SOFT COMPUTING APPROACHES FOR ESTIMATING SOFTWARE DEVELOPMENT EFFORT

MELAKU DEMSIS

A Thesis submitted to the School of Graduate Studies of Addis Ababa University in partial fulfillment of the requirements for the Degree of M Sc in Computer Science

June 2005

Approval Board of Examiners

1. Chairman, Department Committee Signature

2. Advisor Signature

3. Examiner Signature

4. Examiner Signature

5. Examiner Signature

This t hesis my or iginal w ork a nd has not b een p resented f or a d egree i n an y o ther university, and t hat a ll s ources o f material used f or t he t hesis ha ve be en d uly acknowledged.

Name	Signature
Melaku Demsis	_____

Advisor Confirmation

Name	Signature
Dr. Yirsaw Ayalew	_____

Acknowledgements

This thesis would not have been possible without the extraordinary support, advice and encouragement of Dr.Yirsaw Ayalew. I would like to thank him, both for the invaluable comments and leadership during this thesis, and for accepting the responsibility of being my advisor even when he left the university. It has been an honor for me to work with him, and I hope I will have the opportunity to keep learning from his insights and character.

My heart felt gratitude although goes to my family especially to my mother Woltemeskel G/Hiwot and my brother Dr.Abebe Demsis.

Finally, my deepest gratitude goes to all my friends. I have been fortunate to have their friendship ever since.

Abstract

Estimating software development effort has been the focus of intensive research investigations in the field of software engineering, especially software project management. As a result, various cost estimation models have been proposed. However, most of the cost estimation models that have been proposed do not properly handle imprecision and uncertainty that is inherent in software project effort estimation. In this thesis, we propose a neuro-fuzzy approach that has the potential to improve software development effort estimation by applying fuzzy logic and neural networks in the process of effort estimation. This approach has been applied for the use case based estimation model, which is used in the process of object-oriented software development project. The applicability of the approach has been tested using an experiment with fifteen historical software project data. The result of the experiment shows that the neuro-fuzzy approach provides a reasonably better accuracy in software development effort estimation.

Keywords: software project planning, effort estimation, use case based effort estimation, software development

Table of Contents

Contents **Pages**

LIST OF TABLES

List of Figures

1. Introduction

From the start of the software era in the 1950s until today, software development has become increasingly more complex and expensive, particularly if it includes many people working over a relatively long time. A large software project may involve hundreds of people and span over years. A project of this dimension can easily turn into chaos if proper management controls are not in place. Traditionally, computer professionals have attached little importance to management and more to technical skills. As a result, major software disasters were encountered leading to unexpected loss of investment, personnel and even company closures due to bankruptcy [3]. These failures made the developers realize the importance of management in software development. Therefore, software project management is taken as one of the major activities in software development process.

Software project management involves project planning, monitoring and control. Planning is the most important activity without which, no real monitoring or controlling is possible. The basic activity in software project plan is estimating effort, schedule, and other resource expenditures. The process of such estimation is referred to as software cost estimation [13]. The bulk of the cost of software development is due to the human effort, and most cost estimation methods focus on this aspect and give estimates in terms of person-months.

Reasonable software cost estimates are critical to both developers and customers. These estimates can be used for generating request for proposals, contract negotiations, scheduling, monitoring and control [5, 24]. Underestimating the costs may result in management approving proposed systems that then exceed their budgets, with underdeveloped functions and poor quality, and failure to complete on time. Overestimating may result in too many resources committed to the project, or during contract bidding, result in not winning the contract, which can lead to loss of jobs.

Although software cost estimation is so important, it is difficult to undertake. There are factors accountable for this [4]:

- A great number of factors (cost drivers) have an influence on the effort and time to develop software. These cost drivers are difficult to determine in operation
- The software industry is usually developing new products. In other industries, it is more usual to produce the same product over and over
- Estimates are made in a hurry. Often estimates are needed early in the development process, causing estimators to write an estimate too quickly for a system they do not fully understand.
- There is a lack of data on completed software projects, which could support project management in making estimates

Faced with the fact that software cost estimation is difficult, how should we approach this challenge? That question has no simple answer. Since the software era until about 1970, software cost estimation was performed manually; using simple rules of thumb or local estimating algorithms developed using trial and error methods. Since the late 1960s there has been a continuous development of estimation models and tools to support the estimation process. These models can be grouped into two categories [3]:*Algorithmic models*, and *Non-algorithmic models*.

Algorithmic models provide one or more algorithms which produce a software cost estimate as a function of a number of variables considered to be the main cost drivers [3]. Estimation models such as COCOMO, Function Point, and Use case based are all examples of algorithmic model. There are three main drawbacks to these models [11,12]. First, they rely heavily on estimates of quantities such as size (KLOC) at early stage where much is unknown about the project. Second, current models use linear regression techniques to correlate cost drivers, which, in fact, are not linear relationships at all. Third, the historical project data used to develop these models are usually limited in size, and represent dissimilar projects

Non- algorithmic models include estimation by analogy, expert judgment, parkinson, price-to-win, bottom-up, and top-down approaches [3]. Each of these approaches has their own advantages and disadvantages but non-algorithmic models are generally subjective and are not repeatable.

In general, whilst most practitioners recognize the importance of accurate and reliable predictions of development effort, current estimation techniques are often inaccurate and unreliable.

1.1. Statement of the Problem

Software cost estimation has been subject to research for more than 30 years, but the problem has not yet been fully addressed. To overcome the problem, many researchers have recently begun to turn their attention to a new approach of estimating software development effort known as soft computing. *Soft computing* is a collection of techniques that aim at exploring the tolerance for imprecision and uncertainty to achieve tractability, robustness, and low solution cost. Its principal constituents are fuzzy logic, neurocomputing, and probabilistic reasoning [14]. These methods have been applied to many real-world problems. In many cases, good results have been achieved by combining different soft computing methods.

The rationale behind the visibility and importance of soft computing approach is that there are many real world problems, which do not lend themselves to solution by the techniques of hard computing because the information needed is not available and/or the systems under consideration are not sufficiently well defined. These problems are common in such areas as economic planning, living systems, large-scale societal systems, and human decision-making. The trouble with software development effort estimation can also be categorized into such problems. Therefore, soft computing approach could have a potential to solve this problem. This potential however is not explored in previous works. Therefore, the main purpose of this paper is to explore the potential of soft computing techniques for estimating software development effort.

1.2. Overview of the Thesis

The remainder of this thesis is organized as follows:

Chapter 2 presents related works. It contains a brief discussion of four research works that are related to soft computing in software cost estimation.

Chapter 3 presents the approach used in this work. It starts with a brief introduction on the problem of software cost estimation. Section 3.1 contains description of use case based estimation model and section 3.2 discusses how fuzzy logic is applied to use case based estimation model. The neural network approach for use case based estimation model is presented in section 3.3 and finally section 3.4 describes the datasets used for experiment.

Chapter 4 describes the process and result of experiment. The fuzzy logic and neural network experiments are presented in section 4.1 and 4.2 respectively. In chapter 5, discussion of the experiment results are presented. The results of the experiments are evaluated from different perspectives. The benefits and limitations of the neuro-fuzzy approach are also described in this chapter.

Chapter 6 describes some of the implementation of the neuro-fuzzy approach. It presents the architecture, class diagram and interface of the tool developed at the prototype level. Finally, chapter 7 presents some concluding remarks and possible future works.

2. Related Works

Several research works which are more or less related to this work have been carried out in the last few years [5, 6, 7, 10, 11, 12, 13, and 25]. Until the mid 1990s, the main focus of software cost estimation research was on algorithmic models. The lack of consistent results caused researchers to look at alternative techniques, such as soft computing. In this section, some research works dealing with soft computing approach for software cost estimation are reviewed.

2.1 Works on Fuzzy Logic Approach

Idri et al [10] have produced a paper entitled "Investigating Soft Computing in Case-based Reasoning for software Cost Estimation". The main purpose of their work was to improve the Fuzzy Analogy approach, which was proposed by their previous work [12], in order to handle the uncertainty in its development effort estimate. They discussed two sources of uncertainty in the fuzzy analogy approach. The first source is from measurement errors when evaluating the input variable for the approach. To avoid this uncertainty, they used the membership degrees in the evaluation of the similarity between projects. The second source of the uncertainty is related to the accuracy of the final output. To handle this one, they adopted the idea of other authors that says cost estimation models must be able to generate a set of values with a probability distribution for the estimated cost. To check for the possibility, they have conducted an experiment using COCOMO'81 data set. Based on their experiment they have concluded that the deterministic (their previous approach) is not useful in software cost estimation. Consequently, they have chosen the non-deterministic approach to handle the uncertainty in the fuzzy analogy approach. The non-deterministic approach uses possibility rules and a method called max-based aggregation to generate the cost possibility distribution for new project. By using fuzzy logic and the possibility theory in its estimation process, their approach fulfills two criteria of soft computing, i.e., the tolerance of imprecision when describing software project, and the uncertainty when estimating the development cost.

The same authors have carried out another research on applying fuzzy logic to COCOMO cost model [11]. Their purpose was to investigate the issue of the compatibility of COCOMO with fuzzy logic. They have used fuzzy sets rather than classical intervals to represent the linguistic values ('*very low*', '*low*', etc) in the COCOMO's cost drivers set. For each cost driver and its associated linguistic values, they have defined the corresponding fuzzy set. They have also evaluated their work by comparing it with the original intermediate COCOMO. Their result implies that the fuzzy intermediate COCOMO tolerates imprecision in its inputs (cost drivers) and consequently it generates more gradual output (cost). According to them, the fuzzy COCOMO is less sensitive to the changes in the inputs, contrary to the original intermediate COCOMO. Finally, they have concluded that the accuracy of COCOMO model is certainly affected by fuzzy logic approach.

Both of the above works are similar with the current one as regards fuzzy logic. Learning ability, which is one important criterion of soft computing concept, is not incorporated in their approaches. Their approaches are highly depend on kilo lines of code, which is difficult to estimate at the early stage of software development process.

2. 2 A work on Neural Network Approach

Boetticher [5], conducted an assessment of metric contribution in the construction of neural network based effort estimator. This work describes the process of building a neural network based model for measuring software effort. It demonstrates a process for extracting a set of software metrics from a program, associating the metrics with a program effort to construct a neural network model. Using data from a company of petrochemical-industry software, over 33, 000 experiments were conducted varying test suites, neural network architectures and input parameters. The inputs for the various network models were program units corresponding to a program written in Delphi. Each unit may contain object, variable, type and content definitions along with programming logic in the form of functions or procedures. In neural network terminology, each

program is considered to be a vector. For each vector, nine input measures were collected reflecting a program size, vocabulary, number of objects, and complexity. The output is effort in hours, needed to create the unit. In his neural network architectures, the maximum number of hidden layers was limited to 3 and the numbers of hidden nodes per hidden layers were twice the inputs. In order to build and train neural network, an automated neural network program was used. The program uses an algorithm called quickprop, which is a variant of back-propagation learning algorithm. This approach was cross validated with data collected from different projects of different companies. The validation results produce estimates within 30% of the actual 73.26% of the time. Based on the result, Boetticher has concluded that neural network model has reasonably good predictive qualities. In general, this work has described the neural network approach, which is one of the methods of soft computing. Incorporating other methods might provide more accurate models. This research focused only on product metrics, in the formulation of effort estimation. It doesn't consider other cost drivers like personnel factors.

2.3 A work on Neuro-genetic Approach

Shukla [25] has also done a work on neuro-genetic prediction of software development effort. His work presents a genetically trained neural network (NN) predictor trained on historical data. He has used a multi-layered feed forward NN with 39 input neurons, each neuron corresponding to one of Bohem's[3] features, hidden layers of neurons to develop the desired mapping and 1 out put corresponding to the predicted effort in person-months. He has also applied genetic algorithm to the feed-forward NN and his model was named as GANN(Genetic Algorithm Neural Network). In his work, COCOMO data set comprising 63 projects and Kemrer dataset consisting of 15 projects were merged to form a single data set of 78 projects. The merged data set was used to validate the neuro-genetic approach. He has conducted several simulation experiments to obtain the relative performance of his approach with recursive partition regression and back propagation trained NN which were reported in previous works. The result of his work shows that the neuro-genetic approach is significantly better prediction of software development effort

than the others. However, the inputs to his model were still subjective. As a result, the reported result may not be the same for other projects.

None of the above works have used the two concepts of soft computing, i.e., fuzzy logic and neural network together in their approaches. This work attempts to explore the potential of soft computing by combining the two concepts. Besides combining the two concepts, this approach is quite different from all previous works in that, it uses use case based estimation model as a basis for incorporating the soft computing concept. The details of our approach are discussed in the next chapter.

3. Effort Estimation using Neuro-fuzzy Techniques

Software development involves a number of interrelated factors, which affect development effort and productivity. Since many of these relationships are not well understood, reasonable estimation of software development time and effort is a difficult problem. Most estimation models in use or proposed in the literature are based on regression techniques. This thesis examines the potential of two soft computing approaches i.e. fuzzy logic and artificial neural networks for creating development effort estimation models.

As mentioned in Chapter 1, soft computing approaches have already been used successfully to solve problems in a number of different areas, such as adaptive control, voice and speech recognition from text and medical diagnosis. It is also believed that this approach could solve some problems in the area of effort estimation. According to Idri [12], soft computing approach can offer the following advantages to software effort estimation:

- Capability to adequately model the complex set of relationship between factors (cost drivers) or between effort and the cost drivers
- Capability to learn from historical project data (specifically to neural network)

Even though such advantages are reported, studies in this area are not yet mature. In the current work, fuzzy logic and neural network have been applied to use case based estimation model (UCBEM). UCBEM was selected as a basis for this work because first, it deals with object-oriented software development paradigm, which is current and widely used paradigm. Second, use cases and other factors (cost drivers) used in this model could be identified and described at the early stage of software development. Some studies [1] have also shown that UCBEM could estimate software development effort with acceptable level of accuracy. So, why and how soft computing methods should be incorporated to UCBEM? This question is answered in the next three sections of this chapter. While section 3.1 describes UCBEM, the fuzzy logic and neural network approaches are discussed in section 3.2 and 3.3 respectively.

3.1 Use case Based Estimation Model

Cost estimation based on objects was introduced in 1993 by Karner [17]. This estimation method requires that it should be possible to count the number of transactions in each use case. A transaction is an event occurring between an actor and the system. The general process in use case based estimation model involves four major steps:

1. The actors in the use case model are categorized as *simple*, *average* or *complex*. A simple actor represents another system with a defined Application Interface (API); an average actor is another system interacting through a protocol such as TCP/IP; and a complex actor may be a person interacting through a graphical user interface or a web page. A weighting factor is assigned to each actor category: Simple: 1, Average: 2, Complex: 3. The total *unadjusted actor weight* (*UAW*) is calculated counting the number of actors in each category, multiplying each total by its specified weighting factor, and then adding the products.
2. The use cases are also categorized as *simple*, *average* or *complex*, depending on the number of transactions, including the transactions in alternative flows. Included and extended use cases are not considered. A simple use case has 3 or fewer transactions; an average use case has 4 to 7 transactions; a complex use case has more than 7 transactions. A weighting factor is assigned to each use case category: *unadjusted use case* Simple: 5, Average: 10, Comple
   weights (UUCW) are calculated counting the number of use cases in each category, multiplying each category of use case with its weight and adding the products. The UAW is added to the UUCW to get the *unadjusted use case points* (*UUPC*).

3. The use case points are adjusted as per the values assigned to a number of technical and environmental factors. Each factor is assigned a value between 0 and 5 depending on its assumed influence on the project. A rating of 0 means the factor is irrelevant to the project; 5 means it is essential. The technical factors measure the complexity of a project regarding non-functional requirements. These factors influence the result of the use case point. According to Karner [17], the

project co mplexity f actors ar e t he features related t o p erformance, p ortability, security, reusability of the code, and others (See Table 3.1).

Factor	Description	Weight (examples)
T1	Distributed System	2
T2	Response adjectives	2
T3	End-user efficiency	1
T4	Complex processing	1
T5	Reusable code	1
T6	Ease of installing	0.5
T7	Ease of use	0.5
T8	Portable	2
T9	Ease of changing	1
T10	Concurrent	1
T11	Security features	1
T12	Access for third parties	1
T13	Special training required	1

Table 3 .1 Technical Complexity Factors (TCF)

The Technical Factor (TCF) is computed using the following formula:

TCF = 0.6 + (.01*TFactor).

The environment factors are related to familiarity with the development process to be used in the project, the past experience in the application, motivation, stable requirements, and others (See Table 3.2).

Factor	Description	Weight (examples)
F1	Familiar with RUP	1.5
F2	Application experience	0.5
F3	Object-oriented experience	1
F4	Lead analyst capability	0.5
F5	Motivation	1
F6	Stable requirements	2
F7	Part-time workers	1
F8	Difficult programming language	1

Table 3.2– Environmental Factors (EF)

The Environmental Factor (EF) is computed as follows:

EF = 1.4+(-0.03*EFactor)

The *adjusted use case points (UCP*) are computed as follows:

UCP = UUCP*TCF*EF

4. Karner [17] proposed a factor of 20 staff hours per use case point for a project estimate. Therefore, the effort will be calculated as : UCP*20.

The following two sections describe how the soft computing methods are applied to the above described use case based estimation method.

3.2 Fuzzy Logic Approach for Use Case Based Cost Estimation Model

The primary motivation of fuzzy set theory is the desire to build a formal quantitative structure capable of capturing the imprecision of human knowledge, that is, the manner in which knowledge is expressed in natural language. This theory seeks to bridge the gap between traditional mathematical models needed for physical systems, and the mental representation, generally imprecise, of such systems.

Fuzzy set is characterized by a membership function, $\tilde{A}$, which maps the elements of a domain, space or discourse universe X to a real number in [0,1]. Formally, $\tilde{A} : X \rightarrow [0,1]$. Thus, a fuzzy set is represented as a set of ordered pairs in which the first element is $x \in X$, and the second, $\mu\tilde{A}(x)$, is the degree of membership of x in X, which maps x in the interval [0,1].

The membership of an element within a certain set becomes a question of degree, substituting the actual process imposed by set theory, when this treatment is not suitable. In extreme cases, the degree of membership is 0, in which case the element is not a member of the set, or the degree of membership is 1, if the element is a 100% member of the set [15].

The central idea of extending Use Case Based Estimation Model (UCBEM) to Fuzzy - UCBEM through fuzzy set theory is to expands emantics (Simple, Average, and Complex) used to categorize use case complexities. This expansion is required because, the original UCBEM uses disjoint manner of classifying use case complexities. For example, a use case with 7 transactions is classified as “average”, receiving 10 weighting factor. Another use case, which has 8 transactions, is classified as “complex”, receiving 15 points. Two serious problems are immediately evident in such system of classification: (i) dissimilar use cases receive the same point values, and (ii) similar use cases are abruptly classified into different groups. This inconsistency can become even

worse when there is a large number of transactions in the system that lie within the border areas of the specified intervals.

The above-mentioned problems could be solved using triangular membership fuzzy numbers. A triangular membership fuzzy number can be represented by A (a, b, c), whose membership functions are presented in the following equation:

$$A = \begin{cases} 0, & x \le a \\ (x-a)/(b-a), & x \in (a, b) \\ (c-x)/(c-b), & x \in (b, c) \\ 0, & x \ge c \end{cases}$$

The values a, *b,* and c respectively identify the lower, middle, and upper limits that determine the shape of the triangle.

In general, extending UCBEM to Fuzzy- UCBEM consists of two steps:

1. Fuzzification of the linguistic terms of use case complexity by generating fuzzy numbers
2. Defuzzification of the linguistic terms (Calculate the numeric values).

First Step

In this step, triangular fuzzy numbers *A*(a, *b, c*) are generated for each linguistic term T_i (*Simple*, *average*, *Complex*) belonging to the complexity of use cases. Initially, the value a_i assumes the lower limit of the linguistic term *i* of the complexity being considered. In the use case based estimation model, a_i is 1,4 and 7 for *simple, average* and *complex* respectively. The value b_i is calculated as: $b_i = a_i + (a_{i+1})/2$. The values for c_i will be values of b_{i+1} ($c_i = b_{i+1}$). Using these formulas, the values of b_i are 2.5, 5.5, and 8.5 for simple, average, and complex respectively. After computing the values for b_i, the values for a_i needs to be re computed except for the first one. Therefore, a_i will be equal to b_{i-1} for the second and third linguistic terms ($a_i = b_{i-1}$).

Note: The value for *b* for the last linguistic term (complex) is 8.5, which is calculated using a simple linear extrapolation (equal distance) between other linguistic terms.

As an example, observe the linguistic term *simple*, as depicted in Fig. 3.1. In this case, *a* = 1, *b*= (4+1)/2 = 2.5 and *c*, = (7+4)/2 = 5.5

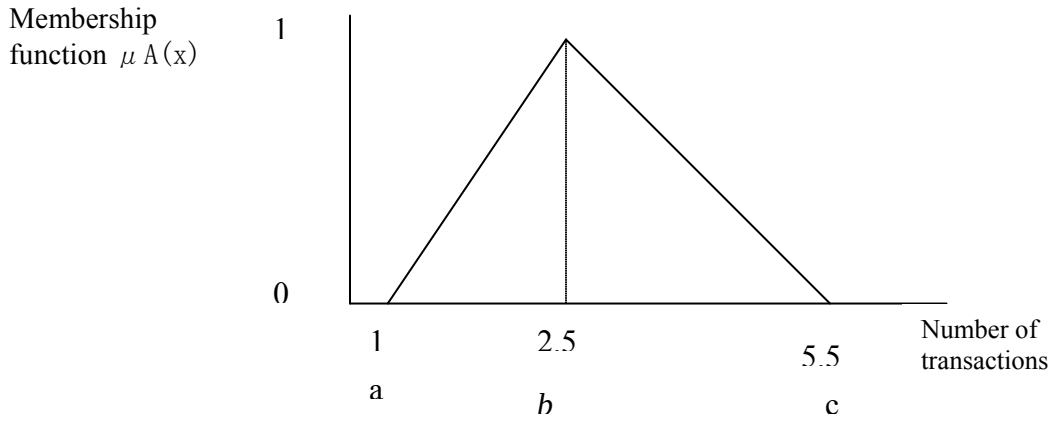

Figure 3.1: Triangular fuzzy number for *simple* complexity

The following figure shows triangular fuzzy numbers for the three complexities.

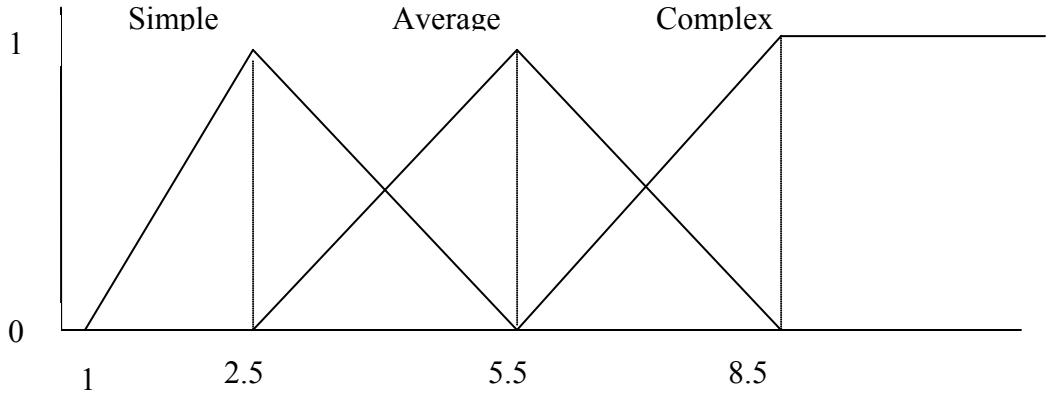

Figure 3.2. Triangular fuzzy Number

Second Step

In Fuzzy- UCBEM, to obtain the weighting factor of a use case *Wu* from fuzzy numbers, where $\mu A(x) < 1$, we can execute the following defuzzification process:

$Wu = \mu A(x)Wui+(1- \mu A(x)).Wui+1$

Example

Let's consider a simple library system with the following use cases and their respective number of transactions:

Use case Name	Number of transactions	Weighting factor based on UCBEM
Borrow Item	4	10
Return Item	3	5
Catalogue New Item	3	5
Maintain Borrower Details	8	15
Total Weighting		35

Table 3.3: Use cases and their weighting factor according to UCBEM

In the Fuzzified UCBEM, the weighting factor of use cases could be changed based on the triangular shaped fuzzy rules. For instance, the weighting factor of use cases with 3 transactions can be computed as follows:

$\mu A(3) = (5.5– 3)/(5.5 – 1) = 0.55$, and the complement of $\mu A(3) = 0.45$

This means that the use case with 3 transactions is a member of *simple* use case with degree of 0.55 and it is a member of average use case with degree of 0.45. Accordingly, the weighting factor of the use case will be calculated as : 0.55(5)+0.45(10) = 7.25

Based on the triangular fuzzy rules, Table3.3 will be converted as follows (Table 3.4)

Use case Name	Number of transactions	Weighting factor based on Fuzzy -UCBEM
Borrow Item	4	8.3
Return Item	3	7.25
Catalogue New Item	3	7.25
Maintain Borrower Details	8	14.15
Total Weighting		36.95

Table 3.4: Use cases and their weighting factor according to Fuzzy-UCBEM

According to the above tables, the total weighting factor of UCBEM is 35 and that of Fuzzy-UCBEM is 36.95. This increment can bring a change on the final estimation. The changes will be more significant when there are large number of transactions in the system those lies within the border areas of the specified intervals.

3.3 Neural Network Approach for Use Case Based Cost Estimation Model

Artificial Neural Networks (ANN) commonly called neural networks, are massively parallel systems inspired by the architecture of biological neural networks, comprising simple interconnected units (artificial neurons). The neuron computes a weighted sum of its inputs and generates an output. Neural network can be considered as a recent development but enjoys a continuous interest by many researchers in the field of computer engineering and artificial intelligence. The interest in neural networks comes from their potential to find a solution to various problems of application domains. Neural networks have been successfully applied to a wider range of data-intensive applications, such as voice recognition, medical diagnosis and financial forecasting. They are also being applied in software cost estimation. Performing estimation by neural network could have two main advantages. First, it allows learning from previous situations and outcomes. The learning criterion is very important to software cost estimation models because software development is supposed to be continuously evolving, and software effort estimation is commonly done based on past completed

projects. Second, it can model a complex set of relationships between the dependent variable (such as cost or effort) and the independent variables (cost drivers).

There are different types of neural network models, but the most common neural network model is the multi-layer perceptron (MLP) [9]. The goal of MLP is to create a model that correctly maps the input to the output using historical data so that the model can be used to predict an output. A graphical representation of a MLP is showed in Figure 3.3.

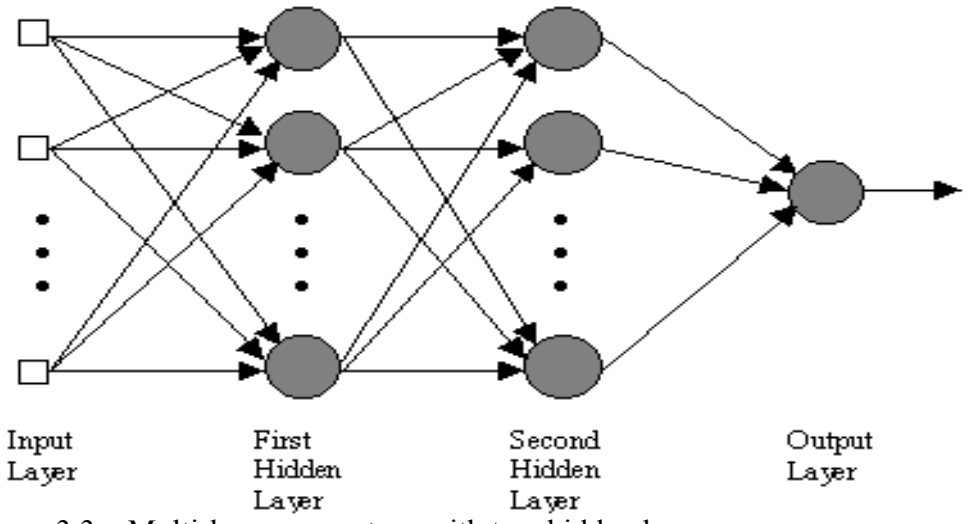

Figure 3.3 – Multi-layer perceptron with two hidden layers

The development of such a neural model began by working out an appropriate layout of neurons, or connections between network nodes. This includes defining the number of layers of neurons, the number of neurons within each layer, and the manner in which they are all linked. The weighted estimating functions between the nodes and the specific training algorithm to be used must also be determined. Once the network has been built, the model must be trained by providing it with a set of historical project data inputs and the corresponding known actual values for project schedule and/or cost. The model then iterates on its training algorithm, automatically adjusting the parameters of its estimation functions until the model estimate and the actual values are within some pre-specified delta set.

In general, building a neural network for any kind of prediction (estimation) involves the following major steps:

1. Identify the network inputs and outputs
2. Process the input and output values so that they fall into a numeric range, usually between 0 and 1
3. Choose an appropriate topology for the network by defining the number of hidden layers
4. Train the network on a representative set of examples
5. Test the network on a test set independent of the training set and retrain the network if necessary
6. Apply the generated model to predict outcomes in real situations.

In our case, the inputs for the network are the independent variables of the use case based estimation model. These variables include actor's complexity weighting factors, the fuzzified use case complexity weighting factors, the technical factors, and the environmental factors. The output of the network is the predicted effort.

In addition to inputs and outputs, the use of the neural network approach to estimate the software effort requires certain decisions and choices about the architecture, learning algorithm, and activation function.

The architecture of a neural network is the specific arrangement and connection of the neurons organized in the form of layers [14]. It is defined by the number of layers, the number of units per layer, and interconnection pattern between layers. The most important decision to be made in neural network architecture is the number of hidden nodes. It is this choice that determines the accuracy with which the network can fit the data. The greater the number of nodes, the more precisely the network can learn a given function. While this may imply that more is better this only applies where the goal is to learn some deterministic function, where an exact mapping from inputs to outputs exists. In such a case identical projects would have identical outputs (such as effort or error density). This is obviously not the case with software development where there are

dissimilar projects and dissimilar data. For such stochastic data, the goal is instead to develop a model that generalizes to new data. Using more hidden nodes and thus more weights generally increases the chance that the network will find a solution that is not the best possible (called a local minimum). In general, the recommended approach for problems like software effort estimation is to start with a small number of hidden nodes, even with one or two, and after developing the model, add more nodes gradually until performance on validation data stops improving. Based on this recommendation, our typical network will have one input layer with four neurons, at least one hidden layer, and one output layer with one neuron. This kind of architecture is known as multi layered feed forward architecture or multi layer perceptron (MLP).

After defining the architecture, the next step is dealing with the learning process. According to Hykin [9], learning in a neural network is defined as "a process by which the free parameters of a neural network are adopted through a process of simulation by the environment in which the network is embedded". In other words, it is the process of adjusting the network connection weights so that the network produces the appropriate output patterns (effort estimation, in our case) for corresponding input patterns (software attributes and cost drivers values, in our case). The idea is to use a set of examples called a training set, to adjust the network weights to the right predictive values.

The process of making the network to learn the solution of a problem follows a set of well-defined rules called the learning algorithm. Many learning algorithms have been invented to help find an optimum solution, but the most common learning algorithm is *back-propagation*. *Back-propagation* starts with a random set of weights uses an example in the training set to estimate the output, and compares the estimate with the actual value. The *back-propagation* algorithm uses this comparison to calculate the estimation or classification error. The error is then feedback through the network and the weights in the network are adjusted to minimize the error. The bigger the error the more the weights are modified. Each node sees their input node as advisors. The more an input node contributes with the wrong advice, the more its weight is downgraded. This way of weight adjustment is suitable for effort estimation problem because the output (effort)

should depend on the contributions of the different inputs (cost drivers). Therefore, the learning algorithm for our network model is *back-propagation.*

The actual weight calculation also includes the slope of the filtering (activation) function and a learning rate value. The favored function for the back-propagation algorithm is the *sigmoid* function. Since the function is not linear, it is appropriate for software effort estimation problem.

In back-propagation process, after the adjusted weights are back-propagated from the output to the input nodes, a new example is shown to the network. After being shown enough training examples, the weights on the network no longer change significantly. This is when the training stops and the network is said to have learned the concept. Critical to this process is the learning rate, i.e., how much the weights are adjusted to compensate the error at each training cycle. The learning rate controls how quickly the network reacts to a particular error. If the learning rate is too aggressive, the network will act quickly to correct one type of error but may corrupt the weight structure previously assembled to prevent other types of error. This might bar the network from converging to a specific set of weights. On the other hand, if the learning rate is too conservative, the network will take a long time to converge to a specific set of weights. This is especially troublesome when one has a limited training set, as is in our case. So, for our case, good approach for the learning rate is to start with aggressive and to slowly decrease it as the network is being trained. Initially, the weights are random, so large oscillations are useful to get in the range of their best values. However, as the network gets closer to a solution, the learning rate should be decreased so that the weights can be fine-tuned to an optimum solution.

A graphical representation of our network model is showed in Figure 3.4

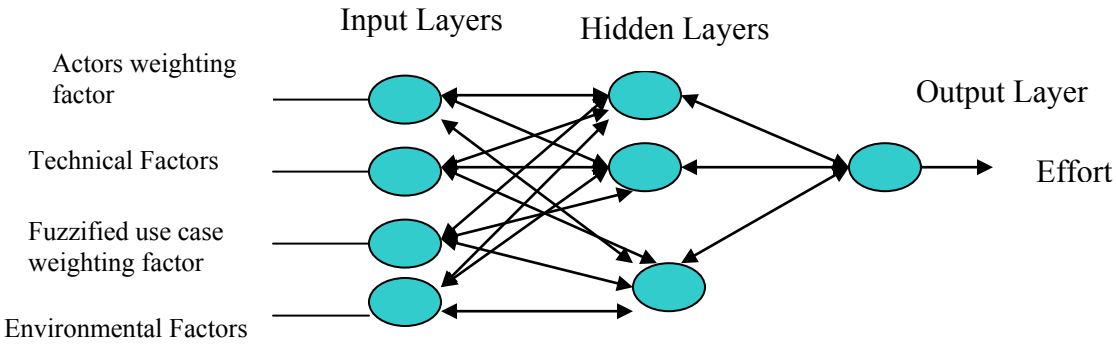

Figure 3.4. *A Neural Network for Software Development Effort Estimation*

3.4 The Data Used for the Experiment

To validate our approach, we were required to get historical data on completed software projects. Since our approach requires data on projects developed using an object-oriented paradigm, getting access to such data was a major impediment to this work. An effort was made to acquire the required data by contacting different researchers in the area of software development effort estimation, specifically in the area of use case based estimation models, but most of them did not reply. Finally, we managed to secure data on fifteen software projects, out of which, five were on industry projects and ten on student projects. Data on the twelve projects (two on industry and ten student projects) were originally collected by Ribu [21] for MSc thesis work. One of the industry projects was an Internet application for a major bank in Norway, and the other was a real-time system developed as part of a large commercial system. The student projects were of two categories. Five of them were on Internet applications, and the other five were on stand-alone applications for office management.

The other data on three industry projects were collected by Karner[17] who is the inventor of use case based estimation model. One of the industry projects was an information system for operation support of performance management in telecommunication networks. The second and third projects were a LAN management system and a telecommunication system, respectively

Cognizant of the scarcity of data on the area, it is assumed that the data we obtained are sufficient to validate our approach. Consequently, our experiment is done based on these available data. The process and the results of the experiment are discussed in the next chapter.

4. Experiment

Our approach for software development effort estimation is based on fuzzy logic and neural networks. The applicability of each method was tested using the acquired data set. This chapter presents the process of the experiment and the impact of each method on the use case based estimation model.

4.1. The Fuzzy Logic Experiment

Fuzzification and defuzzification of use case complexity weighting factor was applied as per the proposed approach. For the purpose of this experiment, the fuzzy computation was performed using spreadsheet software. As described in chapter 3, the membership function of each use case complexity-weighting factor was computed based on the number of transactions.

Each membership functions were computed according to the triangular membership function. The membership computations are shown in the following formulas:

Membership to Simple (MS):

$$MS = \begin{cases} 1, & x<a \\ (b-x)/(b-1), & a<x<b \\ 0, & x>b \end{cases}$$

where: a, b, and c represent values 2.5, 5.5, and 8.5, as they are computed in chapter three. x is number of transactions in a use case

Membership to Average (MA):

$$MA = \begin{cases} 0, & x<a \\ 1\text{-}MS, & a<x<b \\ (c-x)/(c-b), & b<x<c \end{cases}$$

Membership to Complex (MC):

$$MC = \begin{cases} 0, & x<b \\ 1\text{-}MA, & b<x<c \\ 1, & x>c \end{cases}$$

After computing the membership of each use case, the fuzzy weight (FW)is computed as:

$$FW = \{ (MS*5) + (MA*10) + MC*15$$

The following table (table 4.1) shows a sample of use cases with their fuzzy membership function. The use cases in the table are drawn from project 15. Project 15 is one of the industry projects obtained from Karner's work [17]. The project was on a telecommunication system.

Use Cases	Number of transaction	Membership to Simple	Membership to Average	Membership to Complex	Fuzzy Weight
UC1	4	0.33	0.67	0.00	8.33
UC2	5	0.11	0.89	0.00	9.44
UC3	6	0.00	0.83	0.17	10.83
UC4	6	0.00	0.83	0.17	10.83
UC5	7	0.00	0.50	0.50	12.50
UC6	5	0.11	0.89	0.00	9.44
UC7	4	0.33	0.67	0.00	8.33
UC8	7	0.00	0.50	0.50	12.50
UC9	5	0.11	0.89	0.00	9.44
UC10	7	0.00	0.50	0.50	12.50
UC11	5	0.11	0.89	0.00	9.44
UC12	4	0.33	0.67	0.00	8.33
UC13	7	0.00	0.50	0.50	12.50
UC14	5	0.11	0.89	0.00	9.44
UC15	4	0.33	0.67	0.00	8.33
Total					152.22

Table 4.1 Fuzzy membership functions of use cases in project 15.

Accordingly, the total fuzzy weights of each project in the data set are computed (see Apendix A for details).

The total fuzzy weights of each project are shown in the following table (table 4.2).

PROJECTS	Total Fuzzy Weight
Project1	547.22
Project 2	662.50
Project3	45.56
Project4	41.67
Project5	36.67
Project6	46.94
Project7	38.06
Project8	76.67
Project9	24.17
Project10	57.22
Project11	47.22
Project12	59.17
Project13	99.17
Project14	488.61
Project15	152.22

Table 4.2 Total fuzzy weights of use cases.

4.1.1. The impact of Fuzzy logic on the Use case based estimation model.

In order to show the impact of the fuzzy logic, the total fuzzy weight was used to compute the effort. To compute the effort, the calculated total fuzzy weights were adjusted using the Karner's formula:

AUCP = AW+UUCP*TCF*EF.

Effort =AUCP*20

Where, AUP = Adjusted Use Case Point

AW= Actor Weight

UUCP= Unadjusted Use Case Point

TCF= Technical Factors

EF= Environmental Factors.

In our case, i.e., for the Fuzzy–UCBEM, the UUCP is total fuzzy weight. The effort (hour) of each project is computed using this formula. The estimated effort which is computed using both UCBEM and Fuzzy-UCBEM, and the actual effort is shown in the following table (table 4.3).

Projects	**Estimated Effort (hour) Using Fuzzy-UCBEM**	**Estimated Effort (hour) Using UCBEM**	**Actual Effort**
Project 1	10586	10831	10043
Project 2	12725	14965	13933
Project 3	356	455	294
Project 4	357	421	371
Project 5	317	321	232
Project 6	381	421	371
Project 7	318	320	420
Project 8	590	570	580
Project 9	234	231	243
Project 10	459	498	595
Project 11	382	340	578
Project 12	481	469	490
Project 13	2115	2145	2150
Project 14	11667	12500	11980
Project 15	3791	3760	5400

Table 4.3 Actual versus Estimated results.

In some cases, Fuzzy-UCBEM can be worse than the classical UCBEM depending on the number of transactions of use cases in a project. For example, if many of the use cases in a project have 4 or 5 transactions (as project 13), fuzzy-UCBEM reduces the total use case point and the estimated effort. This is true because use cases with 4 transactions are categorized as average use case and obtain a weight of 10, but in fuzzy-UCBEM the weight of use case with 4 transactions is computed as 8.4. Such reduction has a

disadvantage if the project is underestimated by the classical UCBEM, but it will be advantageous if the project is overestimated by the classical UCBEM

4.1.2. Evaluation of UCBEM and Fuzzy-UCBEM

In order to evaluate the techniques with respect to their fitting accuracy, Mean Magnitude of Relative Error (MMRE) and Pred (I) were used. These methods are accepted as common accuracy indicators of software cost estimation models [8, 19]. They are also considered as more reasonable variables than other statistical criteria, since they measure the capability of prediction, not the statistical explanation. The relative error (RE) is ((actual effort-estimated effort) / actual effort) *100. The magnitude of relative error (MRE) is the absolute value of the relative error (MRE=|RE|). The mean magnitude of relative error (MMRE) is the average of all magnitudes of relative errors. MMRE is defined as follows:

$$MMRE = \frac{\sum_{i=1}^{N} \left| \frac{\text{Actual Effort} - \text{Estimated Effort}}{\text{Actual Effort}} \right| \text{ X } 100}{N}$$

where N is the number of projects.

If the MMRE is small, then we have a good set of Predictions. A model is accepted as a good estimation model, if it estimates with MMRE ≤ 0.25 [8, 19].

Prediction at Level I (Pred(I)), where I is a percentage, is defined as the quotient of number of cases in which the estimates are within the I absolute limit of the actual values divided by the total number of cases. For example, Pred(0.1) = 0.9 means that 90% of the cases have estimates within the 10% range of their actual values. A standard criterion for considering a model as acceptable is Pred(0.25) ≥ 0.75. This means that at least 75% of the estimates are within the range of the 25% of the actual values. The accuracy of

Fuzzy-UCBEM and UCBEM in terms of the above criteria's is computed, and the result is shown in the following table (table 4.4).

	Fuzzy-UCBEM	**UCBEM**
MMRE	0.13	0.18
Pred(0.25)	0.8	0.73

Table 4.4 Accuracy in terms of MMRE and Pred(0.25)

The above table shows that both Fuzzy-UCBEM and UCBEM could be accepted as good estimation models because both of them introduced a MMRE<0.25. The result also shows the level of accuracy of Fuzzy-UCBEM is better than UCBEM; it introduces less MMRE.

In terms of Pred(0.25), the result of Fuzzy-UCBEM is 0.8. This means, 80% of the estimates are within the range of the 25% of the actual values. In general, the result shows that applying fuzzy logic slightly improves the accuracy of UCBEM in terms of both MMRE and Pred (0.25).

4.2. The Neural Network Experiment

The other soft computing method applied to the UCBEM is the neural network approach. This section describes the neural network experiment carried out and its result. In order to see the effect of the fuzzified use case weighting factors in the neural network model, each experiment is carried out for both UCBEM and Fuzzy-UCBEM.

4.2.1. Pre-Processing of Data Sets

The input variables selected for the neural network model are based on the use case based effort estimation model. These variables are

1. Actor's complexity weighting factors (AW)
2. Use case complexity weighting factors (Fuzzified use case complexity weighting factors in the case of Fuzzy-UCBEM)-UCW
3. Technical complexity weighting factors (TCF)
4. Environmental factors (EF)

After setting the input variables, the next step was to characterize the input variables and normalize them because a neural network works best when the input and output data are between 0 and 1. The data were normalized to fit between 0 and 1 by dividing each value by 100000. This number was used because the maximum value in the data set is a five-digit number.

The available datasets were divided into training and test sets using *holdout* and *leave-one-out* methods. *Holdout* method means dividing the data into a mutually exclusive training and test sets. This means just keeping some percentage of the data for testing and others for training. Following this method, two data files were prepared, namely, the training data file, and testing data file. The training data set included data representative of the overall data range, and it includes 80% of the total data. The other 20% of the total data set was used for testing purpose. The testing data were selected randomly from the total data sets. The testing and training data were kept separate in order to create a model that will generalize new input situations, and not simply memorize the data. Table 4.5 and 4.6 respectively show data used for training and testing in Neuro-UCBEM, and table 4.7 and 4.8 show that of Neuro-fuzzy-UCBEM.

Projects	UCW	AW	TCF	EF	Actual
Project 1	0.0047	0.00006	0.0000104	0.0000092	0.10043
Project 2	0.00545	0.00048	0.0000103	0.0000087	0.13933
Project 3	0.0006	0.00005	0.0000085	0.00083	0.00294
Project 4	0.0004	0.00009	0.0000085	0.00083	0.00298
Project 6	0.00055	0.00007	0.0000085	0.00083	0.00371
Project 7	0.0004	0.00007	0.0000085	0.00083	0.0042
Project 8	0.0008	0.00007	0.0000085	0.00083	0.0058
Project 9	0.00025	0.00009	0.0000085	0.00083	0.00243
Project 10	0.00055	0.00008	0.0000085	0.00083	0.00595
Project 12	0.0006	0.00009	0.0000085	0.00083	0.0049
Project 13	0.001	0.0001	0.00001	0.0000097	0.0215
Project 14	0.005	0.0001	0.00001	0.0000117	0.125

Table 4.5 Normalized data set used for **training** in *holdout* method with UCW

Projects	UCW	AW	TCF	EF	Actual
Project 5	0.0004	0.00009	0.0000085	0.00083	0.00232
Project 11	0.0004	0.00007	0.0000085	0.00083	0.00578
Project 15	0.0015	0.0001	0.00001	0.0000117	0.054

Table 4.6 Normalized data set used for **testing** in *holdout* method with UCW

Projects	Fuzzy-UCW	AW	TCF	EF	Actual
Project 1	0.005472222	0.00006	0.0000104	0.0000092	0.10043
Project 2	0.006625	0.00048	0.0000103	0.0000087	0.13933
Project 3	0.000455556	0.00005	0.0000085	0.00083	0.00294
Project 4	0.000416667	0.00009	0.0000085	0.00083	0.00298
Project 6	0.000469444	0.00007	0.0000085	0.00083	0.00371
Project 7	0.000380556	0.00007	0.0000085	0.00083	0.0042
Project 8	0.000766667	0.00007	0.0000085	0.00083	0.0058
Project 9	0.000241667	0.00009	0.0000085	0.00083	0.00243
Project 10	0.000572222	0.00008	0.0000085	0.00083	0.00595
Project 12	0.000591667	0.00009	0.0000085	0.00083	0.0049
Project 13	0.000991667	0.0001	0.00001	0.0000097	0.0215
Project 14	0.004886111	0.0001	0.00001	0.0000117	0.125

Table 4.7 Normalized data set used for **training** in *holdout* method with **fuzzified** UCW

Projects	Fuzzy-UCW	AW	TCF	EF	Actual
Project 5	0.000366667	0.00009	0.0000085	0.00083	0.00232
Project 11	0.000472222	0.00007	0.0000085	0.00083	0.00578
Project 15	0.001522222	0.0001	0.00001	0.0000117	0.054

Table 4.8 Normalized data set used for **testing** in *holdout* method with **fuzzified** UCW

In *leave–one-out* method data sets are used for training and testing using the following steps:

- Leave one data point out of the dataset,
- Train the network model using the remaining projects,
- Test the model by predicting the value of the omitted project, and
- Repeat this process for each project in the dataset.

According to Kitchenham [18], this method is appropriate when we have a small dataset (e.g. 10-30 projects). In order to apply this method, data sets for training and testing were prepared for UCBEM and Fuzzy-UCBEM (table 4.9 and 4.10 respectively).

Projects	**UCW**	**AW**	**TCF**	**EF**	**Actual**
Project 1	0.0047	0.00006	0.0000104	0.0000092	0.10043
Project 2	0.00545	0.00048	0.0000103	0.0000087	0.13933
Project 3	0.0006	0.00005	0.0000085	0.00083	0.00294
Project 4	0.0004	0.00009	0.0000085	0.00083	0.00298
Project 6	0.00055	0.00007	0.0000085	0.00083	0.00371
Project 7	0.0004	0.00007	0.0000085	0.00083	0.0042
Project 8	0.0008	0.00007	0.0000085	0.00083	0.0058
Project 9	0.00025	0.00009	0.0000085	0.00083	0.00243
Project 10	0.00055	0.00008	0.0000085	0.00083	0.00595
Project 12	0.0006	0.00009	0.0000085	0.00083	0.0049
Project 13	0.001	0.0001	0.00001	0.0000097	0.0215
Project 14	0.005	0.0001	0.00001	0.0000117	0.125
Project 5	0.0004	0.00009	0.0000085	0.00083	0.00232
Project 11	0.0004	0.00007	0.0000085	0.00083	0.00578
Project 15	0.0015	0.0001	0.00001	0.0000117	0.054

Table 4.9 Normalized data set used for training and testing in *leave-one-out* method with unfuzzified UCW

Projects	Fuzzy-UCW	AW	TCF	EF	Actual
Project 1	0.005472222	0.00006	0.0000104	0.0000092	0.10043
Project 2	0.006625	0.00048	0.0000103	0.0000087	0.13933
Project 3	0.000455556	0.00005	0.0000085	0.00083	0.00294
Project 4	0.000416667	0.00009	0.0000085	0.00083	0.00298
Project 6	0.000469444	0.00007	0.0000085	0.00083	0.00371
Project 7	0.000380556	0.00007	0.0000085	0.00083	0.0042
Project 8	0.000766667	0.00007	0.0000085	0.00083	0.0058
Project 9	0.000241667	0.00009	0.0000085	0.00083	0.00243
Project 10	0.000572222	0.00008	0.0000085	0.00083	0.00595
Project 12	0.000591667	0.00009	0.0000085	0.00083	0.0049
Project 13	0.000991667	0.0001	0.00001	0.0000097	0.0215
Project 14	0.004886111	0.0001	0.00001	0.0000117	0.125
Project 5	0.000366667	0.00009	0.0000085	0.00083	0.00232
Project 11	0.000472222	0.00007	0.0000085	0.00083	0.00578
Project 15	0.001522222	0.0001	0.00001	0.0000117	0.054

Table 4.10 Normalized data set used for training and testing in leave-one-out method with fuzzified UCW

After preparing and pre-processing the data, we built, trained, and tested the network model. All these processes are carried out using Matlab's Neural Network Toolbox.

4.2.2. Building the Model

The modeling process involves selecting the neural network type and building the model by setting the parameters. For software development effort estimation model, a fully connected three layer feed forward neural network was proposed and built. Determining the number of neurons in the hidden layer is one important decision in building the network model. If they are very few, it will not be flexible enough to model the data; if

they are too many, the model will over fit the data. A common practice in neural network experiment suggests that the number of neurons in the hidden layer should be determined by using a trail and error method [9]. This means that, by following the trail and error method, the number of processing elements that will lead to the lowest error in the prediction of output versus actual output, should be settled as the final number. Following this method, an experiment was run with alternative numbers (numbers from 1 to 10) and then the one with 7 neurons was found to be the best prediction model (see table 4.11). The range from 1 to 10 was used, because it is recommended that the number of neurons in the hidden layer should fall somewhere between the number of input variables and the number of outputs in the data set. The structure of the designed network is shown in Figure 4.1.

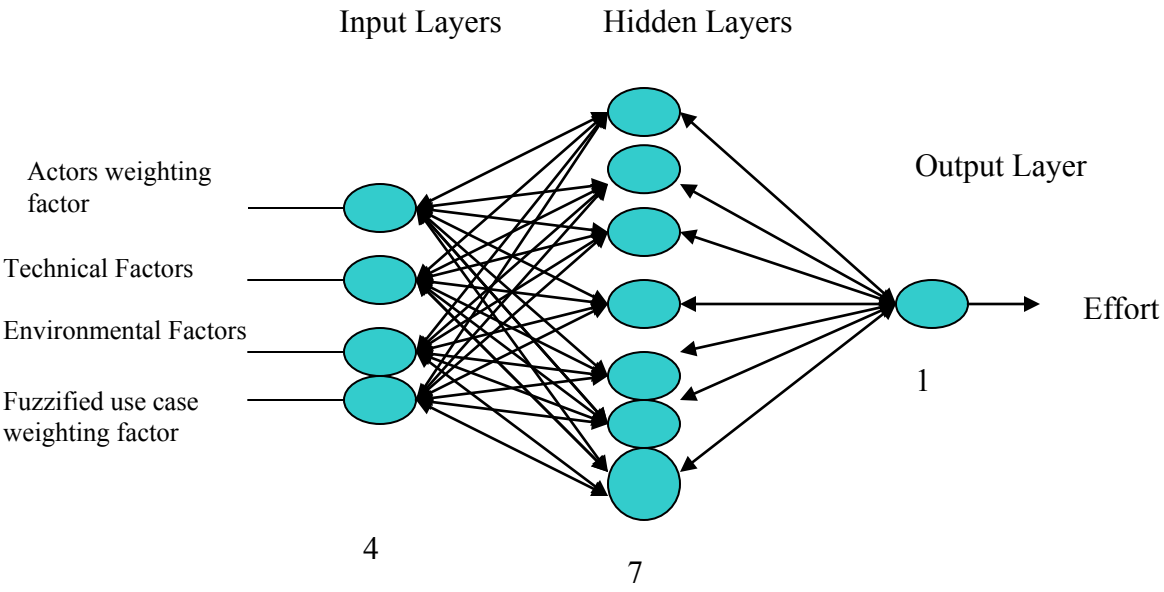

Figure 4.1 The Neural network Architecture

In both the hidden and output layers, the sigmoid function is used as the activation function. Network weights and biases are initialized randomly. The epoch limit used in this experiment was 10000.

Network No.	Number of hidden units	Performance error (MSE) in UCBEM	Performance error (MSE) in Fuzzy-UCBEM
1	1	0.000794723	0.000812877
2	2	0.000126995	0.000163923
3	3	0.000688152	0.0000895252
4	4	0.0000118358	0.0000134342
5	5	0.000153409	0.0000572389
6	6	0.000049027	0.0000321704
7	7	0.0000101808	0.0000054059
8	8	0.0000544377	0.000010997
9	9	0.0000862509	0.000053296
10	10	0.000173085	0.000229305

Table 4.11 Performance of neural network model with variable hidden units

4.2.3. Statistical evaluation of the model

Evaluating a neural network in terms of some statistical factors is important to show the performance of the network. According to Flexer [7], it is by no means justified to report just the best result of the multiple runs of experiment. Instead, at least the mean of the performance measures over all those runs and the corresponding variance should be reported. Taking this into account, mean and variance of the performance measures of the designed network were computed for one hundred runs. The multiple runs were computed varying the sequence of input data. The mean shows the normality of the distribution of measures, and the variance shows the generalization performance of the network model.

The computed mean and variance were 0.000 02126 and 0.00000000016 respectively. The result shows that performance errors were normally distributed among all runs because the mean is not so far from the result reported for best performance, which is

0.0000054. The result also shows that the network has very good generalization performance because the value of the variance is very small.

4.2.4. Training the Model

The training of a neural network is the process by which the neural network is presented with actual data from the process. It uses this to find the most appropriate set of weights for each connection.

As it was proposed, *back-propagation* learning algorithm was used for training the network. The implementation of *back-propagation* learning algorithm updates the network weights and biases in a direction that the performance function decreases most rapidly - the negative of the gradient. This means that the weights and biases are updated in the direction of the negative gradient of the performance function. The back propagation algorithm can be based on *batch gradient descent* or batch *gradient descent with momentum*. In the MATLAB's neural network toolbox, the steepest descent training function is *traingd*. There are seven training parameters associated with *traingd*: *epochs*, *show*, *goal*, *time, min_grad*, *max_fail*, and *lr*. The learning rate *lr* is the rate at which the neural network learns. If the learning rate is made too large, the algorithm becomes unstable. If the learning rate is set too small, the algorithm takes a long time to converge. The training status is displayed for every *show* iteration of the algorithm. The other parameters determine when the training stops. The training stops if the number of iterations exceeds epochs, the performance function drops below goal, the magnitude of the gradient is less than minimum gradient, or if the training time is longer than time seconds. Momentum can be added to back propagation learning by making weight changes equal to the sum of a fraction of the last weight change and the new change suggested by the back propagation rule. A momentum constant, *mc*, which can be any number between 0 and 1, mediates the magnitude of the effect that the last weight change allowed to have. When the momentum constant is 0, a weight change is based solely on the gradient. When the momentum constant is 1, the new weight change is set to equal the last weight change and the gradient is simply ignored. The gradient is computed by

summing the gradients calculated at each training example, and the weights and biases are only updated after all training examples have been presented. If the new performance function on a given iteration exceeds the performance function on a previous iteration by more than a predefined ratio *max_perf_inc*, the new weights and biases are discarded, and the momentum coefficient *mc* is set to zero. The steepest descent training function with momentum in the neural network toolbox is *traingdm*. It has the same parameters with *traingd* with one additional parameter, i.e., momentum constant (*mc*).

The network with 7 hidden units was trained using both *traingd* and *traingdm* and the result shows that, applying *traingd* introduces less mean squared error (see table 4.12).

Training Function	**Performance error (MSE) in UCBEM**	**Performance error (MSE) in Fuzzy-UCBEM**
Traingd	0.0000101808	0.0000054059
Traingdm	0.0000774829	0.0000393559

Table 4.12 Performance of neural network model with *traingd* and *traingdm*

Using *traingd* back propagation learning algorithm, successive training were performed for different combinations of learning rate. The training process stops upon hitting the error tolerance, or reaching the epoch's limits. The following table (table 4.13) shows the performance of the neural network model with different learning rates.

Learning rate	Performance error (MSE) in UCBEM	Performance error (MSE in Fuzzy-UCBEM)
1.0	0.00130962	0.00573099
0.9	0.00000676729	0.000000970099
0.8	0.000000486227	0.000000573023
0.7	0.000000451983	0.000000517843
0.6	0.00000425146	0.000000473927
0.5	0.000000404466	0.000000439503
0.4	0.00000038898	0.000000365034
0.3	0.00000211475	0.00000034828
0.2	0.00000112837	0.00000033754
0.1	0.00000901719	0.0000003323

Table 4.13 Performance of NN with variable learning rates.

The above table shows that best learning rate for UCBEM is 0.4 and is 0.1 for Fuzzy-UCBEM. The performance error is monotonically decreasing for fuzzy-UCBEM. This may imply that smallest learning rate is appropriate for fuzzy-UCBEM, but this may not be true for all cases .It depends on the nature of data sets used.

4.2.5. Testing the Model

Testing was carried out using the two methods stated earlier in section 4.2.1, i.e., *holdout* and *leave-one-out*. The results and process of testing with the two methods are presented below.

4.2.5.1. Testing with holdout method

Applying the *holdout* method, tests were carried out with three project data, which were selected randomly from the fifteen project data. As mentioned earlier, the data in the test data file are not represented in the training data set. The test data were specified in the

input/output parameters of the model. The network with best performance, i.e., 7 hidden units with 0.4 and 0.1 learning rate for Neuro-UCBEM and Neuro-Fuzzy-UCBEM respectively, were tested. Test results for the *holdout* method are shown in table 4.14 (for UCBEM) and table 4.15(for Fuzzy-UCBEM).

Tests with different data	**Predicted Output**	**Actual output**	**Errors (Actual-Predicted)**
Test1	375.58	232	-143.58
Test2	418.67	578	159.33
Test3	3260	5400	2140

Table 4.14 Test result of Neuro-UCBEM with *holdout* method.

Tests with different data	**Predicted Output**	**Actual output**	**Errors (Actual-Predicted)**
Test1	241.04	232	-9.04
Test2	600.92	578	-22.92
Test3	3565.6	5400	1834.4

Table 4.15 Test result of Neuro-Fuzzy-UCBEM *holdout* method.

4.2.5.2. Testing with leave-one-out method

The *leave-one-out* method enables to use the overall data for both training and testing in a recursive manner. This means one data-point (data of one project) will be taken out from the data set for testing while the rests are used for training. A data-point could be used for training and testing at different times but not concurrently. For example, to run a first test in our case, data of project1 was used for testing, while data of the other fourteen projects were used for training. For a second test, data of project2 was used for testing, while data of the other fourteen projects (including data of project1 that was used for testing in the previous case) were used for training, and the process was carried out for all project data. In the process of applying this method, fifteen input/output files were prepared for

training and another fifteen for testing. The required data files were prepared for both Neuro-UCBEM and Neuro-Fuzzy-UCBEM .

The following tables show test results in *holdout* method. Table 4.16 shows test result of Neuro-UCBEM, and table 4.17 shows that of Neuro-Fuzzy UCBEM

Tests with different data	**Predicted**	**Actual**	**Errors (Actual-Predicted)**
Test1	11747	10043	-1703.5
Test2	12803	13933	1129.9
Test3	204.18	294	89.821
Test4	280.87	298	17.13
Test5	279.79	232	47.793
Test6	458.24	371	-87.241
Test7	455.26	420	-35.258
Test8	458.24	580	88.142
Test9	290.32	243	-47.325
Test10	652.34	595	-57.336
Test11	423	578	154.85
Test12	437.95	490	52.055
Test13	1678.5	2150	471.5
Test14	13992	12500	1492.2
Test15	4670	5400	730.02

Table 4.16 Test result of Neuro-UCBEM with *leave-one-out* method

Tests with different data	Predicted	Actual	Errors (Actual-Predicted)
Test1	10013	10043	29.842
Test2	13948	13933	-14.902
Test3	226.72	294	67.782
Test4	316.04	298	-18.045
Test5	256.47	232	-24.467
Test6	393.69	371	-22.695
Test7	411.62	420	83.762
Test8	585	580	-50.03
Test9	253.82	243	-10.822
Test10	607.15	595	-12.152
Test11	472	578	95
Test12	505.16	490	-15.158
Test13	2110.3	2150	39.727
Test14	12703	12500	-203.22
Test15	4714	5400	-686.67

Table 4.17 Test result of Neuro-Fuzzy-UCBEM with *leave-one-out* method

4.2.6. Test Result Analysis.

Test results obtained by both *holdout* and *leave-one-out* methods were evaluated in terms of Mean Magnitude of Relative Error (MMRE) and PRED (.25). These two statistical methods enable to validate the models with respect to their fitting accuracy.

The following table (table 4.18) shows the accuracy of the neural network model with *holdout* method.

	NeuroFuzzy-UCBEM	**NeuroUCBEM**
MMRE	0.13	0.42
Pred(0.25)	0.66	0

Table 4.18 Accuracy of NN with *holdout* method

As shown above the result of Neuro-Fuzzy-UCBEM in terms of MMRE is 0.13. Since it is less than 0.25, it indicates that the model is acceptable for software development effort estimation. On the other hand, the MMRE of Neuro-UCBEM is 0.42, which is greater than 0.25. This indicates that Neuro-UCBEM is not acceptable as a good model. In terms of Pred(0.25) again the Neuro-Fuzzy-UCBEM shows better accurate result. The Pred(0.25) of the NeuroFuzzy-UCBEM is 0.66 and that of Neuro-UCBEM is 0. As described in section 4.1.2, a standard criterion for considering a model to be accepted as a good estimation model is Pred(0.25) ≥ 0.75. But none of the two models fulfill these criteria. Considering the size of the datasets used for testing (which are only three), it is difficult to generalize that the accuracy of models are poor in terms of Pred(0,25). Especially for Neuro-Fuzzy-UCBEM, 0.66 is good result. This means that two of the three estimates are within the range of 25% of the actual.

The accuracy of the models in terms of both MMRE and Pred(.25) were improved when *leave-one-out* method is applied for training and testing the network. The result obtained by applying the *leave-one-out* method is shown in table 4.19.

	NeuroFuzzy-UCBEM	Neuro-UCBEM
MMRE	0.06	0.16
PRED(0.25)	1	0.86

Table 4.19 Accuracy of NN with *leave-one-out* method

As shown in the above table, both Neuro-Fuzzy-UCBEM and Neuro-UCBEM have a MMRE<0.25. This indicates that both of them are acceptable, but Neuro-Fuzzy-UCBEM

is more acceptable because its MMRE is less than that of Neuro-UCBEM. In terms of Pred(0.25), the result of Neuro-Fuzzy-UCBEM is 1 and that of Neuro-UCBEM is 0.86. This means, 100% of the estimates are within the range of 25% of the actual values in Neuro-Fuzzy-UCBEM, and it is 86% in Neuro-UCBEM. Since the minimum requirement in Pred(.25) is 0.75, both models are acceptable for software development effort estimation.

In terms of both MMRE and Pred(.25), the Neuro-Fuzzy-UCBEM outperforms the Neuro-UCBEM. This indicates that fuzzyfing the use case complexity-weighting factor has an impact on the performance of the neural network. It also indicates that the performance of a neural network is affected not only by the size of the training data set but also by the nature of input-output data used for training.

In general, the accuracy result obtained by applying *leave-one-out* method is encouraging and the method seems more appropriate than the *holdout* method, especially for small data set size. Therefore, discussions and conclusions are made based on the result obtained by the *leave-one-out* method.

5. Discussion

This work has attempted to explore the potential of soft computing approaches for software development effort estimation by applying fuzzy logic and neural network to use case based effort estimation. The results of the experiments are shown in the previous chapter. This chapter discusses the result of the experiments from different perspectives: accuracy, consistency and other related issues.

5.1. Accuracy of the Models

Performance of a model for software development effort estimation is basically evaluated by its level of accuracy to estimate effort/cost. In some cases, however, the accuracy requirements for a model may be low when compared to other goals, and these other motivations may be superseding, especially where the improvements in accuracy are small. It is also important to specify pre-analysis what is meant by accuracy for a particular modeling task. In some cases, absolute errors are of concern, while in others relative errors are more important. Also, a choice must be made between the relative importance of threshold-based errors (like Pred(*l*)) and average errors. These issues are certainly not trivial ones, to be faced with the same solution for all projects. In this study emphasis is placed on MMRE and Pred(*l*), which are considered as a common accuracy indicators in many research works on effort/cost estimation [8, 18, 19].

The results are summarized in Table 5.1 and plotted in figure 4.2 and 4.3, showing the MMRE, Pred(0.25) and correlation measures achieved under the different methods.

	Fuzzy-UCBEM	**UCBEM**	**NeuroFuzzy-UCBEM**	**NeuroUCBEM**
MMRE	0.13	0.18	0.06	0.16
Pred(0.25)	0.8	0.73	1	0.86
Correl	0.994424	0.994792	0.999242	0.990766

Table 5.1 Summary of results

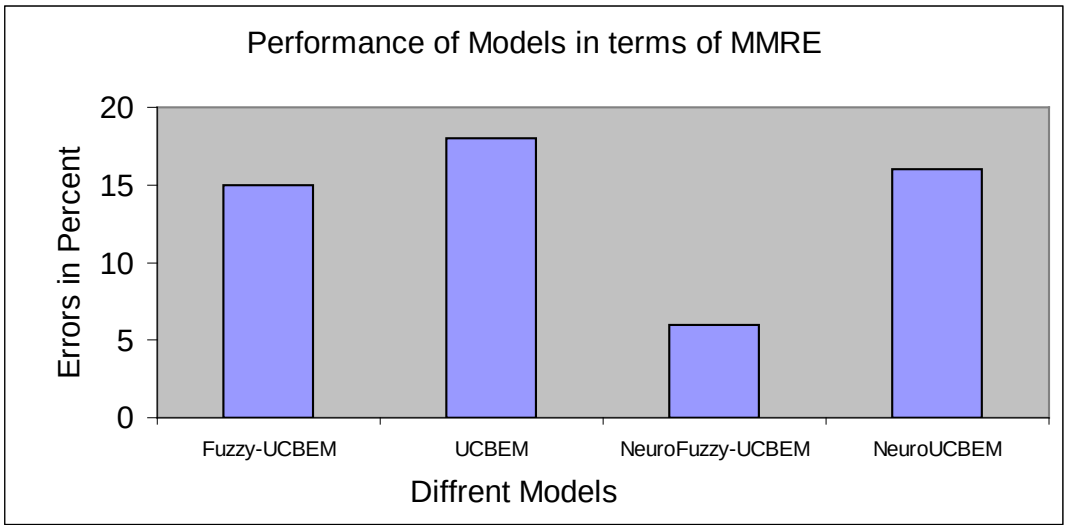

Figure 5.1 Performance of models in terms of MMRE

The model with minimum errors (MMRE) is the best accurate model. According to the above figure, the best one is Neuro-Fuzzy-UCBEM.

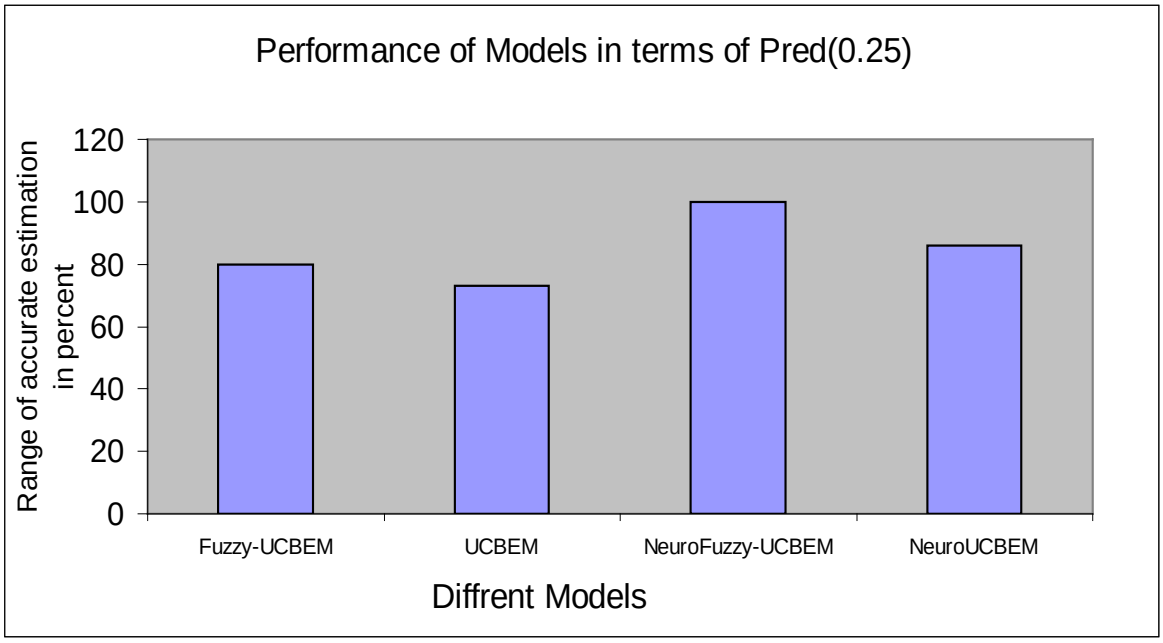

Figure 5.2 Performance of models in terms of Pred(0.25)

In this figure, the model with the highest percentage is the best accurate model. Therefore, the best model according to this figure is again Neuro-Fuzzy-UCBEM.

Examination of the results in terms of both MMRE and Pred(0.25) shows that the NeuroFuzzy-UCBEM technique appears to perform best followed by Fuzzy-UCBEM and Neuro-UCBEM and UCBEM. This result indicates that the neural network model performs best when it is used with fuzzy use case complexity weighting factor inputs, and it is slightly better than UCBEM when it is used without fuzzyfing inputs. This shows that the performance of the neural network model is affected by the values of data in the input vectors (not only in the quantity of input dataset).

In general, the levels of accuracy obtained by applying soft computing methods are encouraging. Based on the above results, we can say that soft computing approaches have a potential to estimate software development effort with reasonable level of accuracy.

5.2. Consistency of the models

The consistency of the estimation method was measured by the correlation (Correl) between the estimates and the actuals. A consistent method should have a high positive correlation between actuals and estimates. Examination of the results in terms of Correl shows that there is 99% correlation between actuals and estimate in all the techniques. There is no significance difference among the techniques. This means that all the techniques are consistent in estimating software development effort.

5.3. Benefits

In addition to accuracy and consistency, the neuro-fuzzy model applied here offers different advantages for software development effort estimation:.

1. It deals with uncertainty and learns from experience (data). The fuzzy logic removes the uncertainty in assigning use case complexity weighting factor, and the neural network enables the model to learn from existing previous project data. These issues were major problems in other cost estimation models.
2. It can be applied at the early stage of software development process. It can be used after requirements are specified in terms of use cases and other technical and complexity factors are identified. This process could be done at requirement specification stage, which is relatively early stage of software development process.
3. It can be applied for any project type. As far as project requirements are specified using use cases and technical & complexity factors are properly identified, the neuro-fuzzy model can be used to estimate effort in any project type of software development. This was also demonstrated by the case studies conducted. The data used for the experiment were of various project types, ranging from management information system to real time system.

5.4. Limitations

The limitations or constraints of any estimation model should be understood before it is applied to practical situations. The neuro-fuzzy model has some limitations:

1. Final results which are difficult to interpret are obtained. Other than accuracy, there is the issue of interpretability of the models in their final form. There are limited opportunities to view the form of the model using a black box technique such as a neural network. While the weights are available, it is difficult to then interpret these in a meaningful manner.
2. The neural network model may not allow replicating results for all cases of estimation. Training a neural network is an algorithmic procedure and the results can most certainly be replicated as long as one uses identical code, the same initial weights, the same training data, and the same deterministic method of presenting the data during training. However, if even one of these parameters is altered, the resulting neural network would almost certainly be different from the original one. This difference is apt to be extremely minor, however it is not inconceivable that major differences could occur.

5.5. Selecting the Optimal Model

In this thesis, estimation models like UCBEM, Fuzzy-UCBEM, Neuro-Fuzzy-UCBEM and Neuro-UCBEM have been described and tested on case study data sets. It is found out that the use of the most appropriate model, Neuro-Fuzzyy-UCBEM, can lead to more accurate results; however, this should not be the only criterion used for selecting a model. Such factors as consistency and interpretability need to be considered in conjunction. For a particular project, the relative importance of each should be assessed, along with the advantages/disadvantages of each model when compared to the others. In many cases, the best solution is to not select any one model, but rather to estimate using two or more methods. Therefore, one can use the above-mentioned models separately to estimate software development effort and select the one which seems reasonable.

6. Implementation of the Neuro-Fuzzy UCBEM

In order to demonstrate the automation of the neuro-fuzzy UCBEM approach, a prototype tool is developed. The prototype tool incorporates different packages (modules) that are designed to provide different services. This chapter describes the design and implementation of the prototype tool for the neuro-fuzzy UCBEM. Section 6.1 describes the environment under which the prototype is implemented and section 6.2 presents the architecture defined to automate the effort estimation process based on fuzzy logic and neural networks. Also, a description of the components/sub-systems of the architecture is given in this section.

6.1. Description of Environment

The system is developed using MATLAB. MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. MATLAB features a family of add-on application-specific solutions called toolboxes. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others. Therefore, MATLAB was suitable to implement our system especially for the back-propagation algorithm.

6.2. Architecture

The system implements the neuro-fuzzy technique. The neuro-fuzzy technique uses various parameters like fuzzyfied use case weights, actor weights, and environmental and technical factors as input for a neural network. The neural network is then used to predict an effort for a new project. For the purpose of this prototype, the neuro-fuzzy UCBEM

system was sub divided into four different sub systems. The framework of the prototype is shown in Figure 6.1.

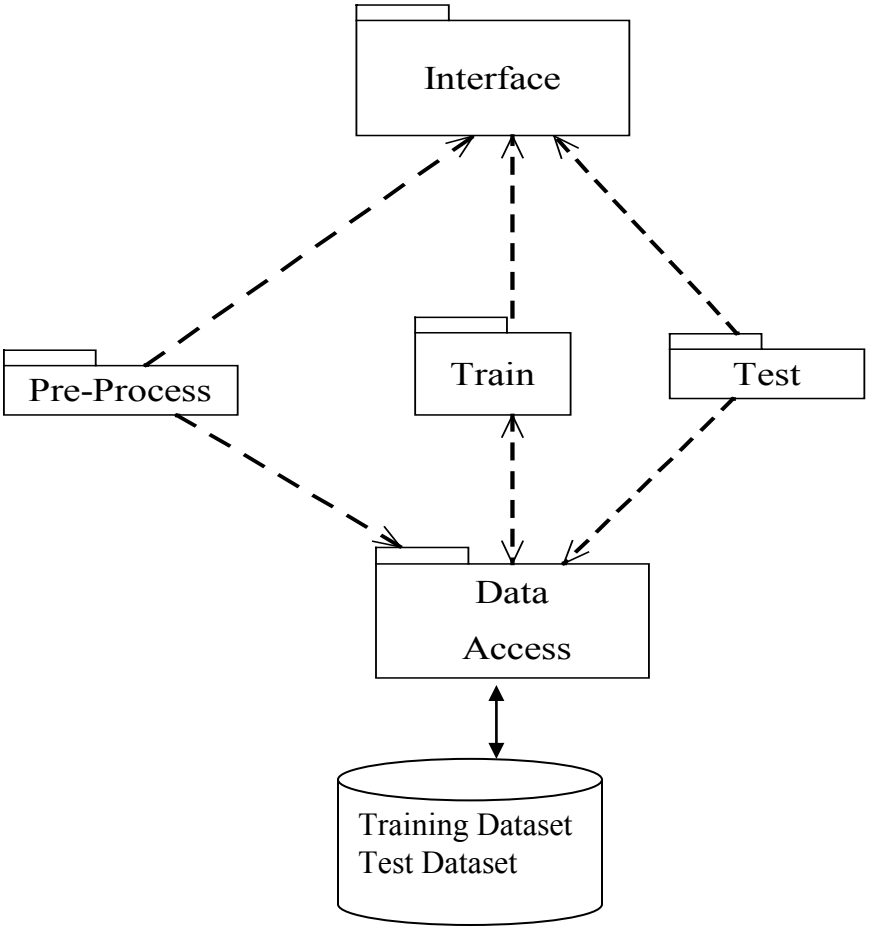

Figure 6.1 System Architecture of the Neuro-fuzzy UCBEM Framework

As shown in the above figure, there are four sub systems namely *Pre-Process*, *Training*, *Test* and *Data Access.* A brief discussion regarding the sub systems is provided below:

1. **The Pre-Process Subsystem**. This subsystem is used to process the data entered through the interface. The processes performed by this subsystem include:
    - Fuzzyfing use case complexity weighting factors
    - Calculating total use case weights
    - Calculating total actor's weight

- Calculating technical and environmental factors
- Normalization of calculated fields
- Writing the processed data into a file.

2. **Train Subsystem**. This subsystem is responsible for implementing the neural network. It implements a three-layer architecture with seven hidden units, back-propagation learning algorithm, and *sigmoid* activation function. It reads pre-processed input/output data for training and trains the network with the data.
3. **Test Subsystem.** This subsystem is responsible for testing the performance of the network. It reads processed data (cost drivers) from file and predicts an effort for a new project based on the trained network
4. **Data Access** Subsystem**.** This subsystem is responsible for reading and writing data. It provides a read - write service for other sub systems.

6.3. Class Diagram

Three major classes namely Use case, Network and File were identified for this prototype. The classes with their services and relationships are shown in Figure 6.2.

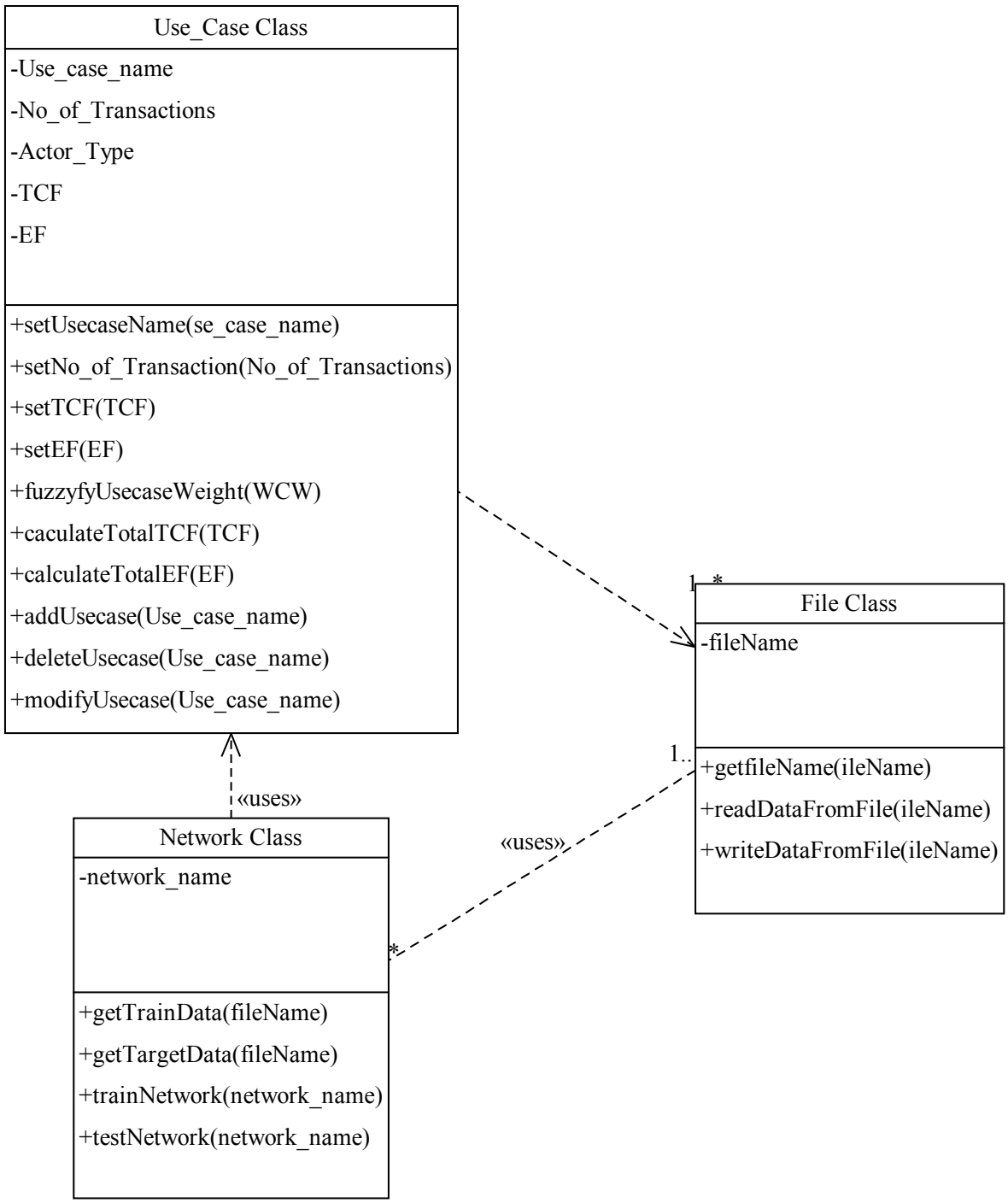

Figure 6.2 Class Diagram for the Neuro-fuzzy UCBEM prototype

6.4. Interface Design

The user interface enables the user to interact with the system. Three major forms are designed for the system.

1. **Main page**. The screenshot of the main page is shown in Figure 6.3.

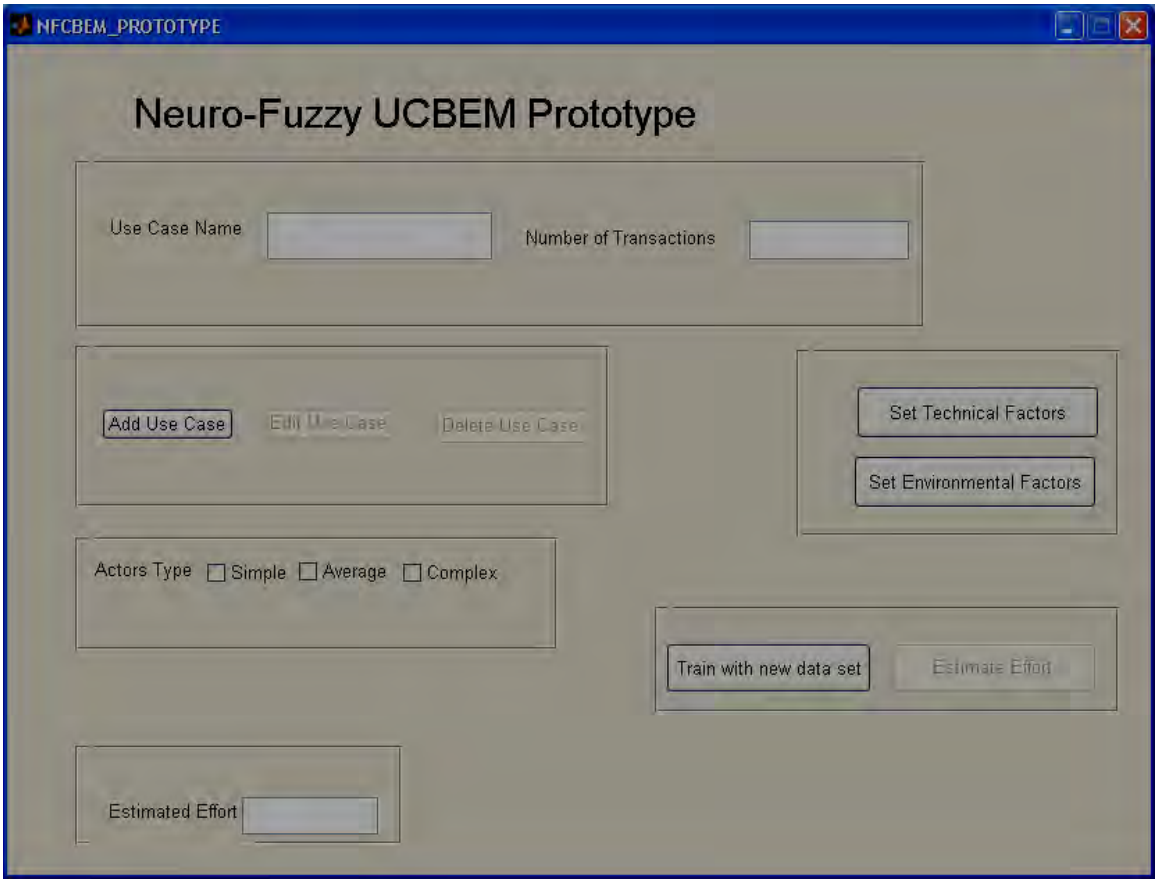

Figure 6.3 Main Page of the prototype

2. TCF form. This form is used to enter the weights of the technical complexity factors, which are identified in use case based estimation model. The screenshot of this form is shown in Figure 6.4.

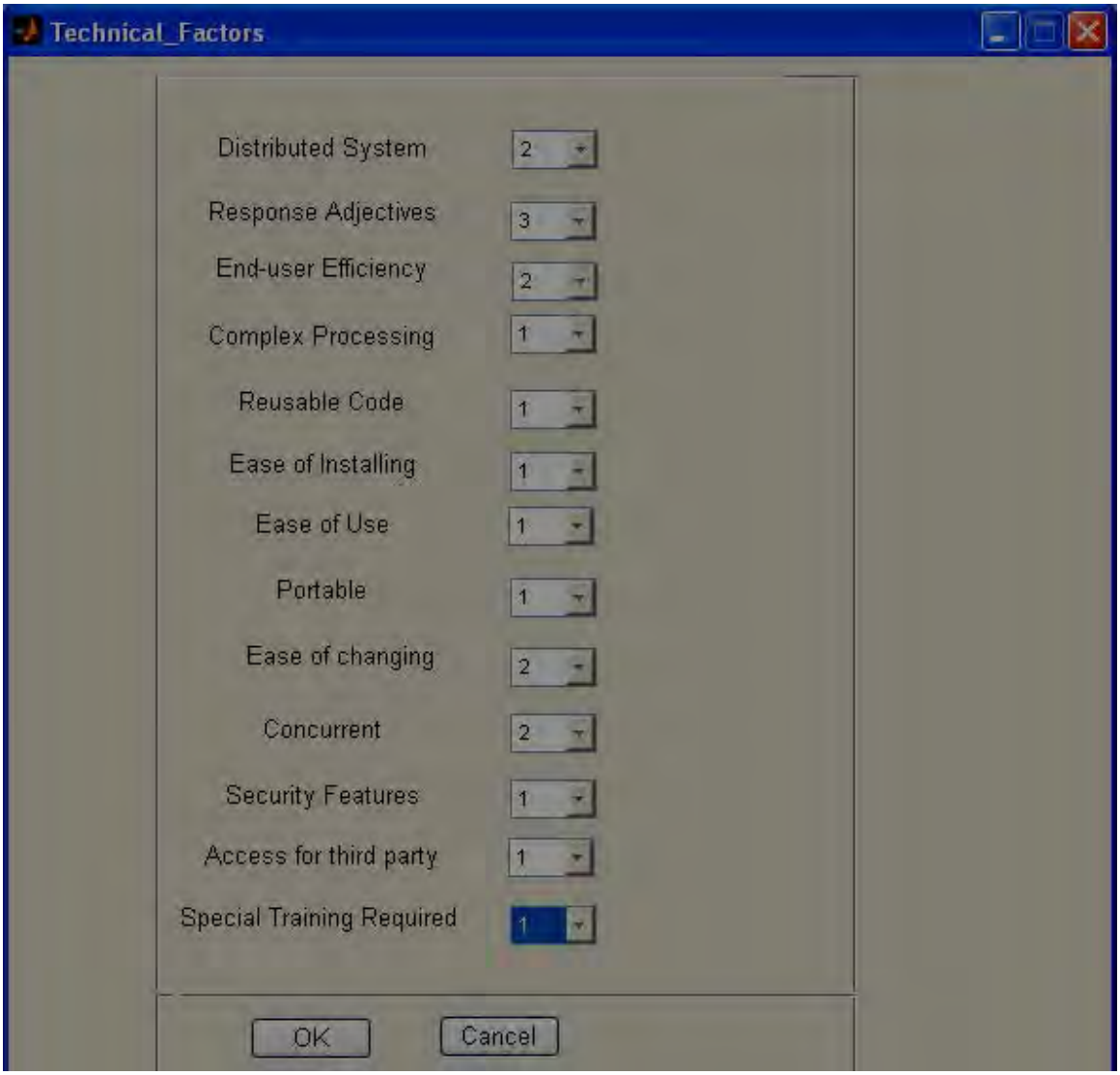

Figure 6.4 The TCF form

3. EF form. This form is used to enter the weights of the environmental factors, which are identified in use case based estimation model. The screenshot of this form is shown in Figure 6.5.

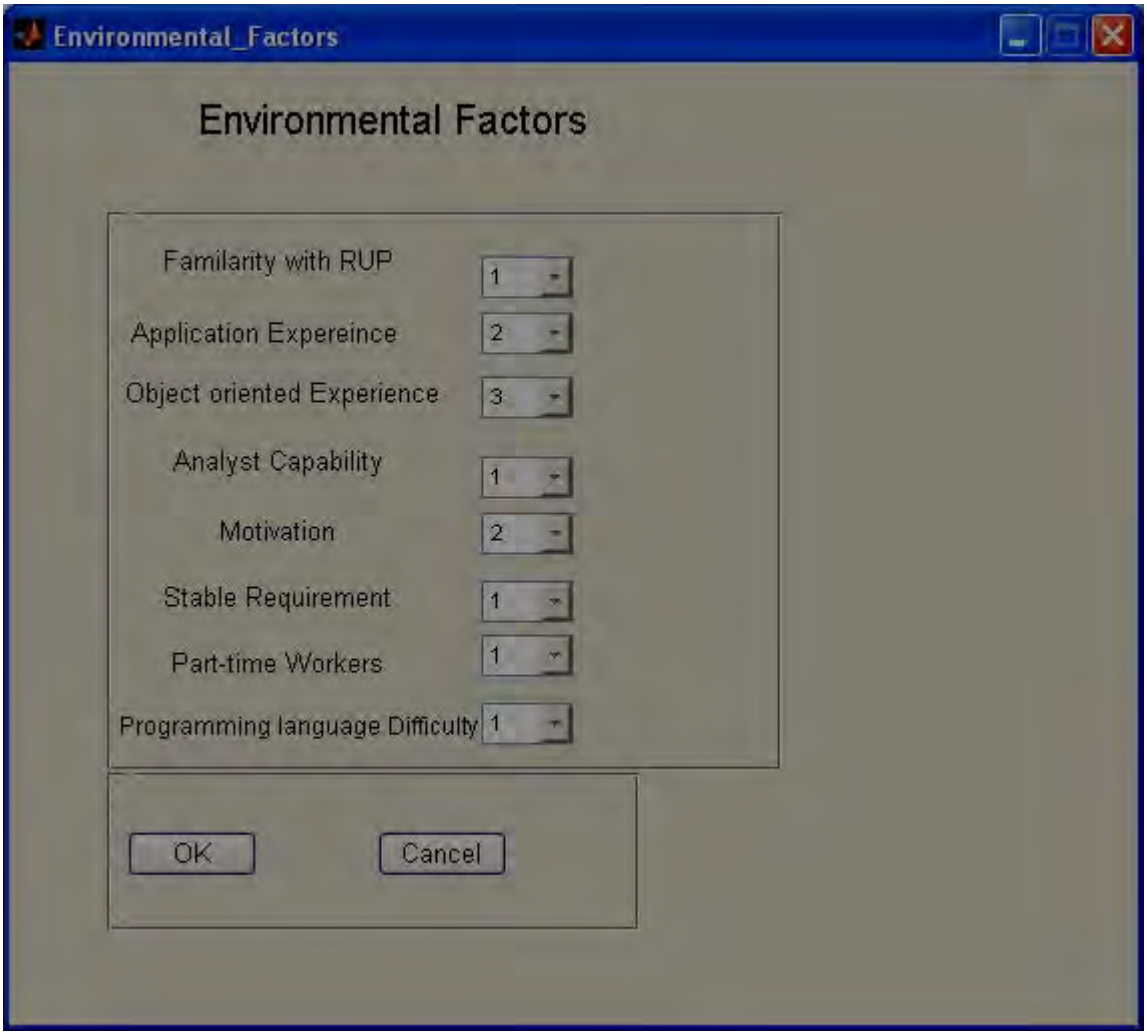

Figure 6.5 The EF form

7. Conclusions

Reasonable prediction of software development effort is a critical factor in making good management decisions in software project management. But it is very challenging and full of uncertainties. Many estimation models have been proposed over the last 30 years, but there is no estimation model that deal with imprecision, uncertainty, and learning from previous situations. Also, approaches developed to address these issues are not satisfactory.

The purpose of this thesis was to explore and apply soft computing techniques that seem promising for handling uncertainties and other issues. A Neuro-fuzzy approach that incorporates two main soft computing methods, i.e., neural network and fuzzy logic was proposed and applied to use case based estimation model. The result shows that these soft computing techniques have an encouraging potential to estimate software development effort.

7.1 Main Contributions

The main contributions of this work are summarized as follows:

1. introduced a neuro-fuzzy technique to software development effort estimation
2. extended use case based estimation model by fuzzyfing use case complexity weighting factors. It is also shown that this process slightly improves the accuracy of the use case based estimation model.
3. designed and implemented a neural network model for software development effort estimation.

As can be seen from this work, applying neuro-fuzzy technique improves the accuracy of use case based estimation model. This indicates the potential of soft computing approaches in estimating software development effort with reasonable accuracy.

7.2 Future Works.

Though several activities are undertaken by this research work, we believe that the potential of soft computing methods could be further explored and one can come up with better results by extending this work. The possible extensions of this work are described as follows:

1. The first and obvious extension of this work is training and testing the neuro-fuzzy approach with large and more reliable data sets.
2. Fuzzification of other cost drivers like actor's weighting factor, assignment of technical and environmental factors is also a possible extension.
3. The neural network model designed in this work assigns weights to each input just randomly. Applying fuzzy logic to determine weight for each input unit in the neural network model is a possible extension of this work. This may help to come up with a better stable and accurate model. It also helps to interpret decisions made by the network.
4. It is also possible to check the performance of the neural network model with other learning algorithms like unsupervised learning.

References:

[1] Bente Anda, Hege Dreiem, Magne Jorgensen, and Dag Sjoberg. "Estimating Software Development Effort based on Use Cases: Experience from Industry". *The Unified Modeling Language.* Springer-*Verlag. 4th International Conference*, Toronto, Canada, October 1-5, 2001, LNCS 218, 2001.

[2] Bmdwin J., Martine B.Azvine T., " Soft Computing for Intelligent Knowledge Based Systems", *BT Techno J* vol.16 No.3 July 1998, pp.165-179

[3] Boehm B., "Software Engineering Economics", Prentice-Hall, 1981

[4] Boem B., Abts C., "Software Development Cost Estimation Approaches : A Survey", Ph.D thesis extension unpublished

[5] Boetticher G., "An Assessment of Metric Contribution in the Construction of a Neural Network-Based Effort Estimator," *Second Int. Workshop on Soft Computing Applied to Soft. Engineering*, 2001.

[6] Edmond VanDoren. *Software Technology Review: Function Point Analysis* at http://www.sei.cmu.edu/activities/str/descriptions/fpa_body.html.

[7] F lexer, A . " Statstical E valuation o f N eural N etwork E xperiments: M inimum Requirements an d C urrent P ractice", *The Austrian Research Institute for Artificial Intelligence Technical Report*, 1995

[8] Gray, A. MacDonell, S. "Software Metrics Data Analysis - Exploring the Relative Performance of S ome C ommonly U sed M odeling T echniques" *The Information Science Discussion Paper Series* no. 99/11,1999.

[9] Hykin S., "Neural Networks: A comprehensive Foundation", 2nd ed. Printce Hall, 1999

[10] I dri A ., K jiri L ., and A bran A., " I nvestigating S oft C omputing i n C ase-based Reasoning f or s oftware Cost E stimation", *Engineering Intelligent Systems*, vol .3, 2002, pp. 147-157.

[11] I dri A ., K jiri L ., a nd A bran A ., " COCOMO C ost M odel U sing F uzzy L ogic", *In Proceedings of the 7th International Conference on Fuzzy Theory and Technology, Atlantic City, NJ, February 2000, pp.219-223.*

[12] Idri A., Abran A., and Khoshgoftaar T., "Fuzzy Analogy: A new Approach for Software Cost Estimation", *In Proceedings of the 11th International Workshop on Software Measurements* 2001, pp. 93-101, Montreal, Canada.

[13] Idri A., Khoshgoftaar T., Abran A., Robert S. "Can Neural Networks be easily Interpreted in Software Cost Estimation?", *Fuzz-IEEE*, 12-17 May, Hawai, 2002, pp. 1162-1166

[14] Jang R., C.Sun, E. Mizutani Neuro-Fuzzy and Soft Computing, Printce Hall, 1997

[15] Junior O., Farias P., Belchior A., " A Fuzzy Model for Function Point Analysis to Development and Enhancement Project Assessment", CLEI Electronic Journal 5(2), 1999

[16] Kadoda G., Cartwright M., Chen L., and Shepperd M., "Experiences Using Case-Based Reasoning to Predict Software Project Effort", In *Proceedings of EASE*, p.23-28, Keele, 2000, UK.

[17] Karner G. "Metrics for Objectory". Diploma thesis, University of Linköping, Sweden. No. LiTHIDA-Ex-9344:21. December 1993.

[18] Kitchenham, B [et al]. "Assessing Prediction Systems",*The Information Science Discussion Paper Series* no. 99/14,1999

[19] Lionel C. [et al]. "An Assessment and Comparison of Common Software Cost Estimation Modeling Techniques", *International Software Engineering Research Network Technical Report* ISERN-98-27

[20] Pedrycz W., Peters J., Romanna S., " A Fuzzy Set Approach to Cost Estimation of SoftwareProjects", *In Proceedings of the 1999 IEEE Conference on Electrical and Computer Engineering.*

[21] Ribu, Kirsten. " Estimating Object-Oriented Software Projects with Use Cases" University of Oslo Department of Informatics, *Master of ScienceThesis*, 2001.

[22] Shepperd M. and Kadoda G. "Using Simulation to Evaluate Predictions Systems", In *Proceedings of the 7th International Symposium on Software Metrics*, .2001 pp. 349-358, England, UK, IEEE Computer Society.

[23] Shepperd M. and Schofield C., " Estimating Software Project Effort Using Analogies", *IEEE Transactions on Software Engineering*, vol. 23, no. 12, pp. 736-743, November 1997.

[24] Shepperd M., Schofield C., and Kitchenham B., "Effort Estimation using Analogy", In *Proceedings of the 18th International Conference on Software Engineering*, 1996, pp. 170-178, Berlin.

[25] Shukla K., "Neuro-Genetic Prediction of Software Development Effort", *Information SoftwareTechnology*, Vol. 42, 2000, pp. 701-713

Abbreviations

KLOC : Kilo Lines Of Code

COCOMO: COnstractive Cost MOdel

UCBEM : Use Case Based Estimation Model

NN: Neural Network

UUCW: Unadjusted Use Case Weight

UAW: Unadjusted Actor's Weight

UUPC: Unadjusted Use Case Point

TCF: Technical Complexity factors

EF: Environmental Factors

Appendix A

Data Sets Used for the Experiment

Industry Projects

Project A

Use Cases	Number of transaction	Membership to Simple	Membership to Average	Membership to Complex	Fuzzy Weight
UC1	3	0.555555556	0.44444444	0	7.22222222
UC2	2	1	0	0	5
UC3	1	1	0	0	5
UC4	5	0.111111111	0.88888889	0	9.44444444
UC5	4	0.333333333	0.66666667	0	8.33333333
UC6	6	0	0.83333333	0.166666667	10.8333333
UC7	3	0.555555556	0.44444444	0	7.22222222
UC8	7	0	0.5	0.5	12.5
UC9	5	0.111111111	0.88888889	0	9.44444444
UC10	6	0	0.83333333	0.166666667	10.8333333
UC11	6	0	0.83333333	0.166666667	10.8333333
UC12	5	0.111111111	0.88888889	0	9.44444444
UC13	4	0.333333333	0.66666667	0	8.33333333
UC14	5	0.111111111	0.88888889	0	9.44444444
UC15	3	0.555555556	0.44444444	0	7.22222222
UC16	1	1	0	0	5
UC17	5	0.111111111	0.88888889	0	9.44444444
UC18	2	1	0	0	5
UC19	2	1	0	0	5
UC20	4	0.333333333	0.66666667	0	8.33333333
UC21	5	0.111111111	0.88888889	0	9.44444444
UC22	4	0.333333333	0.66666667	0	8.33333333
UC23	6	0	0.83333333	0.166666667	10.8333333
UC24	4	0.333333333	0.66666667	0	8.33333333
UC25	6	0	0.83333333	0.166666667	10.8333333
UC26	1	1	0	0	5
UC27	8	0	0.16666667	0.833333333	14.1666667
UC28	5	0.111111111	0.88888889	0	9.44444444

UC29	6	0	0.83333333	0.166666667	10.8333333
UC30	1	1	0	0	5
UC31	9	0	0	1	15
UC32	2	1	0	0	5
UC33	1	1	0	0	5
UC34	4	0.333333333	0.66666667	0	8.33333333
UC35	4	0.333333333	0.66666667	0	8.33333333
UC36	5	0.111111111	0.88888889	0	9.44444444
UC37	6	0	0.83333333	0.166666667	10.8333333
UC38	3	0.555555556	0.44444444	0	7.22222222
UC39	4	0.333333333	0.66666667	0	8.33333333
UC40	4	0.333333333	0.66666667	0	8.33333333
UC41	9	0	0	1	15
UC42	5	0.111111111	0.88888889	0	9.44444444
UC43	1	1	0	0	5
UC44	6	0	0.83333333	0.166666667	10.8333333
UC45	4	0.333333333	0.66666667	0	8.33333333
UC46	6	0	0.83333333	0.166666667	10.8333333
UC47	5	0.111111111	0.88888889	0	9.44444444
UC48	6	0	0.83333333	0.166666667	10.8333333
UC49	3	0.555555556	0.44444444	0	7.22222222
UC50	4	0.333333333	0.66666667	0	8.33333333
UC51	5	0.111111111	0.88888889	0	9.44444444
UC52	5	0.111111111	0.88888889	0	9.44444444
UC53	4	0.333333333	0.66666667	0	8.33333333
UC54	2	1	0	0	5
UC55	5	0.111111111	0.88888889	0	9.44444444
UC56	4	0.333333333	0.66666667	0	8.33333333
UC57	6	0	0.83333333	0.166666667	10.8333333
UC58	3	0.555555556	0.44444444	0	7.22222222
UC59	5	0.111111111	0.88888889	0	9.44444444
UC60	4	0.333333333	0.66666667	0	8.33333333
UC61	2	1	0	0	5
UC62	6	0	0.83333333	0.166666667	10.8333333
UC63	4	0.333333333	0.66666667	0	8.33333333
	Total				547.222222

	Actors		Weights	Total
	Simple	0	1	0
	Medium	0	2	0
	Complex	2	3	6
			Total	6

Project B

Use Cases	Number of transaction	Membership to Simple	Membership to Average	Membership to Complex	Fuzzy Weight
UC1	3	0.555555556	0.44444444	0	7.22222222
UC2	3	0.555555556	0.44444444	0	7.22222222
UC3	3	0.555555556	0.44444444	0	7.22222222
UC4	7	0	0.5	0.5	12.5
UC5	7	0	0.5	0.5	12.5
UC6	3	0.555555556	0.44444444	0	7.22222222
UC7	9	0	0	1	15
UC8	3	0.555555556	0.44444444	0	7.22222222
UC9	7	0	0.5	0.5	12.5
UC10	3	0.555555556	0.44444444	0	7.22222222
UC11	3	0.555555556	0.44444444	0	7.22222222
UC12	7	0	0.5	0.5	12.5
UC13	9	0	0	1	15
UC14	3	0.555555556	0.44444444	0	7.22222222
UC15	7	0	0.5	0.5	12.5
UC16	3	0.555555556	0.44444444	0	7.22222222
UC17	6	0	0.83333333	0.166666667	10.8333333
UC18	7	0	0.5	0.5	12.5
UC19	3	0.555555556	0.44444444	0	7.22222222
UC20	7	0	0.5	0.5	12.5
UC21	9	0	0	1	15
UC22	7	0	0.5	0.5	12.5
UC23	3	0.555555556	0.44444444	0	7.22222222

UC24	6	0	0.83333333	0.166666667	10.8333333
UC25	9	0	0	1	15
UC26	3	0.555555556	0.44444444	0	7.22222222
UC27	7	0	0.5	0.5	12.5
UC28	3	0.555555556	0.44444444	0	7.22222222
UC29	7	0	0.5	0.5	12.5
UC30	9	0	0	1	15
UC31	3	0.555555556	0.44444444	0	7.22222222
UC32	3	0.555555556	0.44444444	0	7.22222222
UC33	6	0	0.83333333	0.166666667	10.8333333
UC34	8	0	0.16666667	0.833333333	14.1666667
UC35	3	0.555555556	0.44444444	0	7.22222222
UC36	7	0	0.5	0.5	12.5
UC37	3	0.555555556	0.44444444	0	7.22222222
UC38	3	0.555555556	0.44444444	0	7.22222222
UC39	9	0	0	1	15
UC40	9	0	0	1	15
UC41	6	0	0.83333333	0.166666667	10.8333333
UC42	3	0.555555556	0.44444444	0	7.22222222
UC43	9	0	0	1	15
UC44	3	0.555555556	0.44444444	0	7.22222222
UC45	9	0	0	1	15
UC46	7	0	0.5	0.5	12.5
UC47	3	0.555555556	0.44444444	0	7.22222222
UC48	9	0	0	1	15
UC49	3	0.555555556	0.44444444	0	7.22222222
UC50	3	0.555555556	0.44444444	0	7.22222222
UC51	7	0	0.5	0.5	12.5
UC52	9	0	0	1	15
UC53	3	0.555555556	0.44444444	0	7.22222222
UC54	7	0	0.5	0.5	12.5
UC55	3	0.555555556	0.44444444	0	7.22222222
UC56	6	0	0.83333333	0.166666667	10.8333333
UC57	7	0	0.5	0.5	12.5
UC58	3	0.555555556	0.44444444	0	7.22222222

UC59	7	0	0.5	0.5	12.5
UC60	3	0.555555556	0.44444444	0	7.22222222
UC61	6	0	0.83333333	0.166666667	10.8333333
UC62	7	0	0.5	0.5	12.5
UC63	6	0	0.83333333	0.166666667	10.8333333
					662.5

Actors			
		Weight	Total
Simple	1	1	1
medium	16	2	32
complex	5	3	15
		UAW	48

Student Projects

Project 1

Use Cases	Number of transaction	Membership to Simple	Membership to Average	Membership to Complex	Fuzzy Weight
UC1	2	1	0	0	5
UC2	5	0.111111111	0.88888889	0	9.44444444
UC3	4	0.333333333	0.66666667	0	8.33333333
UC4	1	1	0	0	5
UC5	5	0.111111111	0.88888889	0	9.44444444
UC6	4	0.333333333	0.66666667	0	8.33333333
			UUCP		45.5555556

Actors			
		Weight	Total
Simple	0	1	0
medium	1	2	2
complex	2	3	3
		UAW	5

Project 2

Use Cases	Number of transaction	Membership to Simple	Membership to Average	Membership to Complex	Fuzzy Weight
UC1	1	1	0	0	5
UC2	2	1	0	0	5
UC3	3	0.555555556	0.44444444	0	7.22222222
UC4	5	0.111111111	0.88888889	0	9.44444444
UC5	1	1	0	0	5
UC6	2	1	0	0	5
UC7	1	1	0	0	5
			UUCP		41.6666667

Actors			
		Weight	Total
Simple	0	1	0
medium	0	2	0
complex	3	3	9
		UAW	9

Project 3

Use Cases	Number of transaction	Membership to Simple	Membership to Average	Membership to Complex	Fuzzy Weight
UC1	3	0.555555556	0.44444444	0	7.22222222
UC2	3	0.555555556	0.44444444	0	7.22222222
UC3	3	0.555555556	0.44444444	0	7.22222222
UC4	2	1	0	0	5
UC5	1	1	0	0	5
UC6	1	1	0	0	5
					36.6666667

Actors			
		Weight	Total
Simple	0	1	0
medium	0	2	0
complex	3	3	9

		UAW	

Project 4

Use Cases	Number of transaction	Membership to Simple	Membership to Average	Membership to Complex	Fuzzy Weight
UC1	4	0.333333333	0.66666667	0	8.33333333
UC2	2	1	0	0	5
UC3	1	1	0	0	5
UC4	5	0.111111111	0.88888889	0	9.44444444
UC5	6	0	0.83333333	0.166666667	10.8333333
UC6	4	0.333333333	0.66666667	0	8.33333333
					46.9444444

Actors			
		Weight	Total
Simple	1	1	1
medium	0	2	0
complex	2	3	6
		UAW	7

Project 5

Use Cases	Number of transaction	Membership to Simple	Membership to Average	Membership to Complex	Fuzzy Weight
UC1	4	0.333333333	0.66666667	0	8.33333333
UC2	5	0.111111111	0.88888889	0	9.44444444
UC3	5	0.111111111	0.88888889	0	9.44444444
UC4	6	0	0.83333333	0.166666667	10.8333333
					38.0555556

Actors			
		Weight	Total
Simple	1	1	1
medium	0	2	0
complex	2	3	6
		UAW	7

Project 6

Use Cases	Number of transaction	Membership to Simple	Membership to Average	Membership to Complex	Fuzzy Weight
UC1	5	0.111111111	0.88888889	0	9.44444444
UC2	2	1	0	0	5
UC3	6	0	0.83333333	0.166666667	10.8333333
UC4	3	0.555555556	0.44444444	0	7.22222222
UC5	4	0.333333333	0.66666667	0	8.33333333
UC6	5	0.111111111	0.88888889	0	9.44444444
UC7	4	0.333333333	0.66666667	0	8.33333333
UC8	3	0.555555556	0.44444444	0	7.22222222
UC9	6	0	0.83333333	0.166666667	10.8333333
					76.6666667

Actors			
		Weight	Total
Simple	1	1	1
medium	0	2	0
complex	2	3	6
		UAW	7

Project 7

Use Cases	Number of transaction	Membership to Simple	Membership to Average	Membership to Complex	Fuzzy Weight
UC1	1	1	0	0	5
UC2	4	0.333333333	0.66666667	0	8.33333333
UC3	6	0	0.83333333	0.166666667	10.8333333
					24.1666667

Actors			
		Weight	Total
Simple	1	1	1
medium	0	2	2
complex	2	3	6

		UAW	9

Project 8

Use Cases	Number of transaction	Membership to Simple	Membership to Average	Membership to Complex	Fuzzy Weight
UC1	3	0.555555556	0.44444444	0	7.22222222
UC2	8	0	0.16666667	0.833333333	14.1666667
UC3	6	0	0.83333333	0.166666667	10.8333333
UC4	3	0.555555556	0.44444444	0	7.22222222
UC5	5	0.111111111	0.88888889	0	9.44444444
UC6	4	0.333333333	0.66666667	0	8.33333333
					57.2222222

Actors			
		Weight	Total
Simple	0	1	0
medium	1	2	2
complex	3	3	6
		UAW	8

Project 9

Use Cases	Number of transaction	Membership to Simple	Membership to Average	Membership to Complex	Fuzzy Weight
UC1	7	0	0.5	0.5	12.5
UC2	3	0.555555556	0.44444444	0	7.22222222
UC3	7	0	0.5	0.5	12.5
UC4	9	0	0	1	15
					47.2222222

Actors			
		Weight	Total
Simple	1	1	1
medium	0	2	0
complex	2	3	6

		UAW	7

Project 10

Use Cases	Number of transaction	Membership to Simple	Membership to Average	Membership to Complex	Fuzzy Weight
UC1	4	0.333333333	0.66666667	0	8.33333333
UC2	6	0	0.83333333	0.166666667	10.8333333
UC3	5	0.111111111	0.88888889	0	9.44444444
UC4	1	1	0	0	5
UC5	4	0.333333333	0.66666667	0	8.33333333
UC6	2	1	0	0	5
UC7	1	1	0	0	5
UC8	3	0.555555556	0.44444444	0	7.22222222
					59.1666667

Actors			
		Weight	Total
Simple	1	1	1
medium	1	2	2
complex	2	3	6
		UAW	9

Projects from Karner's Work

Project A

Use Cases	Number of transaction	Membership to Simple	Membership to Average	Membership to Complex	Fuzzy Weight
UC1	4	0.333333333	0.66666667	0	8.33333333
UC2	4	0.333333333	0.66666667	0	8.33333333
UC3	5	0.111111111	0.88888889	0	9.44444444
UC4	7	0	0.5	0.5	12.5
UC5	4	0.333333333	0.66666667	0	8.33333333
UC6	7	0	0.5	0.5	12.5
UC7	5	0.111111111	0.88888889	0	9.44444444
UC8	7	0	0.5	0.5	12.5
UC9	5	0.111111111	0.88888889	0	9.44444444
UC10	4	0.333333333	0.66666667	0	8.33333333
					99.1666667

Actors			
		Weight	Total
Simple	0	1	0
medium	5	2	10
complex	0	3	0
		UAW	10

Project B

Number of transaction	Membership to Simple	Membership to Average	Membership to Complex	Fuzzy Weight
4	0.333333333	0.66666667	0	8.33333333
5	0.111111111	0.88888889	0	9.44444444
6	0	0.83333333	0.166666667	10.8333333
7	0	0.5	0.5	12.5
4	0.333333333	0.66666667	0	8.33333333
6	0	0.83333333	0.166666667	10.8333333

5	0.111111111	0.88888889	0	9.44444444
7	0	0.5	0.5	12.5
4	0.333333333	0.66666667	0	8.33333333
5	0.111111111	0.88888889	0	9.44444444
7	0	0.5	0.5	12.5
4	0.333333333	0.66666667	0	8.33333333
5	0.111111111	0.88888889	0	9.44444444
5	0.111111111	0.88888889	0	9.44444444
4	0.333333333	0.66666667	0	8.33333333
6	0	0.83333333	0.166666667	10.8333333
5	0.111111111	0.88888889	0	9.44444444
4	0.333333333	0.66666667	0	8.33333333
6	0	0.83333333	0.166666667	10.8333333
5	0.111111111	0.88888889	0	9.44444444
4	0.333333333	0.66666667	0	8.33333333
6	0	0.83333333	0.166666667	10.8333333
5	0.111111111	0.88888889	0	9.44444444
4	0.333333333	0.66666667	0	8.33333333
4	0.333333333	0.66666667	0	8.33333333
5	0.111111111	0.88888889	0	9.44444444
5	0.111111111	0.88888889	0	9.44444444
4	0.333333333	0.66666667	0	8.33333333
6	0	0.83333333	0.166666667	10.8333333
4	0.333333333	0.66666667	0	8.33333333
6	0	0.83333333	0.166666667	10.8333333
4	0.333333333	0.66666667	0	8.33333333
6	0	0.83333333	0.166666667	10.8333333
4	0.333333333	0.66666667	0	8.33333333
7	0	0.5	0.5	12.5
5	0.111111111	0.88888889	0	9.44444444
7	0	0.5	0.5	12.5
4	0.333333333	0.66666667	0	8.33333333
6	0	0.83333333	0.166666667	10.8333333
5	0.111111111	0.88888889	0	9.44444444
4	0.333333333	0.66666667	0	8.33333333

7	0	0.5	0.5	12.5
4	0.333333333	0.66666667	0	8.33333333
6	0	0.83333333	0.166666667	10.8333333
5	0.111111111	0.88888889	0	9.44444444
5	0.111111111	0.88888889	0	9.44444444
4	0.333333333	0.66666667	0	8.33333333
5	0.111111111	0.88888889	0	9.44444444
5	0.111111111	0.88888889	0	9.44444444
7	0	0.5	0.5	12.5
				488.611111

Actors			
		Weight	Total
Simple	0	1	0
medium	5	2	10
complex	0	3	0
		UAW	10

Project C

Use Cases	Number of transaction	Membership to Simple	Membership to Average	Membership to Complex	Fuzzy Weight
UC1	4	0.33	0.67	0.00	8.33
UC2	5	0.11	0.89	0.00	9.44
UC3	6	0.00	0.83	0.17	10.83
UC4	6	0.00	0.83	0.17	10.83
UC5	7	0.00	0.50	0.50	12.50
UC6	5	0.11	0.89	0.00	9.44
UC7	4	0.33	0.67	0.00	8.33
UC8	7	0.00	0.50	0.50	12.50
UC9	5	0.11	0.89	0.00	9.44
UC10	7	0.00	0.50	0.50	12.50
UC11	5	0.11	0.89	0.00	9.44
UC12	4	0.33	0.67	0.00	8.33
UC13	7	0.00	0.50	0.50	12.50
UC14	5	0.11	0.89	0.00	9.44
UC15	4	0.33	0.67	0.00	8.33
					152.22

Actors			
		Weight	Total
Simple	0	1	0
medium	5	2	10
complex	0	3	0
		UAW	10