 4.30, 4.31, 4.32, 4.33 and Figure 4.34.

## 4.3 Implication of Research on Software Developers and Researchers

The effect of fuzzy attributes in estimating development effort on software professionals and software managers can lead to effort(person-hours) estimates that are inaccurate, and hence to loss-inducing bids, project management problems, and low client satisfaction. Estimates that show quality attributes of software, as shown in this research, can help in estimating near to exact man hours needed to develop a software during project planning.

**REFERENCES**

Agrawal V. and Shrivastava V. (2015). Performance Evaluation of Software Development Effort Estimation Using Neuro-Fuzzy Model, International Journal of Emerging Research in Management & Technology, 4(7), 2278-9359.

Ahmad M. (2010). High Performance AC Drives: Modelling Analysis and Control, Heidelberg, New York: Springer

Ajitha S., Geetha D., Kumar T., Dr. K. Rajani Kanth (2010), Neural Network Model for Software Size Estimation Using Use Case Point Approach, International Conference on Industrial and Information Systems, IEEE, 372-376.

Albrecht A. and Gaffney J. (1989). Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation, in IEEE Transactions on Software Engineering, 9(6), 639-648.

Ananda K. and Punithavalli M. (2011). Efficient cancer classification using Fast Adaptive Neuro-Fuzzy Inference System (FANFIS) based on Statistical Techniques, (IJACSA) International Journal of Advanced Computer Science and Applications, Special Issue on Artificial Intelligence, 132-137.

Anda B., Benestad H., and Hove S. (2005). A multiple-case study of effort estimation based on use case points, Fourth International Symposium on Empirical Software Engineering, Noosa, Australia, 407–416

Arora S., Mishra N. (2018). Software Cost Estimation Using Artificial Neural Network. In: Pant M., Ray K., Sharma T., Rawat S., Bandyopadhyay A. (eds) Soft Computing: Theories and Applications. Advances in Intelligent Systems and Computing, 584(2), 51-58

Awadallah M., Bayoumi E. and Soliman H. (2009). Adaptive deadbeat controllers for brushless DC drives using PSO and ANFIS techniques, Journal of Electrical Engineering, 60(1), 3-11.

Azzeh M., and Nassif A. (2017). A replicated assessment and comparison of adaptation techniques for analogy-based effort estimation. Empirical Software Engineering 17(1-2), 90-127

Azzeh M., and Elsheikh M. (2012). Learning Best K analogies from Data Distribution for Case-Based Software Effort Estimation. The Seventh International Conference on Software Engineering Advances, 341-347.

Azzeh M., Neagu D., and Cowling P., (2010). Fuzzy grey relational analysis for software effort estimation, Empirical Software Engineering, 15: 60-90

Aberśek B, Flogie A, Aberšek K, and Šverc M. (2017), Cambridge Scholars Publishing, 2017

Baiquni M. and Sarno R. (2015), Optimizing Constructive Cost Model (COCOMO) for increase accuraracy of Estimation Effort,” JURNAL TEKNIK ITS, 4(1), 1–6

Bhatnagar R., Bhattacharjee V., and Ghose M. (2010). Software Development Effort Estimation – Neural Network Vs. Regression Modeling Approach, International Journal of Engineering Science and Technology (IJEST), 2(7), 2950-2956

Britto R., Mendes E., and Börstler J. (2015). An empirical investigation on effort estimation in agile global software development. In Proceedings of the IEEE International Conference on Global Software Engineering - ICGSE’15.

Boehm W., Horowitz E., Madachy R., Reifer D., Clark K., Steece B., Brown A., Chulani S., and C. Abts (2009). Software Cost Estimation with COCOMO II, Prentice Hall.

Burgess C. and Lefley M. (2001). Can genetic programming improve software effort estimation? A comparative evaluation, Information and Software Technology, 43(14), 863–873.

Clarke J., Dolado J., Harman M., Hierons R., Jones B., Lumkin M., Mitchell B., Rees K., and Roper M. (2003). Reformulating software engineering as a search problem, IEE Proceedings Software.

Cherifi H., Zain J., and El-Qawasmeh E. (2011). Digital Information and Communication Technology and Its Applications: International Conference, DICTAP, 806

Divya K., Ashish T. and Prof. Misra A. (2012). Software Development Effort and Cost Estimation: Neuro-Fuzzy Model, IOSR Journal of Computer Engineering (IOSRJCE). 2(4).

Divya K. and Misra A. (2013). An approach for software effort estimation using fuzzy numbers and genetic algorithm to deal with uncertainty

Gharehchopogh F., Maleki I., and Khaze S. (2014). A novel particle swarm optimization approach for software effort estimation. International Journal of Academic Research Part A, 6(2), 69-76.

Ghose M., Bhatnagar R. and Bhattacharjee V. (2011). Comparing Some Neural Network Models for Software Development Effort Prediction, IEEE conference.

Grimstad, S. and Jørgensen M. (2007). A Framework for the Analysis of Software Cost. In preparation.Journal of Systems and Software, 80(11), 1770-1777

Fenton N. (1994). Software Measurement: A necessary Scientific Basis, IEEE Transactions on Software Engineering, 20, 199-206.

Ferrucci F., Gravino C., Oliveto R., Sarro F. (2010). Genetic Programming for Effort Estimation: An Analysis of the Impact of Different Fitness Functions, 2nd International Symposium on Search Based Software Engineering

Hanchate D. B. and Bichkar R. S. (2015), Mathematical Modeling of Lewis and COCOMO-II software cost estimation using regulatory focus theory, Applied Discrete Mathematics and Heuristic Algorithms

International Scientific Journal 1(2), 5–25.

Harman M. and Jones B. (2001). Search based software engineering, Information and Software Technology, 43(14), 833–839.

Hari C. and Reddy P. (2011), A Fine Parameter Tuning for COCOMO 81 Software Effort Estimation using Particle Swarm Optimization, Journal of Software Engineering, 5, 38-48.

Huang X., Ho D., Ren J., and Capretz F. (2007), Improving the COCOMO model using a neuro-fuzzy approach, Applied Soft Computing 7, 29-40

Idri A., Abran A., and Khoshgoftaa T. (2001), Fuzzy analogy: a new approach for software effort estimation, 11th International Workshop in Software Measurements.

Idri A., Abdelali Z. and Azeddine Z. (2010). Design of Radial Basis Function Neural Networks for Software Effort Estimation, IJCSI International Journal of Computer Science Issues, 7(4).

Jang J. (1993). ANFIS: adaptive-network-based fuzzy inference system. Systems, Man and Cybernetics, IEEE Transactions on, 23(3), 665-685.

Jorgensen M. (2004). Realism in assessment of effort estimation uncertainty: it matter how you ask. IEEE Transaction on software Engineering, 30.

Karner G. (1993). Resource Estimation for Objectory Projects, Objective Systems SF AB, 1-9

Kaushik A., Soni A. and Rachna S. (2013). A Simple Neural Network Approach to Software Cost Estimation Global Journal of Computer Science and Technology Neural and Artificial Intelligence, 13(1).

Kirsopp C., Mendes E., Premraj R., and Shepperd M., (2003). An empirical analysis of linear adaptation techniques for case-based prediction. International Conference on ABE, 231–245

Kocaguneli E., Menzies T., Bener A. and Keung J. (2011). Exploiting the Essential Assumptions of Analogybased Effort Estimation, IEEE transaction on Software Engineering. 0098-5589.

Kumar A. (2014), Fuzzy Expert Systems for Disease Diagnosis, Coimbatore, Tamil Nadu: IGI Global

Kusumoto S., Matukawa F., Inoue K, Hanabusa S. and Maegawa Y. (2004), Estimating effort by Use Case Points: method, tool and case study, Proceedings of the 10th International Symposium on Software Metrics.

Kumar A., Sinhal A., and Verma B. (2013), A Novel Technique of Optimization for Software Metric Using PSO,” International Journal of Soft Computing And Software Engineering, 3(10), 2251– 7545.

Lefley M. and Shepperd M. (2003). Using genetic programming to improve software effort estimation based on general data sets, in Proceedings of Genetic and Evolutionary Computation Conference, 2477–2487.

Li F., Xie M., and Goh N. (2009). A study of the non-linear adjustment for analogy based software cost estimation. Empirical Software Engineering 14, 603–643.

Liu P., Leng W. and Fang W. (2013), Training ANFIS model with an improved quantum-behaved particle swarm optimization algorithm, Mathematical Problems in Engineering.

Lopez-Martin C., Yanez-Marquez C., Gutierrez-Tornes A. (2008). Predictive accuracy comparison of fuzzy models for software development effort of small programs, The journal of systems and Software, 81(6), 949-960.

Mahdheswaran M. and Sivakumar D. (2014). Enhancement of Prediction Accuracy in COCOMO Model for Software Project Using Neural Network, Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT), 1-5.

Mahesh P., Manimala P., Podtar P. (2014). Literature Survey on Algorithmic methods for Software Development Cost Estimation IJCTA, 5(1), 183-188.

Mewada K., Sinhal A., and Verma B. (2013). Adaptive Neuro-Fuzzy Inference System (ANFIS) Based Software Evaluation, IJCSI International Journal of Computer Science, 10(5), 244-250.

Moløkken-Østvold K., and Jørgensen, M. (2004). Group Processes in Software Effort Estimation. Empirical Software Engineering, 9(4), 315–334.

Narendra S., Aman B. and Litoriya R. (2012). A comparison of software cost estimation methods: A Survey, The International Journal of Computer Science and Applications (TIJCSA), 1(3).

Nassif A., Capretz L, and Ho D. (2012). Estimating Software Effort Using an ANN Model Based on Use Case Points, International Conference on Machine Learning and Applications, IEEE, 42-46.

Ochodek O., Nawrocki J., and Kwarciak K. (2011). Simplifying effort estimation based on Use Case Points. Information and Software Technology, 53(3), 200-213.

Parkash J. and Karambir (2014), COCOMO II Model Parameter Optimization using PSO and Effort Estimation, Int. Journal of Information Technology & Mechanical Engineering - IJITME, 1(4), 1-11.

Putnam L., and Myers W. (1991). Measures for Excellence: Reliable Software on Time, Within Budget, Prentice Hall, 234.

Popli R., and Chauhan, N. (2014). Cost and effort estimation in agile software development. Optimization, Reliabilty, and Information Technology (ICROIT). 2014 International Conference on, 57–61.

Rijwani P. and Sonal J. (2016). Enhanced Software Effort Estimation using Multi-Layered Feed Forward Artificial Neural Network Technique, Twelfth International Multi-Conference on Information Processing, 89(1), 307-312.

Sarno R., Sidabutar J., and Sarwosri (2015), Comparison of different Neural Network architectures for software cost estimation, International Conference on Computer, Control, Informatics and its Applications (IC3INA). 68-73

Rijwani P., Sonal J., and Dharmesh S. (2014), Software Effort Estimation: A Comparison Based Perspective, International Journal of Application or Innovation in engineering and management (IJAIEM), 3(12).

Rezvani S. (2015). A New Similarity Measure of Generalized Fuzzy numbers Based on Left and Right Apex Angles (II). Palestine Journal of Mathematics. 4(1), 446–454.

Saade J. and Diab D. (2004). Defuzzification Methods and New Techniques for Fuzzy Controllers, Iranian Journal of Electrical and Computer Engineering, 3(2),161-174

Tailor O., Saini T. and Rijwani P. (2014). Comparative Analysis of Software Cost and Effort Estimation Methods: A Review, International Journal of Computer Science and Mobile Computing, IJCSMC, 3(4), 1364 – 1374.

Tomaszewska K. (2014). The Application of Horizontal Membership Functions to Fuzzy Arithmetic Operations, Journal of Theoretical and Applied Computer Science, 8(2), 3-10.

**REFERENCES**

Agrawal V. and Shrivastava V. (2015). Performance Evaluation of Software Development Effort Estimation Using Neuro-Fuzzy Model, International Journal of Emerging Research in Management & Technology, 4(7), 2278-9359.

Ahmad M. (2010). High Performance AC Drives: Modelling Analysis and Control, Heidelberg, New York: Springer

Ajitha S., Geetha D., Kumar T., Dr. K. Rajani Kanth (2010), Neural Network Model for Software Size Estimation Using Use Case Point Approach, International Conference on Industrial and Information Systems, IEEE, 372-376.

Albrecht A. and Gaffney J. (1989). Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation, in IEEE Transactions on Software Engineering, 9(6), 639-648.

Ananda K. and Punithavalli M. (2011). Efficient cancer classification using Fast Adaptive Neuro-Fuzzy Inference System (FANFIS) based on Statistical Techniques, (IJACSA) International Journal of Advanced Computer Science and Applications, Special Issue on Artificial Intelligence, 132-137.

Anda B., Benestad H., and Hove S. (2005). A multiple-case study of effort estimation based on use case points, Fourth International Symposium on Empirical Software Engineering, Noosa, Australia, 407–416

Arora S., Mishra N. (2018). Software Cost Estimation Using Artificial Neural Network. In: Pant M., Ray K., Sharma T., Rawat S., Bandyopadhyay A. (eds) Soft Computing: Theories and Applications. Advances in Intelligent Systems and Computing, 584(2), 51-58

Awadallah M., Bayoumi E. and Soliman H. (2009). Adaptive deadbeat controllers for brushless DC drives using PSO and ANFIS techniques, Journal of Electrical Engineering, 60(1), 3-11.

Azzeh M., and Nassif A. (2017). A replicated assessment and comparison of adaptation techniques for analogy-based effort estimation. Empirical Software Engineering 17(1-2), 90-127

Azzeh M., and Elsheikh M. (2012). Learning Best K analogies from Data Distribution for Case-Based Software Effort Estimation. The Seventh International Conference on Software Engineering Advances, 341-347.

Azzeh M., Neagu D., and Cowling P., (2010). Fuzzy grey relational analysis for software effort estimation, Empirical Software Engineering, 15: 60-90

Aberśek B, Flogie A, Aberšek K, and Šverc M. (2017), Cambridge Scholars Publishing, 2017

Baiquni M. and Sarno R. (2015), Optimizing Constructive Cost Model (COCOMO) for increase accuraracy of Estimation Effort,” JURNAL TEKNIK ITS, 4(1), 1–6

Bhatnagar R., Bhattacharjee V., and Ghose M. (2010). Software Development Effort Estimation – Neural Network Vs. Regression Modeling Approach, International Journal of Engineering Science and Technology (IJEST), 2(7), 2950-2956

Britto R., Mendes E., and Börstler J. (2015). An empirical investigation on effort estimation in agile global software development. In Proceedings of the IEEE International Conference on Global Software Engineering - ICGSE’15.

Boehm W., Horowitz E., Madachy R., Reifer D., Clark K., Steece B., Brown A., Chulani S., and C. Abts (2009). Software Cost Estimation with COCOMO II, Prentice Hall.

Burgess C. and Lefley M. (2001). Can genetic programming improve software effort estimation? A comparative evaluation, Information and Software Technology, 43(14), 863–873.

Clarke J., Dolado J., Harman M., Hierons R., Jones B., Lumkin M., Mitchell B., Rees K., and Roper M. (2003). Reformulating software engineering as a search problem, IEE Proceedings Software.

Cherifi H., Zain J., and El-Qawasmeh E. (2011). Digital Information and Communication Technology and Its Applications: International Conference, DICTAP, 806

Divya K., Ashish T. and Prof. Misra A. (2012). Software Development Effort and Cost Estimation: Neuro-Fuzzy Model, IOSR Journal of Computer Engineering (IOSRJCE). 2(4).

Divya K. and Misra A. (2013). An approach for software effort estimation using fuzzy numbers and genetic algorithm to deal with uncertainty

Gharehchopogh F., Maleki I., and Khaze S. (2014). A novel particle swarm optimization approach for software effort estimation. International Journal of Academic Research Part A, 6(2), 69-76.

Ghose M., Bhatnagar R. and Bhattacharjee V. (2011). Comparing Some Neural Network Models for Software Development Effort Prediction, IEEE conference.

Grimstad, S. and Jørgensen M. (2007). A Framework for the Analysis of Software Cost. In preparation.Journal of Systems and Software, 80(11), 1770-1777

Fenton N. (1994). Software Measurement: A necessary Scientific Basis, IEEE Transactions on Software Engineering, 20, 199-206.

Ferrucci F., Gravino C., Oliveto R., Sarro F. (2010). Genetic Programming for Effort Estimation: An Analysis of the Impact of Different Fitness Functions, 2nd International Symposium on Search Based Software Engineering

Hanchate D. B. and Bichkar R. S. (2015), Mathematical Modeling of Lewis and COCOMO-II software cost estimation using regulatory focus theory, Applied Discrete Mathematics and Heuristic Algorithms

International Scientific Journal 1(2), 5–25.

Harman M. and Jones B. (2001). Search based software engineering, Information and Software Technology, 43(14), 833–839.

Hari C. and Reddy P. (2011), A Fine Parameter Tuning for COCOMO 81 Software Effort Estimation using Particle Swarm Optimization, Journal of Software Engineering, 5, 38-48.

Huang X., Ho D., Ren J., and Capretz F. (2007), Improving the COCOMO model using a neuro-fuzzy approach, Applied Soft Computing 7, 29-40

Idri A., Abran A., and Khoshgoftaa T. (2001), Fuzzy analogy: a new approach for software effort estimation, 11th International Workshop in Software Measurements.

Idri A., Abdelali Z. and Azeddine Z. (2010). Design of Radial Basis Function Neural Networks for Software Effort Estimation, IJCSI International Journal of Computer Science Issues, 7(4).

Jang J. (1993). ANFIS: adaptive-network-based fuzzy inference system. Systems, Man and Cybernetics, IEEE Transactions on, 23(3), 665-685.

Jorgensen M. (2004). Realism in assessment of effort estimation uncertainty: it matter how you ask. IEEE Transaction on software Engineering, 30.

Karner G. (1993). Resource Estimation for Objectory Projects, Objective Systems SF AB, 1-9

Kaushik A., Soni A. and Rachna S. (2013). A Simple Neural Network Approach to Software Cost Estimation Global Journal of Computer Science and Technology Neural and Artificial Intelligence, 13(1).

Kirsopp C., Mendes E., Premraj R., and Shepperd M., (2003). An empirical analysis of linear adaptation techniques for case-based prediction. International Conference on ABE, 231–245

Kocaguneli E., Menzies T., Bener A. and Keung J. (2011). Exploiting the Essential Assumptions of Analogybased Effort Estimation, IEEE transaction on Software Engineering. 0098-5589.

Kumar A. (2014), Fuzzy Expert Systems for Disease Diagnosis, Coimbatore, Tamil Nadu: IGI Global

Kusumoto S., Matukawa F., Inoue K, Hanabusa S. and Maegawa Y. (2004), Estimating effort by Use Case Points: method, tool and case study, Proceedings of the 10th International Symposium on Software Metrics.

Kumar A., Sinhal A., and Verma B. (2013), A Novel Technique of Optimization for Software Metric Using PSO,” International Journal of Soft Computing And Software Engineering, 3(10), 2251– 7545.

Lefley M. and Shepperd M. (2003). Using genetic programming to improve software effort estimation based on general data sets, in Proceedings of Genetic and Evolutionary Computation Conference, 2477–2487.

Li F., Xie M., and Goh N. (2009). A study of the non-linear adjustment for analogy based software cost estimation. Empirical Software Engineering 14, 603–643.

Liu P., Leng W. and Fang W. (2013), Training ANFIS model with an improved quantum-behaved particle swarm optimization algorithm, Mathematical Problems in Engineering.

Lopez-Martin C., Yanez-Marquez C., Gutierrez-Tornes A. (2008). Predictive accuracy comparison of fuzzy models for software development effort of small programs, The journal of systems and Software, 81(6), 949-960.

Mahdheswaran M. and Sivakumar D. (2014). Enhancement of Prediction Accuracy in COCOMO Model for Software Project Using Neural Network, Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT), 1-5.

Mahesh P., Manimala P., Podtar P. (2014). Literature Survey on Algorithmic methods for Software Development Cost Estimation IJCTA, 5(1), 183-188.

Mewada K., Sinhal A., and Verma B. (2013). Adaptive Neuro-Fuzzy Inference System (ANFIS) Based Software Evaluation, IJCSI International Journal of Computer Science, 10(5), 244-250.

Moløkken-Østvold K., and Jørgensen, M. (2004). Group Processes in Software Effort Estimation. Empirical Software Engineering, 9(4), 315–334.

Narendra S., Aman B. and Litoriya R. (2012). A comparison of software cost estimation methods: A Survey, The International Journal of Computer Science and Applications (TIJCSA), 1(3).

Nassif A., Capretz L, and Ho D. (2012). Estimating Software Effort Using an ANN Model Based on Use Case Points, International Conference on Machine Learning and Applications, IEEE, 42-46.

Ochodek O., Nawrocki J., and Kwarciak K. (2011). Simplifying effort estimation based on Use Case Points. Information and Software Technology, 53(3), 200-213.

Parkash J. and Karambir (2014), COCOMO II Model Parameter Optimization using PSO and Effort Estimation, Int. Journal of Information Technology & Mechanical Engineering - IJITME, 1(4), 1-11.

Putnam L., and Myers W. (1991). Measures for Excellence: Reliable Software on Time, Within Budget, Prentice Hall, 234.

Popli R., and Chauhan, N. (2014). Cost and effort estimation in agile software development. Optimization, Reliabilty, and Information Technology (ICROIT). 2014 International Conference on, 57–61.

Rijwani P. and Sonal J. (2016). Enhanced Software Effort Estimation using Multi-Layered Feed Forward Artificial Neural Network Technique, Twelfth International Multi-Conference on Information Processing, 89(1), 307-312.

Sarno R., Sidabutar J., and Sarwosri (2015), Comparison of different Neural Network architectures for software cost estimation, International Conference on Computer, Control, Informatics and its Applications (IC3INA). 68-73

Rijwani P., Sonal J., and Dharmesh S. (2014), Software Effort Estimation: A Comparison Based Perspective, International Journal of Application or Innovation in engineering and management (IJAIEM), 3(12).

Rezvani S. (2015). A New Similarity Measure of Generalized Fuzzy numbers Based on Left and Right Apex Angles (II). Palestine Journal of Mathematics. 4(1), 446–454.

Saade J. and Diab D. (2004). Defuzzification Methods and New Techniques for Fuzzy Controllers, Iranian Journal of Electrical and Computer Engineering, 3(2),161-174

Tailor O., Saini T. and Rijwani P. (2014). Comparative Analysis of Software Cost and Effort Estimation Methods: A Review, International Journal of Computer Science and Mobile Computing, IJCSMC, 3(4), 1364 – 1374.

Tomaszewska K. (2014). The Application of Horizontal Membership Functions to Fuzzy Arithmetic Operations, Journal of Theoretical and Applied Computer Science, 8(2), 3-10.