2007

A NEURO-FUZZY MODEL FOR SOFTWARE DEVELOPMENT EFFORT ESTIMATION

Besa Muslimi
Western University

A NEURO-FUZZY MODEL FOR SOFTWARE DEVELOPMENT EFFORT ESTIMATION

(Spine title: Model for Software Development Effort Estimation)

(Thesis format: Monograph)

by

Besa Muslimi

Graduate Program in Engineering Science
Department of Electrical and Computer Engineering

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Engineering Science

Faculty of Graduate Studies
The University of Western Ontario
London, Ontario, Canada

ABSTRACT

In the software industry, a reliable development effort estimation model remains to be the missing piece of the puzzle. Existing estimation models provide one-size-fits-all solutions that fail to produce accurate estimates in most environments.

Research has shown that the accomplishment of accurate effort estimates is a long-term process that, above all, requires the extensive collection of effort estimation data by each organization. An effort estimation data point is generally characterized by a set of attributes that are believed to most affect the development effort in the organization. These attributes can then be used as inputs to the effort estimation model. The attributes that most affect development effort vary widely depending on the type of product being developed and the environment in which it is being developed. Thus, any new estimation model must offer the flexibility of customizable inputs. Finally, because software is virtual and therefore intangible, the most important software metrics are notorious for being subjective according to the experience of the estimator. Consequently, a measurement and inference system that is robust to subjectivity and uncertainty must be in place.

The Neuro-Fuzzy Estimation Model (NFEM) presented in this thesis has been designed with the above requirements in mind. It is accompanied with four preparation process steps that allow for any organization implementing it to establish an estimation process. This estimation process facilitates data collection, a defined measurement system for qualitative attributes that suffer from subjectivity and uncertainty, model customization to the organization's needs, model calibration with the organization's data, and the capability of continual improvement. The proposed model described in this thesis was validated in a real software development organization.

ACKNOWLEDGMENTS

It's been almost two years since I embarked on this journey of growth and education. As I see it wind down, I cannot help but think of Isaak Walton's wise words:

"Good company in a journey makes the way seem shorter."

For making my journey seem much shorter, I have many people to thank. First and foremost, I would like to thank my supervisor and mentor, Dr. Miriam Capretz. Her invaluable guidance and support, not only in matters of research, but in matters of life itself, will stay with me for many years to come. I would also like to thank Dr. Jagath Samarabandu for pointing me in the right directions and helping me find my way in my research area of soft computing whenever I lost it. Furthermore, I am very grateful to my industrial partner for enabling me to develop and test my research in a real-world environment. I am especially indebted to those who worked with me on this project: Mark Benko for his unwavering commitment, vital contributions, insightful discussions, and critical analysis; Les Fekete for having the vision to instigate this project and fully support it; Dennis Belletti for continuing to support the project; Tom Mesic and Dave Helmer for their indispensable critiques and suggestions; all the developers who took their time to profile the data; and Carole Gorman for her delightful friendship.

Finally, I would like to thank those closest around me whose support and encouragement helped make this endeavor a success. I thank my good friends for turning even the toughest days into joyous ones; my sisters and my brother, Medija, Vlora, and Gramoz, for their eternal friendship, nourishing love, and for always being there for me; and I dedicate this thesis to my parents, Mitat and Zarife Muslimi, for instilling in me the love for education, among many other values, and for devoting their lives to mine and my siblings' happiness.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I
INTRODUCTION

1.1 Research Motivation

In the years 1968 and 1969, the NATO Science Committee sponsored two conferences on software engineering. Many believe that those two conferences instigated the beginning of the profession of software engineering [1]. The idea of applying a structure and a process, together with quality, schedule, and cost constraints to the field of software development was a new concept, but over the years it proved to be a necessary one. As the field of software engineering exploded with growth, many management and control problems were realized. Attempts of resolving them introduced new processes, new methodologies, and new models.

One of the critical problems that emerged early on in the field of software engineering and continues to haunt the field today is the problem of development effort estimation. Accurate estimates are as essential in the software industry as they are in any other industry. Based on estimates, key project decisions are made, feasible performance objectives are defined and schedules are set up. Overestimation leads to lost bids for projects, while underestimation leads to runaway projects and unsatisfied customers. Existing estimation models are frequently unreliable and ineffective. Yet, as the software industry continues to expand in wide-ranging and far-reaching directions, its products becoming vital components of every other industry in the world, it is important that accurate estimates no longer be perceived as luxuries but as essential information to the business of software development. Thus, there exists a need for a reliable software development effort estimation model.

1.2 Problem Statement

In attempts to obtain accurate estimates, different software development effort estimation models have been developed. Some are as simple as product and domain experts estimating the effort of new projects based on past experiences; others are more elaborate and involve the definition of a new system based on a common group of factors that are believed to have a significant effect on effort. But none of the existing effort estimation models have accomplished the goal of consistently providing accurate estimates. The reasons are numerous: they generally assume a one-size-fits-all solution, presenting one set of factors to be measured for all the different software products and development processes in existence; they rely on estimating effort based on size metrics that are to date widely argued; they mostly ignore the importance of data collection and model calibration; and they apply algorithms that generally do not account for the uncertainty and subjectivity within software development metrics. Thus, while accurate effort estimates are still keenly sought out by software organizations, existing estimation models come short of providing this essential information.

In [11], Fenton and Neil develop a list of the desired characteristics that a software development effort estimation model must have. The characteristics they list are:

- The ability to handle diverse process and product variables
- The ability to incorporate empirical evidence and expert judgment
- The ability to determine genuine cause and effect relationships
- The ability to handle uncertainty
- The ability to handle incomplete information

The purpose of this thesis is to research and develop a software development effort estimation model that contains these desirable characteristics.

1.3 Proposed Solution

This thesis proposes a new software development effort estimation model, entitled the Neuro-Fuzzy Estimation Model (NFEM). The NFEM makes use of intelligent algorithms to provide accurate estimates and establish an estimation process. The proposed model overcomes many of the problems faced by existing effort estimation models.

The NFEM is accompanied by a four step preparation process that allows any organization implementing it to establish an estimation process. The preparation process consists of selecting a set of attributes that highly effect effort and collecting effort estimation data profiled with these attributes. The relationship between effort and the selected attributes is then modeled using the collected data and intelligent algorithms; fuzzy logic is also incorporated in the NFEM to account for the subjective and uncertain nature of the collected data.

1.4 Research Methodology

The proposed software development effort estimation model combines several computational intelligence paradigms, such as neural networks and fuzzy logic. Moreover, it makes use of insights gained from decades of research and experience in the field of software engineering to develop a comprehensive and customizable estimation model.

The model is validated in a real-world setting, with data from a large corporation. This provides a realistic view of the problem and the proposed solution. While the full implementation of the NFEM would require several years of data collection, the initial steps of the preparation process are validated in this thesis.

1.5 Thesis Organization

This thesis is organized as follows: Chapter 2 introduces paradigms of computational intelligence used in the development of the NFEM. This includes fuzzy logic, multilayer feedforward neural networks, the adaptive neuro-fuzzy inference system (ANFIS), and algorithms used to extract rules from trained neural networks. Chapter 3 discusses existing estimation models and their shortcomings. It also covers some research studies that apply computationally intelligent algorithms to the task of effort estimation. Chapter 4 describes the Neuro-Fuzzy Estimation Model in detail, including all four preparation steps involved. Chapter 5 introduces the industrial partner and describes the validation dataset obtained from them. Chapter 6 contains the evaluation of the initial preparation steps of the NFEM on the validation data obtained from the industrial partner. And finally, Chapter 7 concludes with a summary of the contributions of this research and directions for future work.

CHAPTER II
SOFT COMPUTING

Soft computing is a branch of computing that tries to mimic the human mind in order to exploit tolerance for imprecision and uncertainty [55]. This thesis applies a combination of two soft computing techniques, fuzzy logic and neural computing, to more accurately calculate software development task effort estimation. Multilayer feedforward neural networks are used to model the relationship between development effort and the factors that affect it, while fuzzy logic is incorporated to deal with the uncertainty and subjectivity present in these factors. And finally, a rule-extracting technique is used to extract rules from a trained neural network which are then embedded into the Adaptive Neuro-Fuzzy Inference System (ANFIS). In this chapter, these soft computing techniques are described in detail.

Section 2.1 explores fuzzy logic and the zero-order Sugeno fuzzy inference system. Section 2.2 examines multilayer feedforward neural networks and the backpropagation learning algorithm. In section 2.3, the adaptive neuro-fuzzy inference system developed by Jang [21] is described. And finally, Section 2.4 examines some of the different methods used to extract rules from trained multilayer neural networks, and describes in detail the one method that is applied in the preparation process of the Neuro-Fuzzy Estimation Model.

2.1 Fuzzy Logic

Conventional logic (also known as Boolean, classic, or crisp logic) only allows for truth or falsehood. An element either belongs to a set or it doesn't. Such black and white logic satisfies the classes and sets of our world that have well-defined boundaries, such as the set of all integers, the set of living things, the set of liquids, and so on. However, classical logic lacks the ability to satisfy sets with ill-defined boundaries. For example, "the set of warm temperatures," "the set of young people," or "the set of cheap cars," do not constitute sets in the usual mathematical sense. These sets are imprecise and there is a degree of uncertainty associated with each element that falls into one of the above sets, in terms of how well the element fits in.

In solution to this problem, in 1965, Zadeh published his seminal paper titled "Fuzzy Sets" [54]. In it, he proposed a new type of logic that addressed the problem of quantifying these imprecise sets: fuzzy logic. The principal idea behind fuzzy logic is the idea of a fuzzy set where the transition from "belongs to a set" to "does not belong to a set" is gradual. This gradual transition is characterized by a membership function that is associated with the fuzzy set [22]. A more formal definition follows in the succeeding subsection.

2.1.1 Fuzzy Set Definition

Let X be a classical set of points with a generic element of X denoted by x. A fuzzy set A in X is characterized by a membership function, μ_A, which associates each point x in X with a real number in the interval [0,1]. The value of $\mu_A(x)$ at x represents the "grade of membership of x in A" [55]. The closer the value of $\mu_A(x)$ to 1, the higher the membership grade of x into fuzzy set A. In other words, the membership grade is the truth value of the statement "x is an element of A."

Example: Let $X = \Re$, be the set of all possible weather temperatures. Usually, X is referred to as the **universe of discourse**, or simply as the **universe**. Figure 2.1 illustrates the fuzzy set of warm weather temperatures. The set is associated with a bell shaped membership function whose domain extends from 3° Celsius to 30° Celsius and whose range extends from 0 to 1.

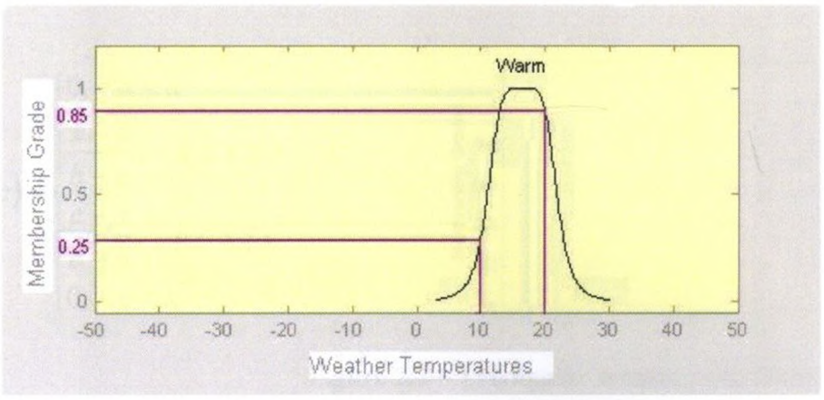


Figure 2.1 – The fuzzy set of warm weather temperatures.

For each temperature value, *x*, we can determine its membership grade by evaluating the membership function, $\mu_{warm\_weather}(x)$, at *x*. Graphically, this is done by extending a vertical line from *x*. The *y*-coordinate of the point of intersection between the vertical line and the membership function curve provides the membership grade. If the line never intersects the curve, then the membership grade is 0. Figure 2.1 shows that the membership value for the temperature 10° Celsius is 0.25 and the membership grade for 20° Celsius is 0.85.

2.1.2 Types of Membership Functions

This sub-section defines two commonly used membership functions and discusses their advantages and disadvantages.

2.1.2.1 Triangular Membership Function

A triangular membership function is specified by three parameters, a, b, and c where $a < b < c$. Together they determine the *x* coordinates of the three corners of the triangular membership function [33], as illustrated in Figure 2.2.

$$triangle(x;a,b,c)=\begin{cases}0, & x<a\\ \dfrac{x-a}{b-a}, & a\le x\le b\\ \dfrac{c-x}{c-b}, & b<x\le c\\ 0, & c<x\end{cases}$$

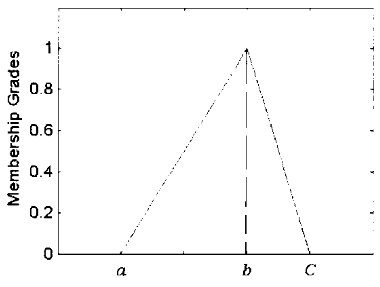


Figure 2.2 – Triangular membership function [33].

The triangular membership function is often used in industrial applications of fuzzy systems due to its simple formula, computational efficiency, and ability to create non-symmetric functions [13]. However, its biggest disadvantage rests in its lack of nonlinear smoothness which is sometimes required.

2.1.2.2 Bell Membership Function

A bell membership function is specified by parameters a, b, and c where a determines the width of the function, b determines the slopes of the sides of the bell function, and c represents the centre of the function [33]. Figure 2.3 illustrates how each parameter affects the shape of the curve. Also, it should be noted that b is usually positive. If b is negative, the shape of the function is an upside down bell.

$$bell(x;a,b,c)=\frac{1}{1+\left|\dfrac{x-c}{a}\right|^{2b}}$$

The bell membership function is computationally inefficient compared to the triangular membership function. However it is nonlinearly smooth, a property required in the adaptive neuro-fuzzy inference system described in Section 2.3 and used in the Neuro-Fuzzy Estimation Model.

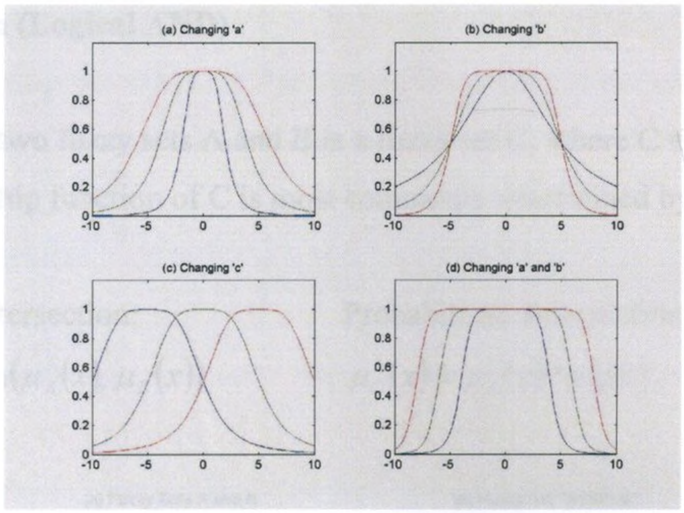


Figure 2.3 – Bell membership function [33].

2.1.3 Fuzzy Operations

This section summarizes two operations through which different fuzzy sets can interact. Many definitions have been developed for each of the operations defined here; however, this section only contains the Zadehian and probabilistic definitions, as they are the most popular [41].

2.1.3.1 Union (Logical OR)

The union of two fuzzy sets A and B is a fuzzy set C, where C = A $\cup$ B (see Figure 2.4). The membership function of C is most commonly determined by:

Zahedian Union:

$$\mu_C(x) = \max(\mu_A(x), \mu_B(x))$$

Probabilistic Union:

$$\mu_C(x) = \mu_A(x) + \mu_B(x) - \mu_A(x) * \mu_B(x)$$

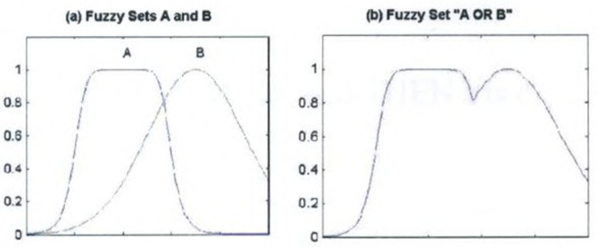


Figure 2.4 – The Zahedian union of fuzzy sets A and B [33].

2.1.3.2 Intersection (Logical AND)

The intersection of two fuzzy sets A and B is a fuzzy set C, where C = $A \cap B$ (see Figure 2.5). The membership function of C is most commonly determined by:

Zahedian Intersection:

$$\mu_C(x) = \min(\mu_A(x), \mu_B(x))$$

Probabilistic Intersection:

$$\mu_C(x) = \mu_A(x) * \mu_B(x)$$

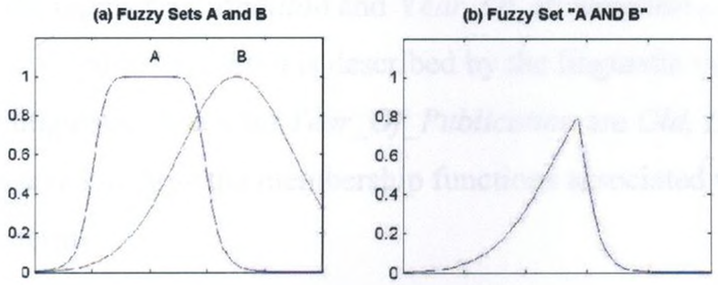


Figure 2.5 – The Zahedian intersection of fuzzy sets A and B [33].

2.1.4 Application of Fuzzy Set Theory: Zero-Order Sugeno Fuzzy Inference

This subsection examines how a fuzzy inference system (FIS) calculates an output from a set of inputs. There are several different variations of the fuzzy inference process. Here, only the zero-order Sugeno-style inference is examined as it is the type of inference system used in the adaptive neuro-fuzzy inference system outlined in Section 2.3 and applied in this thesis.

The zero-order Sugeno inference process is performed in three steps: Fuzzification, rule evaluation, and output determination. The inference system contains rules of the following format:

IF *x* is A AND *y* is B THEN *z* is *c*

The inputs *x* and *y* are linguistic input variables and A and B are fuzzy sets on the universes X and Y, respectively [41]. The output of the system is *z* and it is assigned the value of the constant *c*. A simple example is used to depict the inference process.

2.1.4.1 Example Problem

Let us suppose that we have a zero-order Sugeno fuzzy inference system that determines used computer books' prices based on the condition of the book and the year the book was published. So the inputs are *Condition* and *Year_Of_Publication* and the output is *Price*. The linguistic variable *Condition* is described by the linguistic values *Poor, Good,* and *Excellent*. The linguistic values for *Year_Of_Publication* are *Old, Dated,* and *Recent*. Figures 2.6 and 2.7 show the membership functions associated with each linguistic variable's terms.

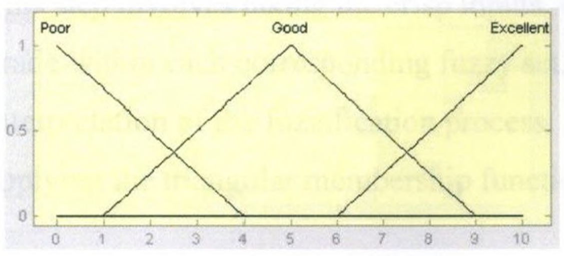


Figure 2.6 – Fuzzy sets Poor, Good, Excellent.

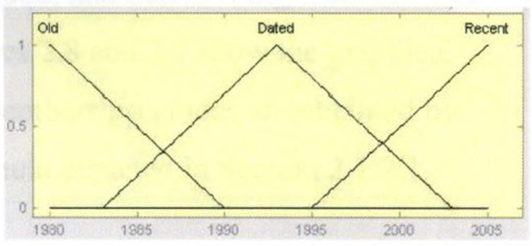


Figure 2.7 – Fuzzy sets Old, Dated, Recent.

For the condition of the book, an integer between 0 and 10 is entered as input, 0 denoting very bad condition, 10 denoting excellent condition. The year of publication is entered for the second input (for simplicity's sake, it is assumed that there are no books to be sold that were published before 1980) and a price between \$5 and \$60 is generated as the output. The example will be illustrated using the inputs outlined in Table 2.1.

Linguistic Variable	Input
Condition	4
Year of publication	2000

Table 2.1 – Sample inputs to the book pricing fuzzy inference system.

The rules for the zero-ordered Sugeno fuzzy system described are:

Rule 1: IF *Condition* is Poor OR *Year_Of_Publication* is Old THEN *Price* is $10

Rule 2: IF *Condition* is Good AND *Year_Of_Publication* is Dated THEN *Price* is $28

Rule 3: IF *Condition* is Excellent AND *Year_Of_Publication* is Recent
THEN *Price* is $55

Rule 4: IF *Condition* is Excellent AND *Year_Of_Publication* is Dated
THEN *Price* is $35

Rule 5: IF *Condition* is Good AND *Year_Of_Publication* is Recent THEN *Price* is $48

To keep the example straightforward, we will assign each rule the weight of 1. It should be noted however, that any weight between 0 and 1 can be associated with each rule, so that some rules affect the output more than others [22].

2.1.4.2 Fuzzification

This step involves taking the crisp inputs, 4 and 2000, and determining their membership grade within each corresponding fuzzy set. Figures 2.8 and 2.9 show the graphical interpretation of the fuzzification process. The membership grades are obtained by applying the triangular membership function formula depicted in Section 2.1.2.1.

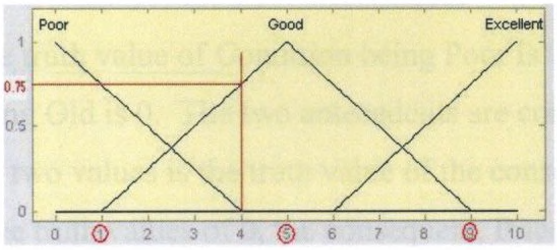


Figure 2.8 – Fuzzification of *Condition*.

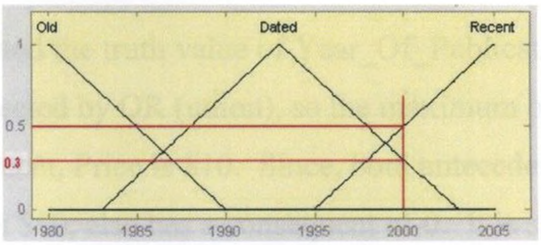


Figure 2.9 – Fuzzification of Year_Of_Pub.

Thus:

$\mu_{Condition=Good}(4) = 0.75$

$\mu_{Year\_Of\_Publication=Dated}(2000) = 0.30$

$\mu_{Year\_Of\_Publication=Recent}(2000) = 0.50$

2.1.4.3 Rule Evaluation

The second step of the Sugeno fuzzy inference process is evaluating each rule. This is done by applying the truth value of the antecedent part of the rule to the consequent part of the rule. The **antecedents** of the rule are the input variables and fuzzy sets that make up the IF part of the rule [41]. The **consequent o**f the rule is the THEN part of the rule [41], as shown below.

$$\text{IF } \overbrace{\underbrace{\textit{Condition} \text{ is Poor}}_{\textbf{Antecedent 1}} \text{ OR } \underbrace{\textit{Year\_Of\_Publication} \text{ is Old}}_{\textbf{Antecedent 2}}}^{\textbf{Antecedents}} \text{ THEN } \overbrace{\textit{Price} \text{ is \$10}}^{\textbf{Consequent}}$$

If there are two antecedents in the rule, then the Zahedian definition of the operation that connects the two parts is applied to determine the consequent truth value. For example:

Rule 1: IF *Condition* is Poor OR *Year_Of_Publication* is Old THEN *Price* is $10

The truth value of Condition being Poor is 0, and the truth value of Year_Of_Publication being Old is 0. The two antecedents are connected by OR (union), so the maximum of the two values is the truth value of the consequent, Price is $10. Since, both antecedents have truth values of 0, the consequent, Price is $10, also has a consequent of 0. It is said that Rule 1 did not fire.

Rule 2: IF *Condition* is Good AND Y*ear_Of_Publication* is Dated THEN *Price* $28

The truth value of Condition is Good is 0.75 and the truth value of Year_Of_Publication is Dated is 0.3. The two antecedents are connected by AND (intersection), so the minimum of the two values is applied as the truth value of the consequent. Thus, Price is $28 by a truth value of 0.3.

Similar analysis of the remaining three rules yield that Rules 3 and 4 do not fire, while Rule 5 fires with a truth value of 0.5.

2.1.4.1 Output Determination

Rules 2 and 5 fired, stating that Price is \$28 and Price is \$48 by truth values of 0.3 and 0.5 respectively. The **weighted average (WA)** of the firing rules' consequents is used to obtain a crisp price:

$$\mathit{Price} = \frac{28 \times 0.3 + 48 \times 0.5}{0.3 + 0.5} = 40.50$$

So the zero-order Sugeno fuzzy model yields a *Price* of \$40.50 when the *Condition* of the book is rated at 4 and the *Year_Of_Publication* for the book is 2000.

2.2 Neural Networks

The artificial neuron was first introduced in 1943 by McCulloch and Pitts [38]. Today, many different types of artificial neural networks are in existence. In this section, only the model of the multilayered feedforward neural network will be examined. This is the most widely applied neural network due to its ability to emulate any input-output relationship [19]. First, the properties of the artificial neuron are explained, followed by the properties of the multilayer feedforward network, and finally the backpropagation algorithm is described.

2.2.1 The Artificial Neuron

The artificial neuron used in today's neural networks is very similar to that developed by Pitts and McCulloch in 1943 [38]. It has a set of input links, where each link is associated with a numerical weight ($w_1,\ldots w_n$), usually between [0, 1] or [-1, 1] (see Figure 2.10).

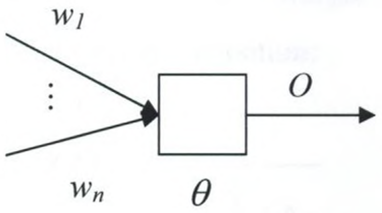


Figure 2.10 – The artificial neuron with *n* inputs.

The neuron has an activation function, which determines the activity level of the neuron. In the multilayered feedforward neural network, this function is usually modeled as the weighted sum of all the neuron's inputs [44]:

$$X = \sum_{j=1}^{n} x_j w_j$$

where x_j is the j^{th} input to the neuron, w_j is the weight of the link supplying the j^{th} input to the neuron, and n is the number of inputs that the neuron has.

Each neuron is also associated with an output function, O, which is usually modeled as the step function or the sigmoid function [44] (Figure 2.11). The sigmoid function approximates the step function but is nonlinearly smooth.

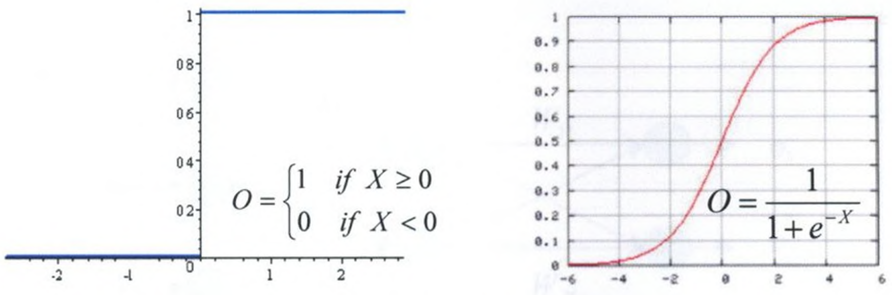


Figure 2.11 – The step function and the sigmoid function.

Furthermore, each neuron is also associated with a threshold value, θ. If the output of the activation function is equal to or above the neuron's threshold value, then the output function is evaluated to be one and the neuron is said to "fire" [44]. Otherwise, the output function is zero and the neuron is said to be "inactive" or "not fire". The node function is the composition of the activation function and the output function, where the threshold of the neuron is considered as the 0th weight of the neuron. Thus, the neuron's firing output is decided by the following equation:

$$O = \begin{cases} 1 & \text{if } \sum_{j=0}^{n} x_j w_j \geq 0 \\ 0 & \text{if } \sum_{j=0}^{n} x_j w_j < 0 \end{cases} \qquad O = \frac{1}{1 + e^{-\sum_{j=0}^{n} x_j w_j}}$$

2.2.2 The Multilayered Feedforward Neural Network

Figure 2.12 shows the architecture of a three-layered feedforward neural network. It consists of three or more distinct layers of nodes, where there are no links between the nodes of the same layer. The outputs of the nodes in the input or a hidden layer are always connected to nodes in the succeeding layers as inputs. The middle layer is called the hidden layer because its inputs and outputs are not observable. Generally, a multilayer feedforward neural network can have more than one hidden layer, but for simplicity, the three-layer neural network is exemplified in Figure 2.12. The nodes of a multilayered network have the same properties as that of the single neuron. Usually, the same node function is used for all the nodes of the network or for all the nodes of a specific layer.

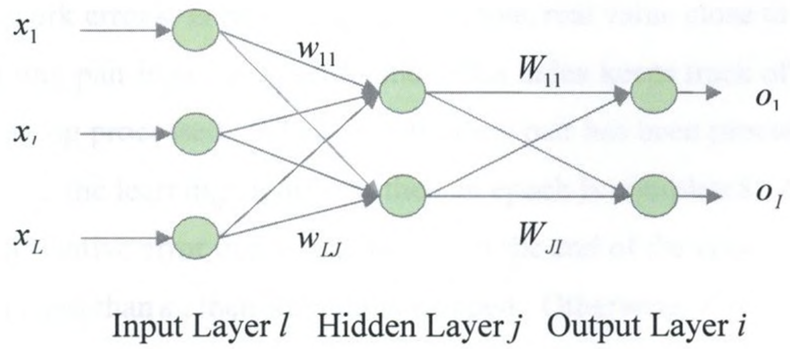


Figure 2.12 – A three-layer feedforward neural network.

2.2.3 The Backpropagation Algorithm

A learning algorithm outlines how a neural network can learn to model the input-output relationship. The backpropagation algorithm allows the neural network to model the input-output relationship by using a training dataset that consists of inputs for which the desired outputs are known. For each data point within the training set, the network output is determined and compared with the desired output. The networks' performance is measured as the discrepancy between the desired output and the network's actual output under the same input [22]. This discrepancy, called the error measure, is usually the mean or the sum of the squared differences between the desired outputs and actual outputs of the training set. The algorithm improves the network's performance by adjusting the weights of the connections between the neurons so that the error measure is minimized. The learning rule is essentially an optimization technique that strives to minimize the error measure. The backpropagation algorithm is based on the gradient descent optimization technique [13]. The gradient descent method takes advantage of the fact that moving in the opposite direction of the derivative of a function leads to a descent. Thus, when the gradient descent method is applied to the error function obtained by comparing desired and actual outputs, the weights of the network are moved in the direction of the descending error.

At the start of the training procedure of a multilayer backpropagation neural network, the initial weights and thresholds are set to small random numbers [13]. A tolerable maximum network error ε is chosen to be a positive, real value close to or equal to zero [13]. The training pair index, k, is set to one. This index keeps track of which input-output pair is being processed. When each training pair has been processed once by the network applying the learning algorithm, then an epoch is completed. A variable E keeps track of the cumulative error during an epoch. At the end of the epoch, E is compared to ε, and if it is less than ε, then training is stopped. Otherwise, E is set to zero again and a new epoch begins. The number of epochs used to train the network can also be used as a criterion for stopping the training because often times, after one thousand epochs, the improvement is negligible [9].

2.3 Neuro-Fuzzy Systems

Neuro-fuzzy systems combine fuzzy inference and neural computing to provide a mechanism that learns from data, is robust to uncertainty and incomplete data, and uses reasoning that is transparent to the user.

In this section, the adaptive neuro-fuzzy inference system (ANFIS) developed by Jang [21] is described, as applied to the zero-order Sugeno fuzzy inference. The system combines the Sugeno fuzzy inference with a multilayered neural network. The rule base of a zero-order Sugeno ANFIS system must be known in advance, while the parameters of the membership functions used for the fuzzy sets of the inputs are adjusted through the training of the ANFIS system. The reason why ANFIS was the neuro-fuzzy system of choice for this thesis was its computational efficiency and availability on the software package MATLAB.

2.3.1 ANFIS Architecture

A generic ANFIS system has five layers (the input layer is not counted as a layer by Jang [21]). Each layer deals with a specific step of the fuzzy inference process such as fuzzification, rule evaluation, firing strength normalization, weighted consequent determination, and weighted consequent summation. Below is a detailed description of each layer. For simplicity, an ANFIS system with only two inputs and one output (shown in Figure 2.13) will be used. The system's inputs are x and y, where x is described by the linguistic values A_1 and A_2, and y is described by linguistic values B_1 and B_2. The membership functions of the fuzzy sets that represent the linguistic values of each input variable are in the shape of the bell function. The output of the system, z, takes on a different value in each rule, corresponding to the constant in the consequent of the rule. Therefore, the rules of the system are:

Rule 1: If x is A_1 and y is B_1 then z is Z_1

Rule 2: If x is A_2 and y is B_2 then z is Z_2

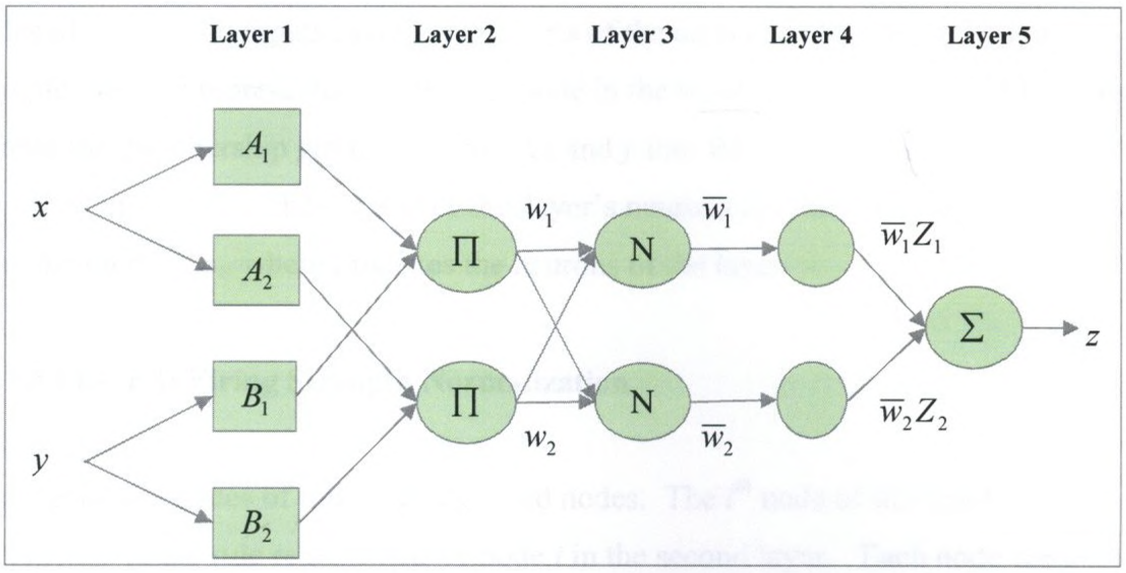


Figure 2.13 – A two-input one-output ANFIS architecture.

2.3.1.1 Layer 1: Fuzzification

Each node in this layer is an adaptable node, represented by a square in Figure 2.13, indicating that the node function's parameters are adjusted during training. Each node function in this layer corresponds to one of the membership functions used for one of the inputs. Consequently, the output of the node is the membership grade of the input within that membership function [21]. Thus, in Figure 2.13, the outputs of the first and second neurons in the first layer are the membership grades of input x into fuzzy sets A_1 and A_2, respectively.

The membership functions are usually generalized bell functions. Therefore, the bell membership function parameters *a, b,* and *c* are adjusted by the learning algorithm during training. The resulting shapes of the membership function are fine tuned to the training data. The backpropagation algorithm is used as the learning algorithm [22].

2.3.1.2 Layer 2: Rule Evaluation

Each neuron of the second layer is a fixed node, represented by a circle in Figure 2.13, indicating that the node function's parameters are not adjusted during training. Each node represents a fuzzy rule of the system. The inputs of a node are the membership grades of the system inputs into the fuzzy sets of the antecedents of the rule. For example, the rule represented by the first node in the second layer of Figure 2.13 takes in as input the membership grades of x into A1 and y into B1. Thus, its antecedents are "If x is A1 AND y is B1". The outputs of this layer's neurons are the firing strengths of the rules, denoted by w_i, where i indexes the neurons of the layer.

2.3.1.3 Layer 3: Firing Strength Normalization

Once again, the nodes of this layer are fixed nodes. The i^{th} node of this third layer corresponds to the rule represented by node i in the second layer. Each node computes the ratio of its corresponding rule's firing strength to the sum of all rules' firing strengths:

$$\overline{w_i} = \frac{w_i}{\sum_{i=0}^{n} w_i}$$

The resulting outputs of this layer's neurons are called the **normalized firing strengths** [22] and each output is denoted with $\overline{w_i}$ where i indexes the nodes of the layer. Thus, in Figure 2.13, the first node computes the normalized firing strength of the rule represented by the first node in the second layer. The input of each node i in this layer is the output of every node in the previous layer, so that the sum of all rules' firing strength can be computed.

2.3.1.4 Layer 4: Weighted Consequent Determination

The neurons of this layer are fixed neurons, and each neuron i of this layer corresponds to the consequent of the rule represented by the i^{th} neuron in layer two. The input of neuron i in this layer is the normalized firing strength of the rule it represents. Its output is the product of the normalized firing strength and the constant value that is contained in the consequent of the rule, resulting in a weighted consequent. Thus, in Figure 2.13, the output of the first neuron in the fourth layer is the product of the normalized firing strength of the first rule, $\overline{w_1}$, and the constant Z_1.

2.3.1.5 Layer 5: Weighted Consequent Summation

There is one node in this layer for each output of the system. For the example system of Figure 2.13, there is only one node, and therefore only one system output. The output of the node is the system output and is computed as the summation of the node's inputs:

$$z = \sum_{i=1}^{2} \overline{w}_i z_i$$

2.4 Extracting Rules from Neural Networks

Because the rule base of an ANFIS system must be known in advance, it is impossible to apply ANFIS unless a rule base exists. In cases where expert knowledge is unavailable, it is possible to extract rules from a trained neural network and implement them into an ANFIS. Rule-extracting algorithms fall into three categories: Decompositional, pedagogical, or eclectic [2]. Due to time constraints, only the decompositional class of algorithms was examined during the development of this thesis. Section 2.4.1 describes the decompositional approach for extracting rules from trained neural networks. In Section 2.4.2 different techniques that have been proposed to do this are reviewed. And Section 2.4.3 describes in detail the rule-extraction technique that is chosen as the most suitable one for the purpose of this thesis.

2.4.1 The Rule Extraction Problem

In a multilayer feedforward neural network, the output of each neuron is calculated as:

$$O_i = Act\left(\sum_{j=0}^{n} x_j w_j\right), \quad x_0 = 1, w_0 = -\theta \qquad (1)$$

where

$$Act(n_i) = \frac{1}{1+e^{-\alpha x}}. \qquad (2)$$

In (1), O_i is the output of neuron i, w_j is the weight on the j^{th} incoming link to neuron i, x_j is the input of the j^{th} incoming link to neuron i, and w_0 is the negative of the neuron's threshold. The activation function $Act(n_i)$ is modeled as the sigmoid function, as shown in (2). The α parameter controls the steepness of the sigmoid function.

The most important characteristic of the decompositional approach is that all neurons in the neural network have inputs of approximately zero or one [2]. In the hidden layer neurons, binary inputs in the input layer allow for this to happen. In the output layer neurons, this is made possible by increasing the α parameter of the hidden neurons (to approximately ten), to ensure that the hidden neurons' outputs approximate Boolean behavior [31]. Having Boolean inputs ensures that the links that are incoming to a neuron carry a signal that is equal to the size of the weight or zero. Thus, rules are extracted from each hidden neuron by finding combinations of incoming weights whose sum exceeds the threshold of the neuron [2]. Since each incoming weight, w_j, is associated with an input, x_j, each combination of weights that results in exceeding the threshold of the hidden neuron can be written as a combination of inputs. By doing so, a rule is created where each input is an antecedent of the rule, and the antecedents are connected by the logical operator AND.

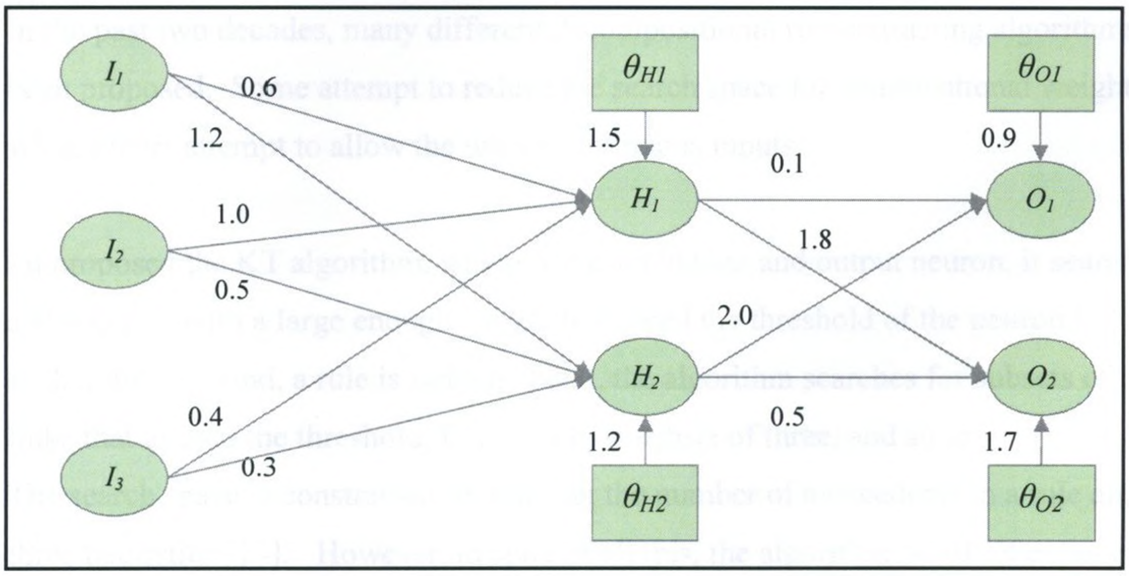


Figure 2.14 – Example neural network used for extracting rules.

For example, Figure 2.14 shows a neural network with three inputs I_1 to I_3, two hidden neurons, H_1 to H_2, and 2 outputs, O_1 to O_2. The thresholds of each neuron are also shown. The threshold of hidden neuron H1 is 1.5. Node H_1 is only activated when inputs I_1 and I_2 are one, because the sum of their weights equals 1.6. The rest of the weight combinations do not yield a sum equal to or greater than 1.5. Thus the successful weight combination is written as a rule "IF I_1 AND I_2 THEN H_1".

Next, for each output neuron, combinations of incoming weights whose sum exceed the threshold of the neuron are determined [2]. Each incoming weight is associated with a hidden neuron, enabling each output to be associated with the set of inputs that activated the hidden neuron. For example, in Figure 2.14 the link between H_1 and O_2 has a weight of 1.8, whereas the threshold of output neuron O2 is 1.7. Thus the incoming link of H_1 is sufficient to activate output neuron, O_2, resulting in the rule "IF H1 THEN O2." By combining the rule extracted from the hidden layer with that extracted from the output layer, the following rule is created: "IF I1 AND I2 THEN O2." This resulting rule models an aspect of the relationship between the inputs and the output.

2.4.2 Review of Existing Decompositional Rule-Extracting Algorithms

In the past two decades, many different decompositional rule-extracting algorithms have been proposed. Some attempt to reduce the search space for combinational weights, while others attempt to allow the use of continuous inputs.

Fu proposed the KT algorithm, where, for each hidden and output neuron, it searches for a single link with a large enough weight to exceed the threshold of the neuron [14]. If such a link is found, a rule is written. Next, the algorithm searches for subsets of two links that exceed the threshold, followed by a subset of three, and so on.
The search space is constrained by limiting the number of antecedents in a rule and using three heuristics [14]. However, in spite of all this, the algorithm is still of exponential complexity and therefore very inefficient [14]. In addition, imposing a maximum number of antecedents in a rule can significantly affect the quality of the rule set [2].

Towell and Shavlik present another rule extracting algorithm in [51]. It is implemented on a special multilayer network developed by them in [52], called the knowledge-based neural network (KBNN). The existing knowledge about the domain is first inserted into the architecture of the network and the network is trained with the backpropagation algorithm. Then links with similar weights are combined into clusters and the average of the cluster's weights is used as the weight of each link belonging to that cluster. Clusters with low link weights and few members are then eliminated as they are assumed to have little influence on the outcome of the network. The weights of the links are then fixed and the network is retrained with the backpropagation algorithm to adapt the thresholds of the network. Finally, a rule is written for each hidden unit and output unit where each antecedent of the rule is associated with a weight and the rule is associated with the threshold of the neuron. The rules take the form:

If M of N antecedents are TRUE then C

The primary goal of the M-of-N algorithm is to refine rules contained in the initial rule base. This limits its use to domains where the input-output relationship knowledge exists. Also, it does not allow for new and unexpected knowledge to be discovered [2].

While the above algorithms extract crisp rules and only deal with Boolean inputs, other researchers have proposed algorithms for extracting fuzzy rules or systems that deal with fuzzy inputs. Hayashi and Imura proposed a fuzzy neural expert system with automated extraction of fuzzy rules that can handle fuzzy and crisp inputs [18]. In addition, each extracted rule is associated with a fuzzy truth value such as *Very True* or *Possibly True*, and each antecedent in a rule is associated with a fuzzy importance value such as *Very Important* or *Moderately Important*. However these truth values and importance values make the algorithm difficult to implement.

Kasabov's REFuNN algorithm [25],[26], applied to the specially constructed fuzzy neural network (FuNN) also has the ability to extract fuzzy weighted rules as well as simple fuzzy rules; however, the number of rules extracted for a fairly simple problem such as the Iris classification data [42] is very large.

NEFCLASS (Neuro Fuzzy CLASSification) is a neuro-fuzzy system for the classification of data and is presented by Nauck and Kruse in [39], [40]. The goal of the system is to learn fuzzy rules from the training data patterns as it classifies each pattern into crisp classes. However, a ceiling is placed on the maximum number of rules extractable, a constraint that could seriously hinder the quality of the rule set [2].

2.4.3 The Selected Algorithm

Krishnan et al. provide a simple and efficient technique for extracting rules from feed-forward neural network in [31]. They start by applying the method presented by Sethi and Yoo [48] to convert any negative weights in the network into positive ones. Then the weights of a given neuron of the hidden or output layer, are sorted in descending order and combinations of all possible sizes are created. Subsequently, the combinations of any particular size are ordered in descending order of the sum of the weights in the combination. Then a combination tree is created where all combinations of size i are placed at the i^{th} level of the tree, while maintaining the descending order [31]. For example, given a node with four weights of values 4, 3, 2, 1 when sorted in a descending order, Figure 2.15 shows the combination tree.

When searching for combinations of weights that exceed the threshold of the neuron, the search space is reduced in two ways. First, if a combination at any level fails, the rest of the combinations in that level can be ignored, because they will also fail [31]. For example, suppose the neuron whose combination tree is shown in Figure 2.15, has a threshold of three. Once weight 2 fails to exceed the threshold in Layer 1, there is no need to check weight 1. Since the weights were sorted in a descending order, it is given that weight 1 will also fail.

The second way to reduce the search space is by eliminating combinations at a lower level of the tree that subsume a combination at a higher level that was successful in forming a rule [31]. For example, in Figure 2.15, weight 4 of Level 1 is successful in activating the neuron. Therefore, all combinations in the lower levels of the tree (shown in italicized red print) that subsume weight 4, will form less general rules and thus can be ignored.

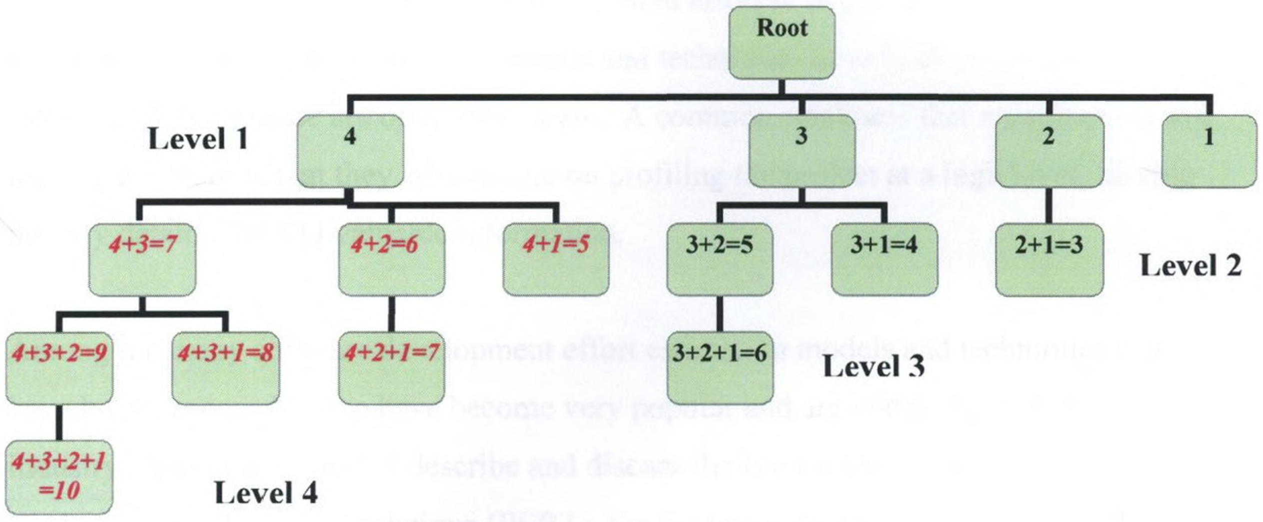


Figure 2.15 – Example combination tree.

This algorithm is selected to be applied in the NFEM preparation process due to its simplicity and efficiency. It is able to extract all significant rules while reducing the search space significantly without introducing complex heuristics or constraints that could hinder the accuracy and comprehensiveness of the extracted rule set.

CHAPTER III
LITERATURE REVIEW

The problem of estimating software development effort is as old as the field itself. Over the decades many effort estimation models and techniques have been proposed, but the estimates they produce are often inaccurate. A common weakness that most models and techniques share is that they concentrate on profiling the project at a high level, leaving out any details that add valuable information.

Among the many software development effort estimation models and techniques that have been proposed, some have become very popular and are widely used by the industry. Sections 3.1 to 3.5 describe and discuss the Expert Delphi technique, the Peer Evaluation and Review Technique (PERT), the Software Life-cycle Model (SLIM), COCOMO II, and Function Point Analysis, respectively. In addition, in recent years, many effort estimation techniques using soft computing have been proposed, though not many have been put to use by industry in everyday practice. Section 3.6 presents some of the software estimation techniques proposed that make use of soft computing. Finally, Section 3.7 discusses three factors that have been found to strongly affect software development effort, throughout many research studies conducted.

3.1 The Expert Delphi Technique

One of the very first estimation techniques arose in the late 1950's from the RAND Corporation and became known as the Expert Delphi technique [50]. The technique is iterative in nature, consisting of a set of rounds. In each round, several developers estimate the value of one or more items anonymously and independently. An item is a software development task and can vary from being a function, an object, a full system feature, or any other unit into which the software system being estimated is decomposed.

At the end of a round the estimators are only shown the maximum, minimum, and average estimated values for each item for that round. The idea behind this technique is that eventually a convergence value for each item being estimated will be reached.

Today, most organizations which confirm to use this technique do not follow the formal process described above [24]. Instead, usually each item being estimated is discussed by the team until an effort estimate is agreed upon.

The problem with this technique is that it is subjective to the individual experience of the estimator, his or her incomplete recall and bias, and as such, the estimates are no better than the participants involved [50]. But perhaps the greatest challenge that is present in the use of this technique is that studies continually show most estimators tend to underestimate [24]. Jørgensen cites 8 studies that are consistent in concluding that "experts can be strongly biased and misled by irrelevant information towards over-optimism" while estimation models are not [24]. In addition, Laranjeria cites a study done by Yourdon Inc., where experienced managers were asked to estimate the size of 16 projects [32]. Over half of the projects were severely under-estimated with an MRE of 100% or more, and many of them had an MRE of over 200%. The study found that the reasons for the underestimation tendency include the desire to please management, incomplete recall of previous experience, lack of familiarity with the entire software job, and the lack of sufficient knowledge of the particular project being estimated.

3.2 Program Evaluation and Review Technique (PERT)

Originally developed by Lockheed and the U.S Navy in the late 1960's, the Program Evaluation and Review Technique (PERT) is an easy and simple technique which, to some extent, takes into account the estimator's uncertainty in determining the estimate [50].

The technique requires the estimator to provide three estimation values, the pessimistic value, the most likely value, and the optimistic value. Assuming that the optimistic and pessimistic values correspond to minus and plus the three-sigma limits of the distribution, respectively, the expected value is calculated as:

$$E = (O + 4*M + P)/6$$

where O represents the optimistic value, M represents the most likely value, and P represents the pessimistic value. The value obtained by dividing the standard deviation by the estimated effort value indicates the degree of uncertainty in the estimator's part [50]:

$$Uncertainty\ Degree = \sigma/E$$

where

$$\sigma = (P-O)/6.$$

In comparison to the expert technique, PERT offers the benefit of quantifying the estimator's degree of uncertainty. However, PERT also suffers from the same underestimation tendency that the expert estimates suffer: People's "most likely" estimates tend to cluster toward the optimistic estimates [7], [50].

3.3 Software Lifecycle Model (SLIM)

In 1978, Lawrence Putnam published a paper, [46], in which he described a new way of estimating the software lifecycle effort. Putnam's SLIM model is based on the Norden-Rayleigh distribution and builds on the ideas of the PERT technique. As shown in Figure 3.1, the Rayleigh curve shows a build up at the start, peaks when the product is delivered to the customer, and then tails off.

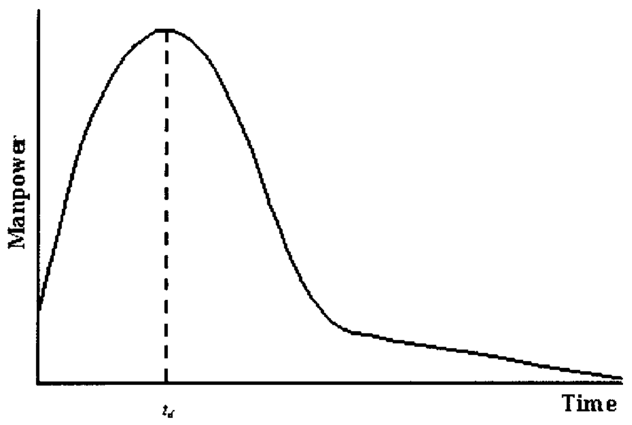


Figure 3.1 – The Norden-Rayleigh distribution.

Based on the findings of the Norden-Rayleigh Distribution, Putnam developed the the general equation of the SLIM model [46]:

$$E = \left(\frac{Size}{P}\right)^{\frac{9}{7}} MBI^{\frac{4}{7}}$$

where P is the *productivity process parameter*, E is the *total software lifetime effort* measured in person years, *Size* is measured in lines of code or function points, and constant *MBI* is the *manpower buildup index.*

Putnam's technique is modeled by the SLIM tool, which is developed by Quantitative Software Management (QSM), a company established by Putnam [47]. The tool allows the inputs to be for the pessimistic, most likely, and optimistic scenarios. The manpower buildup index and the productivity process parameter are best determined from historical data. If no such data exists, then the user of the SLIM tool must answer a series of 22 questions from which the tool determines the two parameters based on data collected from over 6,300 industry projects [37]. The productivity process parameter entails such factors as tools being used, languages being used, process methodologies being followed, and so on [49]. The manpower buildup index is based on factors such as management constraints (e.g. maximum allowable schedule), accounting (e.g. labor rates), personnel skill and qualifications, and other such factors [37]. While in the original model, size could only be measured in lines of code (LOC), today, QSM's SLIM tool allows several other options such as function points, objects, etc. [37].

Despite the use of 6300 industrial project data, studies have found that SLIM performs poorly when used in an environment for which it is not calibrated [45], [24]. Yet, because SLIM is designed to estimate effort at a high level, it takes a long time to collect enough calibration data, and as a result most companies do not invest the time in collecting the data. In addition the model relies on the assumption that the size parameter which is measured in terms of lines of code or function points is correct. The problem with this assumption is that while the LOC metric is easy to measure and it is an "artifact" of all software development projects [45], it also has a lot of disadvantages. First of all, it is programming language dependent, so when used, the productivity appears to decrease as the level of the programming language increases. Also, well-designed but shorter programs are penalized [50]. In addition, the LOC metric is less suitable for non-procedural languages and to estimate the lines of code, one requires a level of detail that may be difficult to achieve [45]. Due to these disadvantages, studies have found that LOC is not a reliable metric [32], and that in fact, it is easier to estimate effort terms of hours than lines of code [15]. The disadvantages of using function points are discussed in Section 3.5 of this chapter.

3.4 COCOMO II

In 1981, Barry Boehm published his famous book titled Software Engineering Economics, [7], where he first described the Constructive Cost Model (COCOMO). The second version of this model, COCOMO II, was introduced in 2000 as a result of the major changes that had taken place in the software development field between 1981 and 2000 [8]. Thus only COCOMO II is described in this thesis, since its prequel, COCOMO, has become outdated.

The COCOMO II model consists of three level models that are used at different stages of the development process: *Application Composition*, *Early Design*, and *Post Architecture* [8]. The Post Architecture model is the most frequently used model version today [50], and is explained in detail.

In the *Post Architecture* model, effort is determined based on the size of the product (measured in LOC or function points), a set of 17 cost drivers (CD), and variables *A* and *B*. The cost drivers quantify the effect that different product, hardware, personnel, and project factors (shown in Table 3.1) have on the effort. Each cost driver is rated on a scale of *Very Low* up to *Extra High*. Variables *A* and *B* account for the linear and the non-linear effect that increasing project size has on the effort estimate [8], respectively. They are determined based on the five scale factors summarized in Table 3.2. The effort estimation equation used in COCOMO II is:

$$E = \prod_{i=1}^{17} CD_i \times A \times Size^B \quad where \quad B = 1.01 + 0.01\sum_{i=1}^{5} SF_i$$

Product Cost Drivers	Hardware Cost Drivers	Personnel Cost Drivers	Project Cost Drivers
Required software reliability	Execution time constraints	Personnel continuity	Use of software tools
Complexity of the product	Platform volatility	Programmer capability	Multisite development
Size of application database	Main storage constraints	Programming language and tool experience	Required development schedule
Required reusability		Analyst capability	
Documentation match to lifecycle needs		Applications experience	
		Platform experience	

Table 3.1 – Post-Architecture COCOMO II cost drivers.

1. Precedentedness	4. Team cohesion
2. Development flexibility	5. Process maturity
3. Architecture/risk resolution	

Table 3.2 – Post-Architecture COCOMO II scale factors.

COCOMO II suffers from the same shortcomings that SLIM does. Studies have found that it must be calibrated to the environment using it in order to be used with some success [45], [24]. Furthermore, it is also based on the assumption that the lines of code estimate or function point estimate is accurate, which, as discussed in Sections 3.3 and 3.5, is often an incorrect assumption.

3.5 Function Point Analysis

Unsatisfied with the often erroneous estimates that the lines of code metric produced, Allen Albrecht designed his own size metric in the late 1970s, while working at IBM [50]. The new metric, called *function point*, was designed to deliver functionality in terms that users could understand, be independent of process, technology, or programming language, and give a reliable indication of software size in the early design stages [50].

The function point analysis process quantifies product functionality based on the following system elements which Albrecht called function types: external inputs, external outputs, external inquiries, internal logical files, and external interface files [50]. Once all instances of each function type are identified, they are associated with a numerical complexity value representing low, average, or high complexity, and then summed. This sum is the *unadjusted function point (UFP)* count of the system [50]. Next, the fourteen *general system characteristics* (*GSC*) shown in Table 3.3 are rated on their degree of influence, 0 being no influence and 5 being strong influence throughout. *The total degree of influence* (*TDI*) is calculated by summing the degree of influence values of all the GSCs. Finally, the *value adjustment factor* (*VAF*) is calculated (1), followed by the *adjusted function point* count (*AFP*) as shown in (2):

$$VAF = 0.65+0.01*TDI \quad (1)$$

$$AFP = VAF*UFP \quad (2)$$

Data communications	Heavily used configuration	End-user efficiency
Distributed data processing	Transaction rate	On-line update
Performance	On-line data entry	Complex processing
Reusability	Operational ease	Facilitate change
Installation ease	Multiple sites	

Table 3.3 – The fourteen general system characteristics.

The function point metric consists of a process that forces the estimator to perform a careful analysis of the system or component being implemented, which generally leads to a more accurate estimate [50]. Also, in general, if performed by a trained and experienced estimator, it is much more accurate than the LOC metric [50]. However, many of the entities counted in the FP process are still very much debated by experts [15]. Furthermore, the FP metric is designed to perform well for data processing applications. But in the past two decades, the software development industry has seen a boom in logic-complex applications, for which the function point model is not well suited [50], [45], [15]. Finally, the FP metric is often used with an effort estimation model such as SLIM or COCOMO II to convert the function point count into effort. As a result, the disadvantages of the effort estimation model used to perform the conversion are inherited.

3.6 Software Development Effort Estimation and Soft Computing

In the past two decades, many researchers have studied the idea of applying soft computing techniques to the problem of effort estimation, and some of these are discussed in this section. The estimation techniques described show that much potential lies in the use of soft computing in software development effort estimation. However none of the proposed techniques have been able to fully encompass the advantages of neural networks while offsetting the disadvantages by fuzzy logic and neuro-fuzzy systems.

In [16], Gray and MacDonell examine the implications of using non-traditional estimation methods, such as neural networks, fuzzy logic, case-based reasoning systems, and regression trees, versus the traditional regression analysis methods. These methods were examined in terms of their ability to model the problem, their reasoning transparency, and their generalisability. It was found that fuzzy and neuro-fuzzy systems performed best in all areas examined. Furthermore, an empirical study that compared neural networks to regression analysis found that the mean absolute relative error generated by neural networks was half of what the regression techniques generated. The data consisted of 81 projects.

Another study was conducted by Boettichier in [9], where the author used data derived from 104 different programs to train multilayer backpropagation neural networks and predict actual effort in hours. The metrics tested consisted of the programs size, vocabulary, objects, and complexity. Each metric was further broken down into one or more quantitative measures, totaling 10 inputs. Different input combinations and network architectures were tested, totaling over 33,000 experiments. The testing showed that using all inputs or the combination of size, object, and vocabulary inputs yielded the best results while individual metrics did not fare well. When the trained model was tested with data from a completely different corporation, on average, the validation results produced estimates within 30% of the actuals, 73.26% of the time. No experimentation results were given that compared the success of various network architectures used. While Boettichier's approach consisted of some novel ideas, the quantitative inputs used require knowledge at a level of detail that is most often not known at the time estimates are created.

Furthermore, in [10], Boettichier conducted further tests, where software projects were estimated using a bottom-up technique and a neural network. Data from two different corporations were used and the only input to the network was size in terms of LOC. Individually, an average of only 9% of the development tasks was predicted with 25% accuracy. But when summed up as total for the project, the project was predicted with 25% accuracy 90% of the time. This study showed that decomposing a project into smaller tasks and using neural networks to generate estimates yields a high accuracy of project effort estimation, even though the tasks are not always accurately predicted.

Finnie and Wittig also conducted experiments using neural networks with the ASMA project data [12]. When only function point count was used as input, only 56% of the data were estimated within 25% accuracy. However, when other attributes such as language and complexity were used, over 77% of the data were estimated within 25%

accuracy. This study, like [9], showed that using more than the estimated size attribute as input, when estimating using neural networks, yields more accurate results.

In an attempt to use the success of neural networks, while avoiding the reasoning opacity that accompanies the use of neural networks, Huang used neuro-fuzzy logic to estimate software development effort [20]. He did so by fuzzifying the inputs of the COCOMO model and using them together with data to train a neuro-fuzzy system [20]. While the resulting neuro-fuzzy model outperforms COCOMO, its further improvement is questionable because the model uses the COCOMO regression equation instead of rules to infer estimates. By doing so, the model inherits the limitations of the COCOMO regression technique, revealed by Gray and MacDonell in [16].

3.7 Factors that Affect Software Development Effort Estimation

The effort estimation models and techniques presented thus far differ in the number and the type of factors they consider to be influential on software development effort estimation. In practice, such factors vary greatly depending on the development environment and type of system being built. However, during the development of the Neuro-Fuzzy Estimation Model, the research conducted led to the conclusion that there are three particular attributes that tend to affect software development effort regardless of all other circumstances. Expert effort estimation, task implementer capability, and complexity were found to have the highest effect on software development effort, by most academic and industrial experts, as well as studies. These effects are discussed in subsections 3.7.1 to 3.7.3.

3.7.1 Expert Effort Estimation

In [24] Jørgensen summarizes a vast number of studies done on expert estimation including how often it is used in the software development industry, why it is used, and how well it performs compared to other estimation models. Many of the studies reveal that informal expert estimation is the most widely used estimation technique in companies all over the world [24]. The results of these studies are summarized in Table 3.4. Furthermore, Jørgensen states that they "were not able to find any study reporting that *most* estimates were based on formal estimation models" [24].

Organization(s) Providing the Data Through Experimental Studies or Questionnaires	% of Time Expert Estimation is Used
Jet Propulsion Laboratory	83
Dutch Companies	62
New Zealand Companies	86
International Financial Company – Information Systems Development Department	100
Telecom Company	84
Software Development Companies	72

Table 3.4 – Results of studies conducted on Expert Estimation [24].

A number of studies that compare expert estimation to model-based estimation techniques are also cited in [24]. Of the 15 studies cited, 5 concluded in favor of the expert technique, another 5 found that there was no difference in the estimated accuracy between the expert estimation technique and model-based techniques, and 5 concluded in favor of model-based techniques. The studies were conducted between 1990 and 2002, and the number of participants in each study varied from 1 to 597.

While such results show that no existing effort estimation model is very accurate, they also show that experts are useful resources when it comes to estimation. In fact, most industry and academic researchers agree that an expert's opinion is not only useful but often necessary when making estimations [15], [45], [50], [24]. Furthermore, software development is not the only field that uses expert estimation, many other domains such as medicine, business, and psychology, recognize it as an important and often decisive tool in planning [24]. Hence, it is strongly recommended that expert effort estimation be included as an input to any effort estimation model.

3.7.2 Implementer Capability

Almost everyone in the software development field will agree that when estimating effort for a development task, one of the most important factors that influences the effort estimate is the quality and capability of the task implementer [15], [28], [45] [7]. Therefore, it is important that any estimation model include implementer capability as one of the inputs to the model. This can be easily done when the effort estimate of an entire project is being determined. But when the effort estimate of a more granular task is being determined, and it is known who the task implementer is, organizations are reluctant to evaluate the capability of that implementer in order to incorporate it into the estimation model. The reluctance is due to the existing confidentiality contract, between employer and employee, such evaluations can breach and also due to the decrease in morale that low evaluations would yield. To have the implementers evaluate themselves poses the problem of bias, how biased the values are. The reason being, that no one would consistently rate him/herself as being a "low quality task implementer" even if that were true. That is, while other attributes would contain some error due to the unconscious bias of the estimator's past experience, the implementer capability attribute could contain error even when the estimator was conscious of it. Therefore, in some organizations, it may be impossible to include implementer capability as one of the inputs to an effort estimation model. In this case, other inputs can be included that evaluate the implementer's familiarity level with the technology, functionality, language, and domain associated with the task. By doing so, the knowledge and capability of the implementer are transformed into task characteristics, and the implementer does not feel it is as personal, yet his or her capability in completing the task is well evaluated.

3.7.3 Complexity

Complexity is another attribute that is considered to have among the highest effect on software development effort [45], [28], [50]. Glass notes that for every 25% increase in problem complexity, there is a 100% increase in the complexity of the software solution [15]. Also, Keyes cites a study done by Lederer and Prasad that found that managers consider complexity to be the most significant factor of a project estimate [28]. While it is argued to be subjective to the developer's experience [15], it is nonetheless important to measure. Therefore, it is strongly recommended that complexity is used as an input to any software development effort estimation model.

CHAPTER IV
THE NEURO-FUZZY ESTIMATION MODEL

This chapter presents the Neuro-Fuzzy Estimation Model (NFEM). The NFEM is a new software development effort estimation model that was designed to encompass several desired characteristics lacking in existing estimation models. It is accompanied by a preparation process that consists of four steps. This preparation process allows the model to be customized according to the specific environment of the organization using it. The NFEM uses historical data from the specific environment implementing it, in conjunction with intelligent algorithms to best model the organization's needs and cope with uncertainty and qualitative data. It is these combined characteristics that render the NFEM usable in many different environments and for the development of many different types of software systems.

Figure 4.1 depicts the NFEM preparation process. As shown, there are four preparation steps that must be completed before the NFEM is closely customized to the organization's environment and ready for use:

1. Attribute selection
2. Data set separation for
    a. Qualitative attributes
    b. Quantitative attributes
3. Neural network training
4. Rule extraction and ANFIS implementation.

In the first step, qualitative and quantitative profiling attributes that are believed to most influence software development task effort are selected by the organization implementing the NFEM.

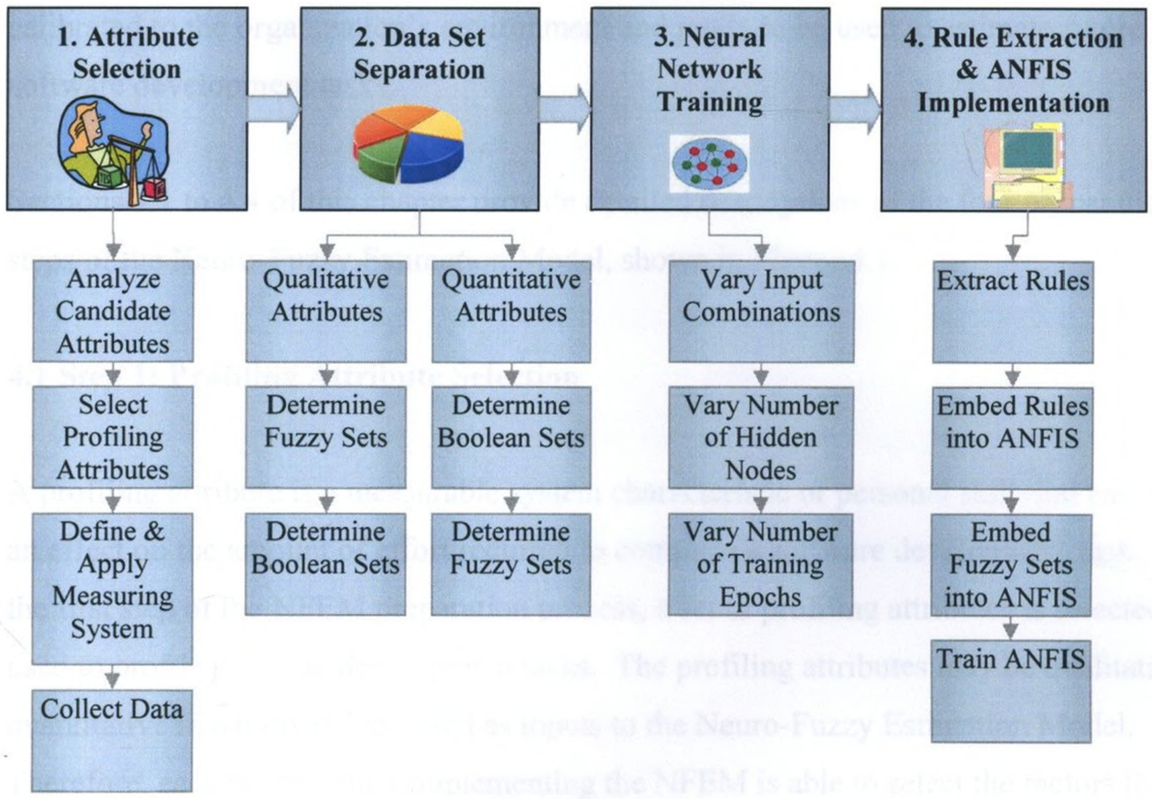


Figure 4.1 – The NFEM preparation process.

These attributes are defined and a measuring system is applied to each of them, allowing them to function as metrics. These attributes then serve as inputs to the NFEM and each software development task is profiled with them. The profile, together with the actual effort required to complete the task (known once the task has been completed) are considered as one data point. When a sufficient amount of data points have been collected, the second step of the preparation process begins: The values of each attribute are separated into fuzzy and Boolean sets. The Boolean sets allow the attribute values to be transformed into Boolean data. The fuzzy sets are not used until Step 4 of the preparation process. The third step consists of using the Boolean data to train neural networks where the output of the neural network is estimated effort. In the fourth and final step, rules that model the relationship between the profiling attributes and development effort are extracted from the most successfully trained neural network of Step 3. They are then embedded into an adaptive neuro-fuzzy inference system, as are the fuzzy sets determined in Step 2. Finally, the historical data collected is used, once again, to train the ANFIS and fine-tune the fuzzy sets. At this point, the NFEM is calibrated to the organization's environment and ready to be used to estimate future software development tasks.

Sections 4.1 to 4.4 of this chapter provide detailed descriptions of the four preparation steps of the Neuro-Fuzzy Estimation Model, shown in Figure 4.1.

4.1 Step 1: Profiling Attribute Selection

A profiling attribute is a measurable system characteristic or personal skill that can have an effect on the amount of effort required to complete a software development task. In the first step of the NFEM preparation process, a set of profiling attributes is selected and used to profile software development tasks. The profiling attributes may be qualitative or quantitative in nature and are used as inputs to the Neuro-Fuzzy Estimation Model. Therefore, each organization implementing the NFEM is able to select the factors that it believes most influence development effort, given the product and environment in which they develop. By profiling each software development task with this common set of profiling attributes and recording the actual effort required to complete each task, a historical data point is created. A collection of such historical data points can then be used to more accurately estimate future development tasks. That is, the effort of new development tasks can be estimated based on the amount of effort historical tasks with similar profiles required. How effort is measured is left to each organization's discretion.

Research has shown that of the hundreds of parameters which can affect software development effort, only a few of these may affect the productivity in a given environment [34]. In order for each organization using the NFEM to be able to select the few attributes that most affect productivity, it is recommended that many attributes are initially selected to be measured and recorded. The ones that turn out to be irrelevant during the neural network training stage (step 3 of the NFEM preparation process) can be discarded and the most predictive ones can be kept. Ideally, the final number of profiling attributes selected as inputs to the NFEM should be low. The reason for this is that the NFEM uses a neural network to learn the input-output relationship from the historical data and the higher the number of input attributes, the higher the volume of data needed to train the neural network successfully. In fact, the number of data points needed to train a neural network grows exponentially with each input [13].

The attribute selection process can also be done on a per-product basis if the organization is large and is involved in software development for several different domains. In addition, different attributes should be selected for different phases of the software development lifecycle. The selected common set of profiling attributes will vary greatly, depending on the development environment and type of system being built. However, most organizations should include the attributes expert effort estimation, implementer capability, and complexity, for the reasons discussed in Section 3.7.

4.1.1 Defining and Applying the Measuring System

In the first step of the NFEM preparation process, not only must the profiling attributes be selected, they must also be clearly defined, to indicate what is being measured by each attribute. In addition, a measuring system must be implemented for each qualitative attribute so that they can be measured. This need not be done for the quantitative attributes, as they can be simply entered as numbers.

The NFEM qualitative attribute measuring system was designed to allow the use of qualitative attributes where each such attribute is measured according to the needs of the organization using it. The measuring system requires that each qualitative attribute be defined and be further decomposed into sub-attributes. Each sub-attribute should be a factor that affects the evaluation of the profiling attribute. Once the user evaluates the sub-attributes, their values can be averaged and used as the overall profiling attribute's measurement. To facilitate this process, Tables 4.1 and 4.2 were designed and should be completed for each selected qualitative profiling attribute. Table 4.1 should also be used to define the quantitative attributes.

Name	The name of the attribute and any short names used for it.
Definition	A clear and concise definition that indicates what is being measured by the attribute. All attributes should be defined in terms of how they affect effort.
Rationale	The rationale used to select the attribute. One or more examples may be given to clarify the rationale.
Implementation	A way of combining all the sub-attributes into a single unit of measurement (usually the arithmetic mean). Also, any clarification notes on how the attribute should be perceived or evaluated.

Table 4.1 – The format to be used to define the selected profiling attributes.

Sub-Attribute Name			
Definition	A clear and concise definition of the sub-attribute		
Scale Values' Definitions	Low	The definition of the "Low" scale value for this particular sub-attribute.	An arrow depicting the direction in which the effort estimate increases due to the sub-attribute's evaluation.
	Medium-Low	No definition required.	
	Medium	The definition of the "Medium" scale value for this particular sub-attribute.	
	Medium-High	No definition required.	
	High	The definition of the "High" scale value for this particular sub-attribute.	

Table 4.2 – The format to be used to define the sub-attributes.

The definitions of the scale values required in Table 4.2 serve as guidelines for the estimator. The scale values "Medium-Low" and "Medium-High" need not be defined because they are to be interpolated by the estimator. Each scale value corresponds to a number between one and five, with one corresponding to the Low set and five corresponding to the High set. The overall profiling attribute valuation is the arithmetic mean of its sub-attributes' values.

It should be noted that while breaking down a profiling attribute into sub-attributes does lengthen the preparation process, it is more beneficial in the long term because it allows the collection of more accurate data. Qualitative attributes that affect software development effort usually encompass several aspects of the quality they describe. For example, when defining the required reliability of a software system, one thinks of how a system failure would affect the environment and users of the system (i.e. a mere inconvenience versus endangerment of human life), the acceptable frequency of failures (mean time to failure, MTTF), and the acceptable repair time (mean time to repair, MTTR). While all three of these factors affect the reliability of a system, they do so in different ways, and they would often result in different evaluations from one another. Therefore, clustering the three factors and measuring them as one would introduce a lot of uncertainty and inaccuracy into the measurement. On the other hand, decomposing the attribute into sub-attributes allows for more accurate data to be collected.

Table 4.3 shows the application of Table 4.1 to the Reliability profiling attribute and Table 4.4 shows application of Table 4.2 to its sub-attributes. The definitions provided are according to a fictional organization that creates business-critical applications.

Name	Reliability
Definition	The degree of reliability required from the component or functionality implemented in the task.
Rationale	A task that involves the development of a highly reliable component or functionality generally requires more effort.
Implementation	Each sub-attribute is evaluated as Low, Medium-Low, Medium, Medium-High, or High, corresponding to the values between 1 and 5, respectively. The average of the sub-attribute evaluations is the overall attribute value.

Table 4.3 – Example definition of the Reliability profiling attribute.

Criticality			
Definition	The problem created if the component or functionality implemented in this task fails.		
Scale Values' Definitions	Low	No critical data will be lost.	⇩
	Medium	Some business data may be lost causing a day's work set back.	
	High	Business data will be lost causing a major set back	
Mean Time to Failure, MTTF			
Definition	The degree of importance for the particular component/functionality being implemented to rarely fail.		
Scale Values' Definitions	Low	It may fail often (once every few days).	⇩
	Medium	It may fail between once a month to once in 3 months.	
	High	It should not fail more than once in six months.	
Mean Time to Repair, MTTR			
Definition	The degree of importance that the particular component/functionality		

	being implemented in the task has a short repair time.		
Scale Values' Definitions	Low	It is not a significant problem if it is down for a week or less.	⇩
	Medium	It is important that it is not down for more than a day.	
	High	It is very important that it is not down for more than an hour.	

Table 4.4 – Example definition of the Reliability sub-attributes.

4.1.2 Data Collection

Upon the completion of the first step, data collection can begin. Step 2 of the NFEM preparation process cannot commence until sufficient data has been collected. However, the question becomes what a sufficient number of data points is. In [6], the authors show that the number of training data points required for a neural network that contains *W* weighted connections is given by:

$$m > \frac{W}{\varepsilon}$$

where *m* is the number of training data points and ε is the allowed fraction of error on the training set. If ε is assumed to be less than 0.125 then approximately ten training data points are required for each weighted connection in the neural network. Therefore, given that an organization knows the largest network architecture to be used in step 3 of the preparation process, it must collect at least ten times the amount of weighted connections within it.

4.2 Step 2: Data Set Separation

The second step of the NFEM preparation process is the data set separation. To be able to extract rules from a trained neural network using the rule-extraction technique described in Section 2.4.3, the data with which the neural network is trained must be Boolean as opposed to continuous. Subsection 4.2.1 describes how the range of values for the qualitative attributes should be separated into fuzzy sets, and then Boolean sets. Subsection 4.2.2 describes how the range of values for the quantitative attributes should be separated into Boolean sets, and then fuzzy sets. It should be noted that the data set separation process must also be applied to the output attribute, effort.

4.2.1 Data Set Separation for the Qualitative Profiling Attributes

Section 4.1.4 described that the sub-attribute values, ranging from one for Low to five for High, should be averaged to obtain an overall value for the profiling attribute. Thus, each qualitative profiling attribute can have a value between one and five and when these values are normalized, the range of values becomes 0.2 to one. The normalization occurs because neural networks and ANFIS training functions implemented by MATLAB only use normalized data [33]. Table 4.5 depicts an example data point profiled by two profiling attributes with normalized values.

Qualitative Profiling Attribute 1 Value	Qualitative Attribute 2 Value	Data Point
0.633	0.25	[0.633 0.25]

Table 4.5 – A sample data point.

4.2.1.1 Determining the Boundaries of the Fuzzy Sets

Each qualitative attribute is separated into three fuzzy set, Low, Medium, and High, as shown in Figure 4.2. The membership function of each fuzzy set is a generalized bell function. The generalized bell function was selected as the membership function of choice for two reasons: It is nonlinearly smooth and it offers three adaptable parameters with which the shape of the function is customized during the ANFIS training [22]. Most other nonlinearly smooth functions offer only two variable parameters [22]. The boundaries of the fuzzy sets Low and Medium are determined such that the cross-over point for each set is at the Medium-Low value two, or 0.4 when normalized. Likewise, the boundaries of the fuzzy sets Medium and High are determined such that the cross-over point for each set is at the Medium-High value four, or 0.8 when normalized. The cross-over point of a fuzzy set A is any point where $\mu_A(x) = 0.5$. The rationale for locating the fuzzy set cross-over points at these values is simple and straightforward: The points Medium-Low and Medium-High are designed to be the point halfway between one set and the next one. Therefore it is only logical that they would be used as the cross-over points of the fuzzy sets. Figure 4.2 illustrates this concept by showing that the cross-over points of the fuzzy sets meet at the Medium-Low and Medium-High values. This condition determines the values for parameters *a* and *b* of the generalized bell function (described in Section 2.1.2.2). In addition, the centre of each of the membership function (i.e. parameter *c* of the generalized bell function) is located at the corresponding value which it represents (illustrated by the dashed lines). For example, the centre of the generalized bell function representing fuzzy set Low is at 0.2 (or one when not normalized).

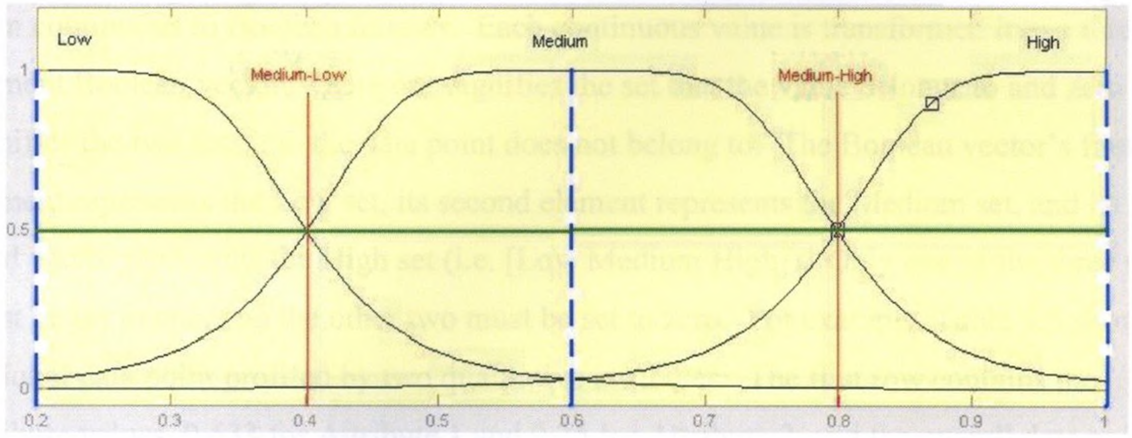


Figure 4.2 – The fuzzy sets of qualitative profiling attributes.

4.2.1.2 Determining the Boundaries of the Boolean Sets

After determining the fuzzy sets, the Boolean sets can also be determined. The boundaries of the Boolean sets should be located at the cross-over points of each fuzzy set. Figure 4.3 illustrates the Boolean sets determine by the crossover points of the fuzzy sets. The points at the boundaries should be included in the set that has the smaller range of values included. Therefore, data points with qualitative attribute values of 0.4 should be included in the Low set because the Low Boolean set only includes points between 0.2 and 0.4, whereas the Medium Boolean set includes data points between 0.4 and 0.8. For the same reasons, the data points with qualitative values of 0.8 should be included in the High Boolean set.

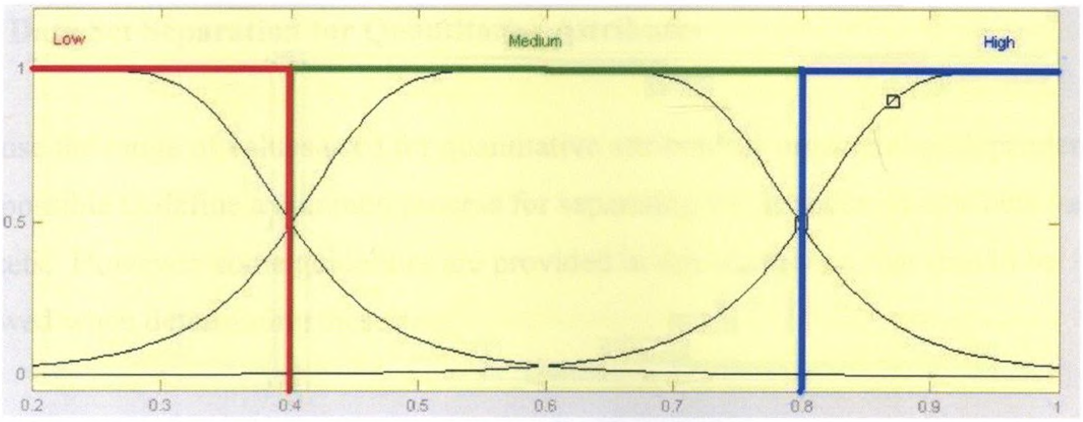


Figure 4.3 – The Boolean sets of qualitative profiling attributes.

Once the boundaries of the Boolean sets are determined, the data value transformation from continuous to Boolean follows. Each continuous value is transformed into a three-element Boolean vector, where one signifies the set that the value belongs to and zero signifies the two sets that the data point does not belong to. The Boolean vector's first element represents the Low set, its second element represents the Medium set, and its third vector represents the High set (i.e. [Low Medium High]). Only one of the three sets must be set to one, and the other two must be set to zero. For example, Table 4.6 shows a fictional data point profiled by two qualitative attributes: The first row contains the attribute values, 0.633 for Attribute 1 and 0.25 for Attribute 2, and the overall data point in continuous vector format ([0.633 0.25]). The second row contains the Boolean transformation of the data point shown in the first row. The Attribute 1 value has been transformed into [0 1 0] because the value 0.633 falls within the bounds of the Medium set. The Attribute 2 value has been transformed into [1 0 0] because the value 0.25 belongs to the Low set.

Format	Qualitative Profiling Attribute 1 Value	Qualitative Attribute 2 Value	Data Point
Continuous	0.633	0.25	[0.633 0.25]
Boolean	[0 1 0]	[1 0 0]	[[0 1 0] [1 0 0]

Table 4.6 – Conversion of a continuous data point into Boolean format.

4.2.2 Data Set Separation for Quantitative Attributes

Because the range of values used for quantitative attributes is organization-dependent, it is impossible to define a common process for separating the quantitative attribute values into sets. However, some guidelines are provided in this subsection that should be followed when determining these sets.

4.2.2.1 Determining the Boundaries of the Boolean Sets

First and foremost, each Boolean set must contain a sufficient number of data points within it so that the neural network is able to associate certain input values with that particular set. Section 4.5.2 discussed what a sufficient number of data points is overall. If only the minimum amount is collected (i.e. ten times the amount of weighted connections in the largest neural network architecture to be used), the quantitative attribute Boolean sets can be separated such that each set has an approximately equal amount of data. If the amount of collected data greatly exceeds the minimum required amount, then a criterion other than equal amount of data can be used to separate the sets. For example, self-organized maps [30] or clustering algorithms [5] can be used to determine the sets. These topics however are outside the scope of this thesis.

In most cases, due to the fact that it takes a long time to collect data, most companies will start implementing the second step of the preparation process once they have gathered what is considered sufficient data. Therefore, the Boolean sets would need to be separated based on an equal-data amount criterion. Due to this, the boundaries of the Boolean sets must be determined first, so as to ensure that each set with which the neural network is trained contains enough data for the neural network to learn with.

4.2.2.2 Determining the Boundaries of the Fuzzy Sets

Once the Boolean sets' boundaries are determined, the fuzzy set membership functions can be determined for the quantitative profiling attributes' sets. The generalized bell is maintained as the membership function of choice for the advantages it provides. The width of each bell membership function, which is controlled by the parameter *a*, should be equal to the width of the Boolean set that corresponds to it. Additionally, the centre of the generalized membership function of each fuzzy set, controlled by the parameter *c*, should be located at the median value of the set it represents. For example, to determine where the centre of the generalized bell membership function of the fuzzy set Low should be located, the median of the data values contained in the Boolean set Low should be used. The reason why the median was chosen as the centre of the generalized bell membership functions, as opposed to the mean, was because the finite breakdown point of the median is much higher than that of the mean [53]. The finite breakdown point is the smallest proportion of outliers that can result in the mean or the median being arbitrarily large or small for a given set of observations [53]. Given a set of n observations, the finite breakdown point of the mean is $1/n$ whereas the finite breakdown point of the median is $n/2$ [53]. Therefore, for a set containing 50 data points, only 1/50, or 1 of the data points need to be outliers, in order to produce an arbitrarily large or small mean. Conversely, the proportion of outliers required to bias the median, for the same set of data points, would be 50/2, or 25 data points.

Finally, for the quantitative output attribute, Effort, no fuzzy sets are required because the zero-order Sugeno inference system implemented in the ANFIS does not require fuzzy sets for the output attribute. Instead, one constant value must be selected to represent each output set. For the same reasons discussed above, the median should be the choice representative value for each output attribute set.

4.3 Step 3: Neural Network Training

Once a sufficient amount of data has been collected and the set separations have been accomplished, a neural network can be trained with the Boolean data. This third step of the NFEM preparation process consists of some trial and error due to the many different factors that can determine how successfully a neural network is trained. In this section some of the most important factors are discussed.

4.3.1 Varying the Inputs

The most important factors are the inputs and outputs used: The stronger the relationship between the inputs and the outputs, the easier it is for the neural network to learn the relationship [41]. In order to determine the combination of profiling attributes that have the greatest affect on development effort, different combinations of the inputs must be tested. The number of profiling attributes collected and the number of sets defined for each attribute determine the number of inputs to the network. For example, if three profiling attributes are selected and the values of each one are separated into three sets, Low, Medium, and High, then the neural network will have 9 inputs.

4.3.2 Varying the Number of Hidden Nodes

The next factor that affects the neural network training phase is the number of nodes used in the middle layer of the network, often referred to as the hidden nodes. As the number of nodes increases in the hidden layer, so does the accuracy of a neural network to predict the output of the training data [36]. However, if the number of hidden nodes is too high, the network loses its ability to generalize, and models itself too closely to the training data. Consequently, the network performs well when the training data is used, but it performs poorly when new data is entered. This phenomenon is often called "over-fitting" [36]. While several methods and parameters have been proposed to determine the number of hidden layer nodes [17], [35], [36], this problem lies beyond the scope of this thesis. It is suggested that a trial and error process is followed to determine the number of hidden neural nodes. It is also best to separate the historical dataset into a training data set and a testing data set. Training several different networks with varying numbers of hidden nodes, and then testing them, will show which architecture yields the best results in accurately predicting training as well as testing data.

4.3.3 Varying the Number of Training Epochs

Finally, the number of training epochs can also be varied to see if training the neural networks with more epochs yields significantly more accurate results. Although it has been shown that most networks are successfully trained with 1000 epochs, this number can sometimes vary [9].

Figure 4.4 illustrates an example network architecture where two profiling attributes are used as inputs, each one consisting of three sets, and the classification attribute is separated into five sets. Eight hidden nodes are used in the middle layer, where the value eight was chosen arbitrarily.

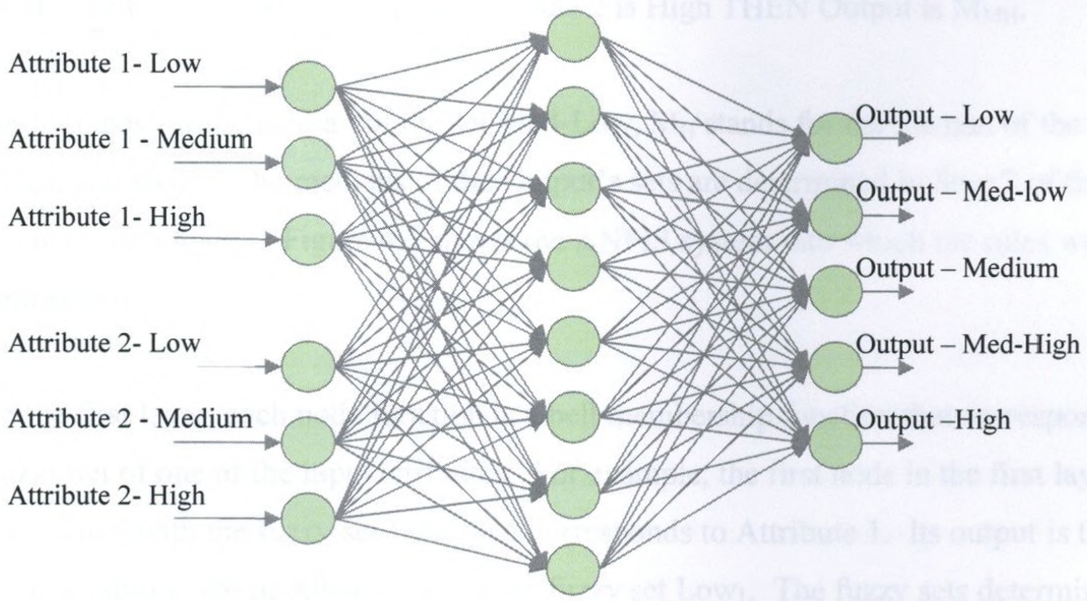


Figure 4.4 – Example architecture of a neural network used in Step 3.

4.4 Step 4: Rule Extraction and ANFIS Implementation

The fourth and final step of the NFEM preparation process is the rule extraction and ANFIS implementation. Upon its completion, the NFEM can be used as a neuro-fuzzy effort estimator for software development tasks.

The rule extraction technique described in Section 2.4.3 is used to extract rules from the neural network that was most successfully trained in Section 4.3. More specifically, the most general rules are extracted from its hidden layer neurons and its output layer neurons, and then combined to create rules modeling the relationship between the profiling attributes and the effort output. These rules are then embedded into an ANFIS system. For example, suppose the following five rules were extracted from the neural network shown in Figure 4.4:

1. IF Attribute 1 is Low AND Attribute 2 is Medium THEN Output is M_{ML}.
2. IF Attribute 1 is High THEN Output is M_H.
3. IF Attribute 2 is Low THEN Output is M_L.
4. IF Attribute 1 is Medium AND Attribute 2 is Low THEN Output is M_M.
5. IF Attribute 1 is Medium AND Attribute 2 is High THEN Output is M_{MH}.

M_{ML} stands for the median of the set Med-Low, M_H stands for the median of the set High, and so on. The medians of the Output's sets are determined in Step 2 of the NFEM preparation process. Figure 4.5 shows the ANFIS system into which the rules would be embedded.

In the first layer, each node function is a bell membership function that corresponds to a fuzzy set of one of the input attributes. For example, the first node in the first layer is associated with the fuzzy set Low_1 that corresponds to Attribute 1. Its output is the membership grade of Attribute 1 into its fuzzy set Low_1. The fuzzy sets determined in Step 2 of the NFEM preparation process are used as the node functions of the first layer neurons. Each neuron in the second layer corresponds to the antecedent of one of the rules. For example, the first neuron of the second layer corresponds to the antecedent of the first rule, A_{R1}. Its inputs are the membership grade of Attribute 1 into its fuzzy set Low_1 and the membership grade of Attribute 2 into its fuzzy set $Medium_2$. The output is the firing strength of the first rule, as described in Section 2.3.

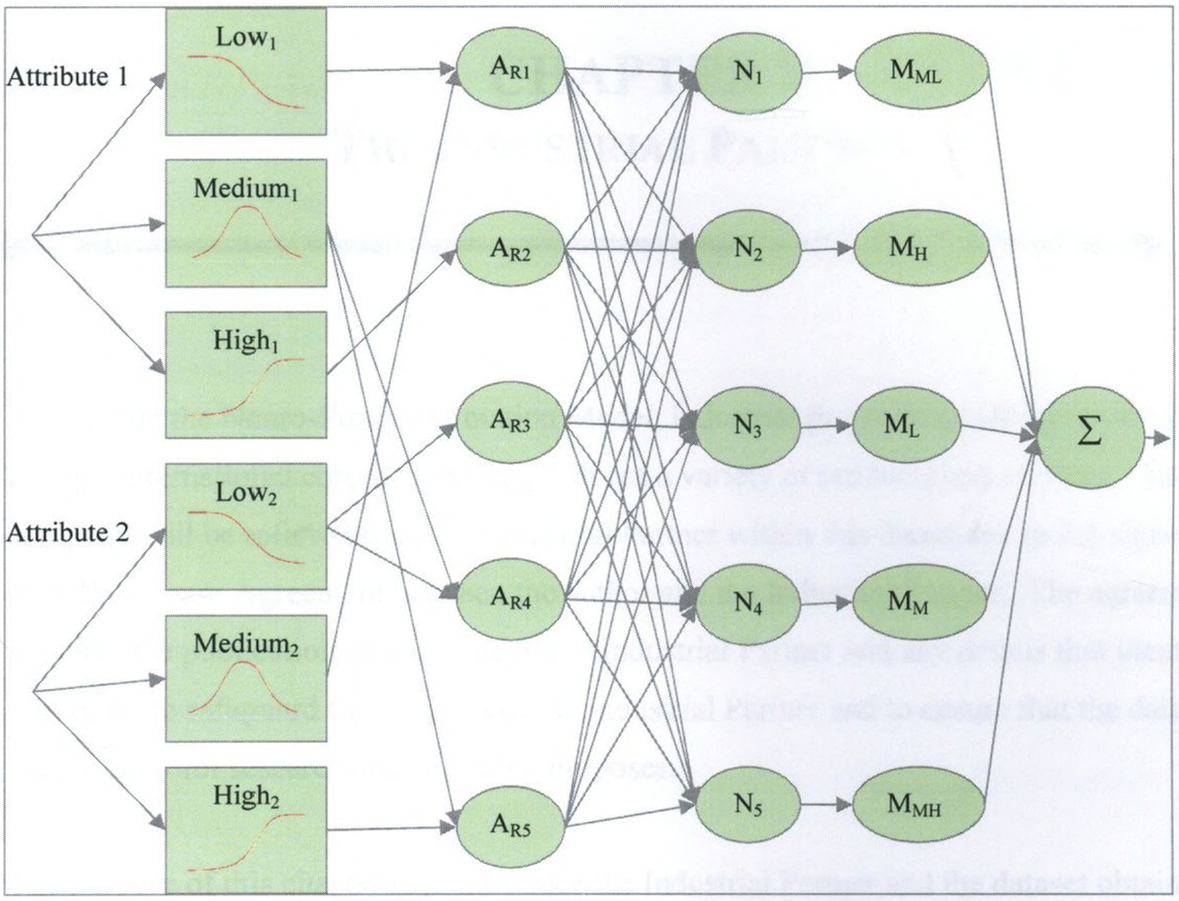


Figure 4.5 – An ANFIS system upon which the NFEM is implemented.

In the third layer, each neuron *i* calculates and outputs the normalized firing strength of the rule represented by neuron *i* in the second layer. Likewise, each neuron *i* in the fourth layer corresponds to the consequent of the rule represented by node *i* in the second layer. The output of each neuron in the fourth layer is the product of the consequent of the rule it represents with the rule's firing strength. In the fifth layer, the outputs of the fourth layer's neurons are summed and output as the effort estimate.

Once the fuzzy sets, medians, and rules are embedded into the ANFIS, the historical data collected after step 1 of the preparation process is used to train the ANFIS. By training the ANFIS with the historical data, the bell shaped fuzzy sets of the input profiling attributes are fine-tuned by having their *a*, *b*, and *c* parameters, representing the width, slope of the bell's sides and centre location, change according to the historical data. Upon the completion of the training, the ANFIS, now representing the Neuro-Fuzzy Estimation Model is ready to be used.

CHAPTER V
THE INDUSTRIAL PARTNER

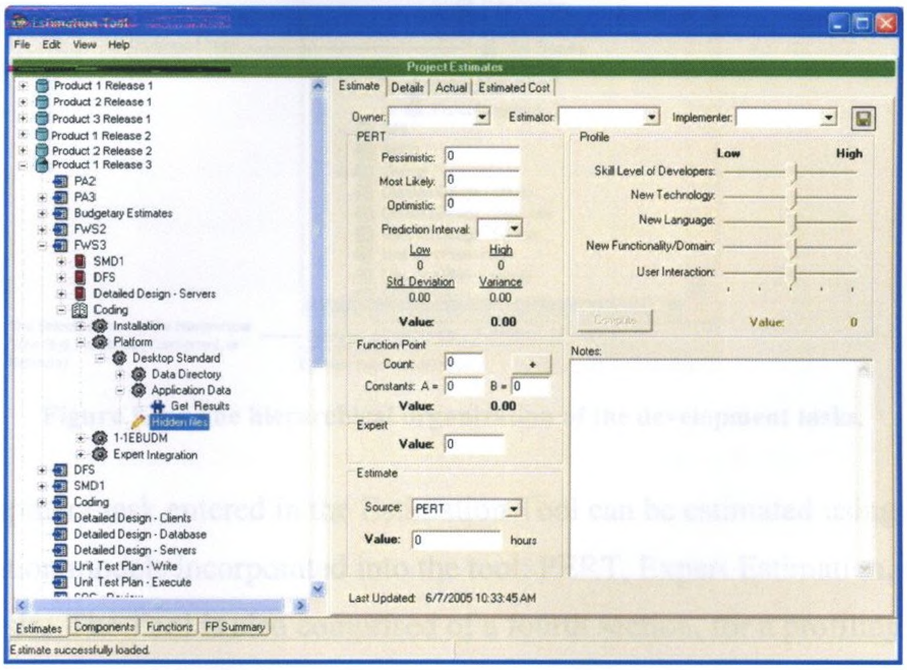
To validate the Neuro-Fuzzy Estimation Model, industrial project data was obtained from a major international corporation that produces a variety of products and services. This company will be referred to as the Industrial Partner within this thesis due to the signed Non-Disclosure Agreement between the author and the Industrial Partner. The agreement prevents the publication of the name of the Industrial Partner and any details that identify it in order to safeguard the interests of the Industrial Partner and to ensure that the data is used strictly for research and academic purposes.

The purpose of this chapter is to introduce the Industrial Partner and the dataset obtained from them. Section 5.1 describes the Industrial Partner and Section 5.2 describes their current process of software development effort estimation. Section 5.3 presents their future goals in the area of effort estimation. Section 5.4 contains the use case developed to improve the Industrial Partner's estimation capabilities by integrating the NFEM into their estimation process. Section 5.5 describes the dataset provided by the Industrial Partner and used to validate the NFEM in Chapter 6.

5.1 The Industrial Partner and Their Current Effort Estimation Process

The Industrial Partner is a Fortune 500 company with a workforce of over 100,000 people world-wide and annual revenue of over $30 billion dollars. It is involved in the production of products and services that cater to a wide variety of customers ranging from private consumers to major Industrial Partners. The company is known for its aggressive implementation and daily practice of the six sigma methodology which is focused on reducing errors and costs.

In the area of software development, the six sigma effort has translated into the development of a custom and comprehensive estimation application (Figure 5.1) by one of the software development teams of the Industrial Partner, hereon referred to as the Estimation Tool. The Estimation Tool allows for the development tasks associated with a project to be displayed in a hierarchical tree structure. For each project, there are three different views of the tree structure available: The functional tree, the component tree, and the estimation tree (refer to Figure 5.2). The functional tree provides a hierarchical decomposition of the functionality within a software application being developed, and the development tasks associated with each of those functionalities. The component tree provides a hierarchical decomposition of software components that are part of a software application in development, and the development tasks associated with each component. And finally, the estimation tree references both the functional and component trees, to provide the complete list of development tasks that map to the software development lifecycle.



Owner:
Estimator:
Implementer:

PERT
Pessimistic: 0
Most Likely: 0
Optimistic: 0
Prediction Interval:
Low 0 High 0
Std. Deviation 0.00 Variance 0.00
Value: 0.00

Profile
Low High
Skill Level of Developers:
New Technology:
New Language:
New Functionality/Domain:
User Interaction:
Value: 0

Function Point
Count: 0
Constants: A = 0 B = 0
Value: 0.00

Expert
Value: 0

Estimate
Source: PERT
Value: 0 hours

Last Updated: 6/7/2005 10:33:45 AM

Estimate successfully loaded.

Figure 5.1 – The Estimation Tool developed by the Industrial Partner.

Figure 5.2 illustrates the estimation tree view. In it, each release is identified as a new node on the tree, representing a new project and corresponds to the Project Node in

Figure 5.2. After every completed deliverable phase of the project, the team re-estimates the development tasks of the future phases. Therefore, each project is broken down into these re-estimation units, which are called sources and are represented as source nodes in the hierarchical tree shown in Figure 5.2. The source nodes are then broken down into future deliverable phase nodes, and furthermore, each deliverable phase node is decomposed into function nodes. Function nodes can either be decomposed into other function and component nodes or can be leaf nodes. Leaf function nodes and component nodes represent development tasks that must be estimated.

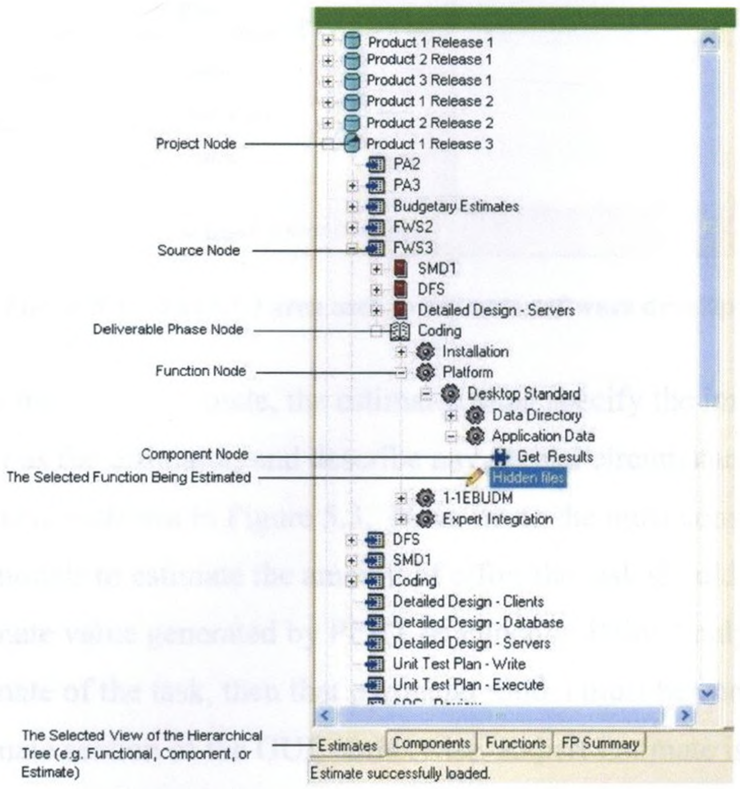
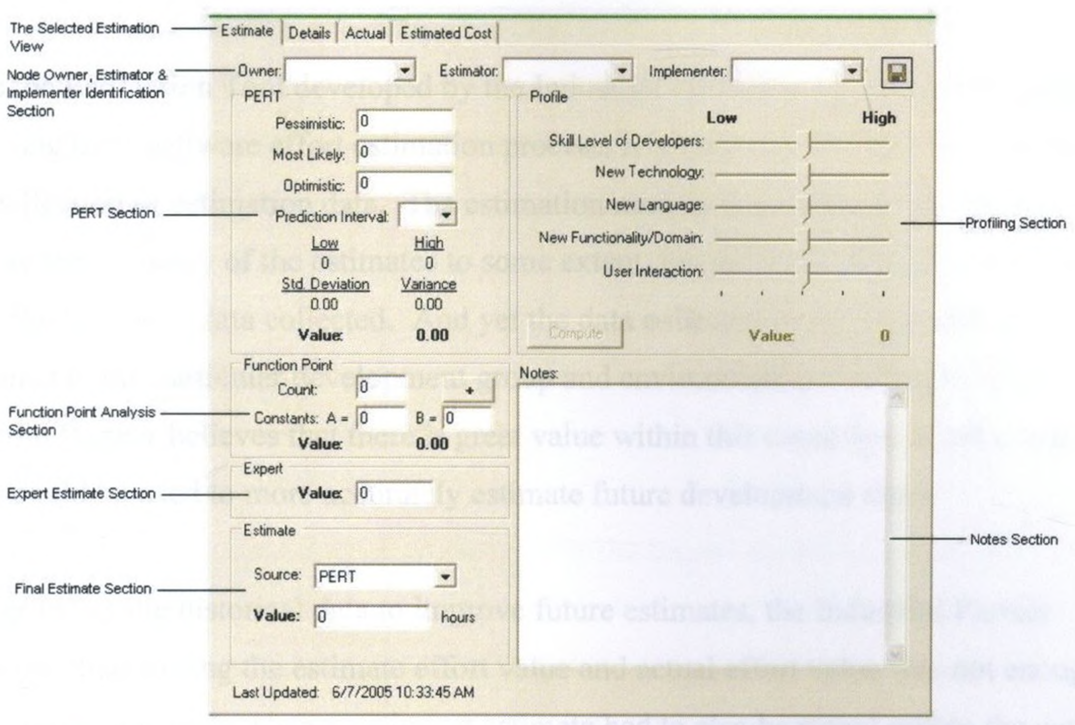


Figure 5.2 – The hierarchical organization of the development tasks.

Each development task entered in the Estimation Tool can be estimated using one of the three estimation models incorporated into the tool: PERT, Expert Estimation, or Function Point Analysis. The Tool is also comprised of a fourth section, for a profiling technique; however, at the time of the Estimation Tool development, only the graphical user interface (GUI) of this technique was implemented, rendering the section useless. Figure 5.3 depicts the GUI section of the Estimation Tool that provides the functionality for estimating a task.



Owner:
Estimator:
Implementer:

PERT
Pessimistic: 0
Most Likely: 0
Optimistic: 0
Prediction Interval:

Low	High
0	0

Std. Deviation	Variance
0.00	0.00

Value: 0.00

Profile
Low High
Skill Level of Developers:
New Technology:
New Language:
New Functionality/Domain:
User Interaction:
Value: 0

Function Point
Count: 0
Constants: A = 0 B = 0
Value: 0.00

Expert
Value: 0

Estimate
Source: PERT
Value: 0 hours

Notes:

Last Updated: 6/7/2005 10:33:45 AM

Figure 5.3 – The GUI area used to estimate software development task effort.

When entering a new estimate, the estimator must specify the implementer of the task, him/herself as the estimator, and describe any special circumstances or assumptions in the notes section shown in Figure 5.3. Next, he or she must consider one or more of the available models to estimate the amount of effort the task should take, in hours. If the effort estimate value generated by PERT or Function Point Analysis is chosen as the effort estimate of the task, then that particular model must be chosen as the source in the Final Estimate section of the GUI. Otherwise, Expert Estimate is chosen as the source, and the value entered in the Expert Estimate Section is recorded as the effort estimate for the task. When the task is completed, the actual time the task took must also be recorded. This value is referred to as the *actual effort* value. An estimation task that contains all of the above information represents one data point. All such data points are stored and maintained in a centralized database, which allows easy access to the team through the client application, without easily risking data corruption.

5.2 The Industrial Partner's Future Effort Estimation Goals

While the Estimation Tool developed by the Industrial Partner is a great start towards improving their software effort estimation process, it is only a tool that allows for the easy collection of estimation data. The estimation models incorporated into the tool may improve the accuracy of the estimates to some extent, but neither of these models makes use of the historical data collected. And yet the data collected is very valuable as it is fine tuned to the particular development group and environment collecting the data. The Industrial Partner believes that there is great value within this collection of data, value which could be used to more accurately estimate future development tasks.

In order to use the historical data to improve future estimates, the Industrial Partner recognized that storing the estimate effort value and actual effort value was not enough. The assumptions made when creating an estimate had to also be stored within the data point. These assumptions would need to follow a specific and consistent format, allowing each task to be characterized by the same parameters. This would enable future tasks to be related to historical tasks. In fact, this idea was what inspired the development team of the Estimation Tool to include the GUI implementation of the profiling technique into the tool. However, no further research was conducted as to which attributes should be used to profile the estimation tasks and no algorithm was implemented to make use of the historical data. The need for this common profile and the existing collection of estimation data made the Industrial Partner an ideal partner for the validation of the NFEM. A use case was developed to summarize NFEM's outcome as desired by the Industrial Partner and is shown in the following section.

5.3 Use Case for Generating an Effort Estimate Based on Historical Data

This use case describes the scenario, where given a profile, an effort estimate value is computed by the Estimation Tool based on the actual effort values of historical tasks with similar profiles, using the NFEM.

Scope: Generation of a software development task effort estimate using the Profiling Section on the Estimation Tool of the Industrial Partner (refer to Figure 5.3)
Primary Actor: Software Estimator
Preconditions: An estimate node has been created in the Estimation Tool and the estimate tab is open for editing.
Scenario:

1. Software Estimator selects the appropriate person for the following fields: Owner, Estimator, and Implementer.
2. Software Estimator enters details, concerning the task being estimated, into the notes field.
3. Software Estimator creates an estimate in the Expert Estimate section of the Estimation Tool. The Expert Estimate can be based on values generated by the Function Point Analysis and/or PERT methods.
4. Software Estimator selects values for each of the attributes in the Profiling section of the Estimation Tool.
5. Software Estimator presses the "Compute" button in the Profiling section and a calculated effort estimation value is displayed in the "Value" field of the Profiling section.
6. Software Estimator analyses the values generated by any or all of the PERT, Function Point, Expert and Profiling models.
7. Software Estimator selects one of the models as the source in the Final Estimate section.
8. Software Estimator either uses the value auto-generated by selecting the source or enters a number in the Final Estimate section of the Estimation Tool that takes into account the values generated by each of the four estimation methods.

5.4 The Industrial Partner's Dataset

The data obtained from the Industrial Partner was collected by a software development team in charge of developing software for a corporate client. The data was collected during the development of three different products, and two releases of each, totaling six projects. The development team consisted of an average of six people, and the team was located on two different continents. Due to the need to deliver high quality software products on time, the Estimation Tool was an integral part of the development process.

Originally, the Industrial Partner estimated to have approximately 2000 historical data points. No concrete value was known because estimates and actual values of historical tasks completed before the development of the Estimation Tool were scattered in different spreadsheet files. Therefore, the data had to be centralized before knowing the true value of the number of data points. Once the data was centralized into one common database, it was discovered than only about 1400 data points existed. Subsequently, data points that contained estimates but no actual values (i.e. bad data), and data points of tasks that did not belong to the implementation phase were also filtered out. Non-implementation tasks were filtered out because finding a common set of attributes for tasks of all phases of the software development cycle would be difficult if not impossible. On the other hand, time limitations made finding a common set of attributes for each phase unfeasible.

Furthermore, all implementation tasks with estimated or actual effort size of over 100 hours or magnitude of relative error greater than 50% were filtered out, leaving only 313 data points. The reason for filtering out tasks with a MRE greater than 50 was that allowing a large range of MRE values required a very high volume of data points to train the neural network. Due to the fact that a high volume of data was not available, the MRE range was limited. Finally, tasks with estimated or actual effort size over 100 hours were filtered out because they were quite rare, and therefore the few data points that did exist would bias the neural network into creating an input-output relationship that was incorrect.

CHAPTER VI
CASE STUDY

In this chapter, the Industrial Partner's historical data is applied to validate the Neuro-Fuzzy Estimation Model. Due to the long term process involved in collecting large amounts of software development effort estimation data, time constraints did not allow for the full validation of the NFEM. However, because some existing historical data was available, the steps that were applied presented a promising outcome.

In Section 6.1, the first step of the NFEM preparation process, the selection and definition of the profiling attributes, is completed followed by a discussion on the selected attributes. Next, Section 6.2 describes the second step of the NFEM preparation process, the data set separation step. In Section 6.3, step 3 of the NFEM preparation process, the neural network training with the Industrial Partner's dataset, is completed and the training results are analyzed. Finally, Section 6.4 describes the implementation of the last step of the NFEM preparation process of extracting the rules from the trained neural networks and implementing them in the ANFIS.

6.1 Step 1: Profiling Attribute Selection

As described in Chapter 4, the first step to the NFEM preparation process is determining the set of profiling attributes. These attributes would be used to characterize the Industrial Partner's dataset. It was decided that a set of attributes would be determined that would be used to profile only implementation tasks for the reasons discussed in Section 5.4.

These attributes were selected for their usefulness, measurability, and significance. They were determined after careful analysis of hundreds of metrics used in existing estimation models or researched internally by the Industrial Partner. In this section they are exemplified and characterized by a measuring scheme. Each selected profiling attribute is defined and described through the sub-attributes into which it was decomposed. While only the definitions of the attributes and sub-attributes are provided in this section, Appendix A contains the full descriptions of each attribute, including the scale value definitions, using the table format shown in Section 4.1.

6.1.1 Skill Level of Implementer

Name	Skill Level of Implementer
Definition	The degree to which the skill level of the task implementer influences the effort estimate.
SUB-ATTRIBUTES	
Analyst Capability	The ability to investigate new strategies or defects, as well as the overall quality, reliability, and robustness of work items previously completed by the task implementer.
Learning Ability	The task implementer's ability to learn new concepts and acquire new skills quickly.
Efficiency	The ability to complete a task accurately and on time (i.e. without over-analyzing the problem and the possible solutions).
Teamwork	The ability to communicate in a timely manner with other team members and management and the ability to co-operate in terms of choosing the best possible solution for the task, while still adhering to time and quality constraints.

Table 6.1 – Description of the Skill Level of Implementer attribute.

6.1.2 Familiarity with Technology

Name	Familiarity with Technology, Technology
Definition	The degree to which the implementer's familiarity with the technology, used to complete the task, influences the estimate.
SUB-ATTRIBUTES	
Familiarity with Documentation	The degree of the task implementer's knowledge /understanding of the technology's documentation: Has the implementer skimmed the documentation or thoroughly read it. Documentation includes: help files, user guides, online tutorials, and books dedicated to the technology.
Usage of Technology	How well the task implementer feels that he/she knows how to implement solutions using the technology. This is a measure of the

	level of comfort he/she has in using the technology.

Table 6.2 – Description of the Familiarity with Technology attribute.

6.1.3 Familiarity with Programming Language

Name	Familiarity with Programming Language, Familiarity with Language, Language
Definition	The degree to which the implementer's familiarity with the software language, to be employed when completing the task, influences the estimate.
SUB-ATTRIBUTES	
Familiarity with Documentation	The degree of the task implementer's knowledge/understanding of the language's support documentation. The documentation includes: help files, user guides, online knowledge databases, and books dedicated to the use of the language.
Usage of Language	How well the task implementer feels that he/she knows how to implement solutions using the chosen language. This is a measure of the level of comfort he/she has in developing solutions with the language.

Table 6.3 – Description of the Familiarity with Programming Language attribute.

6.1.4 User Interface

Name	User Interface, UI
Definition	The degree to which the level of complexity of the user interface influences the estimate.
SUB-ATTRIBTUES	
Amount of UI Controls	A linguistic approximation of the amount of user interface controls needed by the functionality. User interface controls include: text boxes, list boxes, radio buttons, command buttons, menus, combo boxes, etc.
Required Level of Validation	A qualitative measure of the amount of input validation required by the user interface of the task's functionality.
Underlying Architecture Complexity	The overall complexity of the underlying architecture. For example, a simple registry access function will likely have a low architectural complexity whereas functionality providing the ability to insert 3rd party ActiveX controls would likely have a high architectural complexity.

Table 6.4 – Description of the User Interface attribute.

6.1.5 Complexity

Name	Complexity
Definition	The degree to which the complexity of the task influences the estimate.
SUB-ATTRIBUTES	

Difficulty of Definition	The degree of difficulty involved in defining the solution such as the algorithmic complexity of the solution in terms of computational complexity (e.g. nested loops, analysis of differential equations), time computational complexity (e.g. real-time systems), space computational complexity (e.g. distributed database coordination), and information-based complexity (e.g. simple arrays in main memory vs. highly coupled dynamic relational and object structures).
Interdependence with other Features	The amount of other functions/features the current task impacts and/or the amount of functions/features the current task is impacted by.

Table 6.5 – Description of the Complexity attribute.

6.1.6 Familiarity with Functionality

Name	Familiarity with Functionality, Functionality
Definition	The degree to which the implementer's familiarity with the functionality influences the estimate.
SUB-ATTRIBUTES	
Similarity	The degree to which the current task resembles something that the implementer has previously implemented.
Product Knowledge	How familiar the implementer is with the application/product being developed. This will give a measure of how well the implementer understands how the component/functionality will affect the existing components/functionality.
Component Knowledge	How familiar the implementer is with the component the current task involves.

Table 6.6 – Description of the Familiarity with Functionality attribute.

6.1.7 Familiarity with Domain

Name	Familiarity with Domain
Definition	The degree to which familiarity with the application domain influences the estimate.
SUB-ATTRIBUTES	
Product Domain Familiarity	The level of familiarity the implementer has with the application domain (i.e. functions within the industry that the product will be used).
Software Domain Familiarity	The level of familiarity the implementer has with the task's related software domain (e.g. database, GUI, server, web, etc.).

Table 6.7 – Description of the Familiarity with Domain attribute.

6.1.8 Estimated Size

Name	Estimated Size
Definition	The degree to which the size of the task, in hours, influences the estimate.

Table 6.8 – Description of the Size attribute.

6.1.9 Altered Estimation Tool

Figures 6.1 and 6.2 show the altered Profiling Section of the Estimation Tool. Each profiling attribute was implemented as a tab, as shown in Figure 6.2. The first tab contained the overall, read-only profile of a given task being estimated (Figure 6.1).

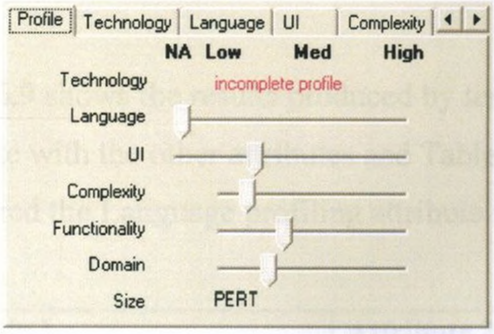


Figure 6.1 – Overall profile tab.

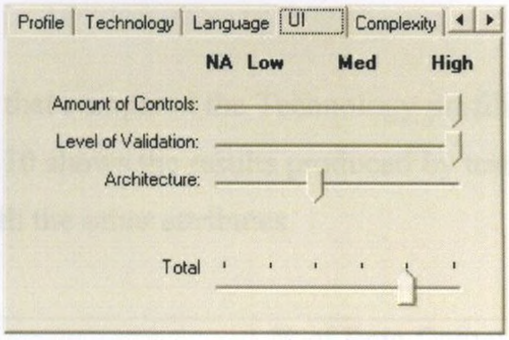


Figure 6.2 – UI profiling attribute tab.

6.1.10 Profiling Attribute Analysis and Discussion

After the profiling attributes were determined, clearly defined, and characterized by the measuring system, they were used to profile the historical implementation tasks of the Industrial Partner. During the profiling process, it became apparent that the attribute Skill Level of Implementer would be difficult to use for the reasons discussed in Chapter 3. As a result, that particular attribute was removed from the profiling set.

Due to low volume of data available to train the neural networks, it was beneficial to have as few inputs as possible. Therefore, after all the historical tasks were profiled, several tests were conducted to see if any of the profiling attributes could be eliminated. Because the User Interface attribute was only applicable to some implementation tasks, it was found that only forty percent of the data made use of it. It was decided that there was insufficient data to correctly evaluate the significance of this attribute. Therefore, the User Interface attribute was eliminated.

In addition, several attributes were found to have equal values for a suspiciously large amount of data points. It was suspected that the members of the development team who profiled the data perceived some of the "Familiarity with" attributes to be very similar and therefore, consistently evaluated these attributes the same. Tests were conducted to evaluate the amount of similarity between the following attributes: Familiarity with Functionality, Familiarity with Technology, Familiarity with Domain, Familiarity with Programming Language, and Complexity. The Complexity attribute was only included to serve as a comparison measure.

Table 6.9 shows the results produced by tests that compared the Technology profiling attribute with the other attributes and Table 6.10 shows the results produced by tests that compared the Language profiling attribute with the other attributes.

Attribute 1	**Attribute 2**	**% of Data Containing the Same Values for Attributes 1 and 2**
Familiarity with Technology	Familiarity with Language	55.7%
Familiarity with Technology	Familiarity with Domain	34.7%
Familiarity with Technology	Familiarity with Functionality	22.3%
Familiarity with Technology	Complexity	10.8%

Table 6.9 – Similarity between Technology and other attributes.

Attribute 1	**Attribute 2**	**% of Data Containing the Same Values for Attributes 1 and 2**
Familiarity with Language	Familiarity with Technology	55.7%
Familiarity with Language	Familiarity with Domain	31.5%
Familiarity with Language	Familiarity with Functionality	16.9%
Familiarity with Language	Complexity	10.8%

Table 6.10 – Similarity between Language and other attributes.

Both tables show that over 55% of the dataset contained the same values for the attributes Familiarity with Technology and Familiarity with Language. This was definitely an abnormal amount of similarity between the two attributes, and therefore one of them had to be eliminated. It was determined that the attribute which showed the highest similarity with other attributes would be eliminated.

At 34.7%, the similarity between the Technology and the Domain attributes was higher than the similarity between the Language and Domain attributes, which was 31.5%. Furthermore, at 22.3%, the similarity between the Technology and Functionality attributes was also higher than the similarity between the Language and Functionality attributes, which was 16.9%. Therefore, the Technology attribute was more often perceived to be the same as the Domain and Functionality attributes, when compared to the Language attribute. As a result, the Familiarity with Technology attribute was eliminated.

Attribute 1	**Attribute 2**	**% of Data Containing the Same Values for Attributes 1 and 2**
Familiarity with Domain	Familiarity with Language	31.5%
Familiarity with Domain	Familiarity with Functionality	25.8%
Familiarity with Domain	Complexity	15%

Table 6.11 – Similarity between Domain and other attributes.

Additionally, the Familiarity with Domain attribute showed to be the same as the Familiarity with Language attribute for over 30% of the data. Likewise, the Domain attribute was evaluated the same as the Familiarity with Functionality attribute for over 25% of the data (as shown in Table 6.11). This showed that the Domain attribute was also often perceived to be the same as the Functionality and Language attributes, and it was therefore also eliminated.

The remaining attributes of Functionality and Language had a low degree of similarity between them, at 16.9% (shown in Table 6.10), and were therefore retained. Thus, the Functionality, Language, Estimated Size, and Complexity attributes were used as inputs in the training of the neural networks.

6.2 Step 2: Data Set Separation

The next step in the NFEM preparation process was to separate the profiling attributes into sets so that rules could be extracted from the trained neural networks and be implemented into an ANFIS. As described in Chapter 4, the NFEM was designed in such a way that it facilitates the data set separation for qualitative and quantitative profiling attributes.

This section describes the properties of the profiled data before applying the data set separation process described in Section 4.2 to the qualitative profiling attributes, and subsequently, to the quantitative attributes.

6.2.1 Experimental Data

Each data point consisted of four profiling attributes (Complexity, Functionality, Language, and Estimated Size) and the output attribute, Effort, in hours. The qualitative profiling attributes Functionality, Language, and Complexity could take on any value between one and five, and when normalized, this became a range between 0.2 and one (the value zero was reserved to indicate the term "not applicable"). The attribute value depended on how the sub-attributes were evaluated. The Estimated Size attribute as well as the Effort attribute could take any positive value between zero and one hundred. All the attribute values were normalized to be between 0 and 1. Table 6.12 illustrates three sample data points.

Complexity	**Estimated Size (in hours)**	**Familiarity with Functionality**	**Familiarity with Language**	**Effort (in hours)**
0.4	0.22	0.2	0.2	0.15
0.9	0.6533	0.4	0.8	0.98
0.2	0.1	0.87	0.7	0.2

Table 6.12 – Sample data points obtained from the Industrial Partner.

6.2.2 Data Set Separation for the Qualitative Profiling Attributes

6.2.2.1 Determining the Boundaries of the Fuzzy Sets

The procedure described in Chapter 4 was followed to separate the possible range of values into three fuzzy sets, Low, Medium, and High, which could then be transformed into Boolean sets. As described in Section 4.2, the values Medium-Low and Medium-High were used as references to locate the crossover points of each fuzzy set.

6.2.2.2 Determining the Boundaries of the Boolean Sets

Once the fuzzy sets were determined, boundaries of the Boolean sets were located at the crossover points of each fuzzy set. The Low set contained any values greater than or equal to 0.2 and less than or equal to 0.4 and was represented with the vector [1 0 0]; the Medium set contained any values greater than 0.4 and less than 0.8 and was represented with the vector [0 1 0]; and the High set contained any values greater than or equal to 0.8 and was represented with the vector [0 0 1]. Table 6.13 reflects the Boolean transformation of the qualitative profiling attributes of the sample data shown in Table 6.12.

Complexity		Functionality		Language	
Continuous	Boolean	Continuous	Boolean	Continuous	Boolean
0.4	[1 0 0]	0.2	[1 0 0]	0.2	[1 0 0]
0.9	[0 0 1]	0.4	[1 0 0]	0.8	[0 0 1]
0.2	[1 0 0]	0.866667	[0 0 1]	0.7	[0 1 0]

Table 6.13 – Sample qualitative attributes transformed into Boolean form.

6.2.3 Data Set Separation for the Quantitative Attributes

6.2.3.1 Determining the Boundaries of the Boolean Sets

The remaining profiling attribute, Estimated Size, and the output attribute, Effort, were not by default separated into sets as they were quantitative in nature. Due to the lack of abundance in data points, the equal data amount criterion was used to first determine the Boolean sets and then the fuzzy sets.

For the Estimated Size profiling attribute, the sets into which the data was separated into are shown in Table 6.14. To illustrate the equal data principle that guided the boundaries of the sets, the percentage of data points within each set is also shown in Table 6.14.

Set Name	**Range of Values included in set (in hours)**	**% of Data Points within the Set**	**Median**	**Boolean Vector**
Low	0<Estimated Size<4	18.2%	1.83	[1 0 0 0 0 0]
Medium-Low	4<= Estimated Size <8	17.9%	4.83	[0 1 0 0 0 0]
Medium	8<= Estimated Size <11	17.9%	8	[0 0 1 0 0 0]
Medium-High	11<= Estimated Size <17	16.0%	16	[0 0 0 1 0 0]
High	17<= Estimated Size <30	15.0%	24	[0 0 0 0 1 0]
Very High	Estimated Size >=30	16.6%	42	[0 0 0 0 0 1]

Table 6.14 – The Estimated Size Boolean sets.

Next, the Boolean sets of the output attribute were determined. The Effort values were separated into six sets, where each set was defined by a range of hours. Once again, the equal-data-amount criterion was used to determine the sets. Table 6.15 shows the boundaries of each set and the percentage of data points that fall within each set. Table 6.16 contains the full Boolean transformation of a sample data point.

Set Name	**Range of Values included in set (in hours)**	**% of Data Points within the Set**	**Median**	**Boolean Vector**
Low	0< Effort<4	16.0%	2	[1 0 0 0 0 0]
Medium-Low	4<= Effort <8	16.9%	5	[0 1 0 0 0 0]
Medium	8<= Effort <11	14.1%	8	[0 0 1 0 0 0]
Medium-High	11<= Effort <17	17.9%	15	[0 0 0 1 0 0]
High	17<= Effort <30	16.0%	22	[0 0 0 0 1 0]
Very High	Effort >=30	19.2%	41	[0 0 0 0 0 1]

Table 6.15 – The Boolean sets of the output attribute Effort

Attribute	**Continuous Format**	**Boolean Format**
Complexity	0.4	[1 0 0]
Estimated Size	0.22	[0 0 0 0 1 0]
Functionality	0.2	[1 0 0]
Language	0.2	[1 0 0]
Effort	0.15	[0 0 0 1 0 0]

Table 6.16 – Data point transformed from continuous to Boolean format.

6.2.3.2 Determining the Boundaries of the Fuzzy Sets

Once the Boolean sets were determined, the fuzzy sets could be determined for the profiling attribute Estimated Size. Again, the procedure described in Section 4.2.2 was followed to do so. For each set of Table 6.14, the median shown was used as the centre of the generalized bell membership function and the width of the membership function was based on the width of the Boolean set. The slope of the sides of each generalized bell function, controlled by the parameter *b* as described in Chapter 2, was set to two, for all the sets. During the ANFIS training stage in Step 4, this parameter would be varied for best results. Figure 6.3 shows the fuzzy sets of the Estimated Size profiling attribute.

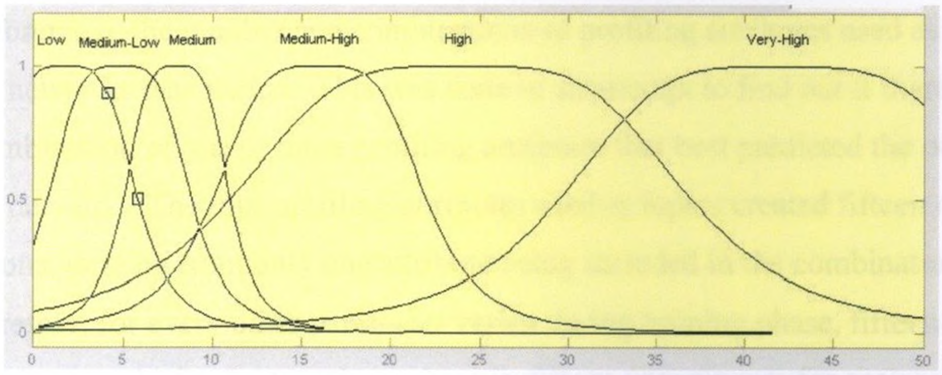


Figure 6.3 – The fuzzy sets of the Estimated Size profiling attribute.

For the zero-order ANFIS implementation in Step 4 of the NFEM preparation process, the output sets needed to be represented by a single constant. As the output of the ANFIS, each Effort set shown in Table 6.15 was to be represented by the median of its data points, also shown in Table 6.15.

6. 3 Step 3: Neural Network Training

Once all the profiling attributes and the classification attribute were separated into sets, and the data was transformed into Boolean form, step 3 of the NFEM preparation process could be applied to the Industrial Partner's dataset. The third step consisted of training neural networks with the Boolean data and extracting rules from the network that achieved the most accurate classification results. These rules could then be implemented into the ANFIS.

6.3.1 Parameters Varied

In order to determine the network architecture, and the combination of inputs that would most accurately describe the factors that affect software implementation task effort, several different parameters were varied during the training of the networks.

6.3.1.1 Varying the Inputs

First and foremost, the number and combinations of profiling attributes used as input to the neural networks was varied. This was done in an attempt to find out if there was a certain combination of one or more profiling attributes that best predicted the output of the neural network. The four profiling attributes used as inputs created fifteen different combinations, ranging from only one attribute being included in the combination to all four. Therefore, for every other parameter varied during training phase, fifteen different input combinations had to be tested, resulting in fifteen experiments for each test case.

6.3.1.2 Varying the Number of Hidden Nodes

The number of hidden nodes was the second testing parameter varied to determine if there was a specific network architecture under which the neural network was best trained. To determine the ideal number of hidden nodes, three different neural network architectures were used: One with fifteen nodes in the hidden layer, another one with thirty-five, and a final one with fifty nodes. These values were determined through trial and error. Once again, in each test case, for each input combination, all three network architectures were tested.

6.3.1.3 Varying the Number of Training Epochs

The final parameter varied was the number of epochs used to train the neural networks. This would test to see if increasing the number of epochs would compensate for the low volume of data. In Test Case 1 the number of epochs was limited to 1000 and in Test Case 2, the same experiments were repeated but the number of epochs used to train the neural networks was increased to 5000.

6.3.2 Testing Setup

The output of each neural network was Effort, in hours. Therefore, each neural network trained in these two test cases consisted of six output neurons in the third layer, each one corresponding to one of the sets of the Effort attribute determined in Section 6.2.3.

To automate each test case, an algorithm was designed and implemented to automatically create a three-layer backpropagation neural network, train it with the data presented, count the number of correct classifications and write it in a comma separated values (CSV) file. A correct classification was considered to be a point that is classified into the right output set. The mean squared error (MSE) of the last epoch of each trained neural network was also recorded within the CSV file, in order to keep track of the how successful the network training was. For each parameter variance, thirty networks were trained and an average was obtained to determine the overall affect of that particular parameter's value change. Due to the scarcity of the number of data points, all 313 data points were used in training the neural networks.

The results of each test case are displayed in a bar graph that contains the experiment number and the names of the input attributes on the x-axis. The names of the inputs are shortened to only the first letter of the attributes' names as shown in Table 6.17. The y-axis shows the average percentage of correctly classified data points. Three bars are shown for each experiment, one representing the networks with fifteen hidden neurons in the middle layer, another representing the networks with thirty-five hidden neurons, and a final one representing the networks with fifty hidden neurons. The last 3 bars represent the overall averages of all the experiments for each test case. For each test case, a second chart is also shown comparing the average MSE of the last training epochs for each experiment.

Experiment No.	Inputs	Short Name Used on the X-axis of Figures 6.4 to 6.11
1	Complexity	1 - C
2	Estimated Size	2 - E
3	Functionality	3 - F
4	Language	4 - L
5	Complexity & Estimated Size	5 - C&E
6	Complexity & Functionality	6 - C&F
7	Complexity & Language	7 - C&L
8	Estimated Size & Functionality	8 - E&F
9	Estimated Size & Language	9 - E&L
10	Functionality & Language	10 - F&L
11	Complexity & Estimated Size & Functionality	11 - C&E&F
12	Complexity & Estimated Size & Language	12 - C&E&L
13	Complexity & Functionality & Language	13 - C&F&L
14	Estimated Size & Functionality & Language	14 - E&F&L
15	Complexity & Estimated Size & Functionality & Language	15 - C&E&F&L

Table 6.17 – The input abbreviations used on the figures displaying the results.

6.3.2 Test Results

6.3.2.1 Test Case 1: Training with 1000 Epochs

Figure 6.4 shows the results of testing all fifteen input combinations and all three network architectures when each of the networks was trained with 1000 epochs. Overall, the trained networks performed poorly. The average percentage of correctly classified data points varies from 5.29%, produced in Experiment 4, when the network architecture with fifty hidden nodes was used, to 33.42%, produced when Estimated Size was the input to the network architecture with fifty hidden nodes. The networks with only Estimated Size as input perform significantly better than the rest of the input combinations regardless of the network architecture used. However, even at the highest data classification performance of 33.42%, the results are too low to allow accurate rules to be extracted from the network.

In terms of network architecture, it can be observed from Figure 6.4, and specifically, the last three bars showing the overall experiments' average, that changing the number of nodes in the hidden layer from thirty-five to fifty does not significantly improve the performance of the networks. The average classification accuracy increase is only one percent between the two different network architectures. In fact, in experiments 3, 4, 7, 10, and 13, the network classification accuracy actually decreases. On the other hand, the network architecture of thirty-five hidden nodes does outperform the network with fifteen hidden nodes, especially in experiments 2, 5, 9, and 12 where the classification accuracy nearly doubles.

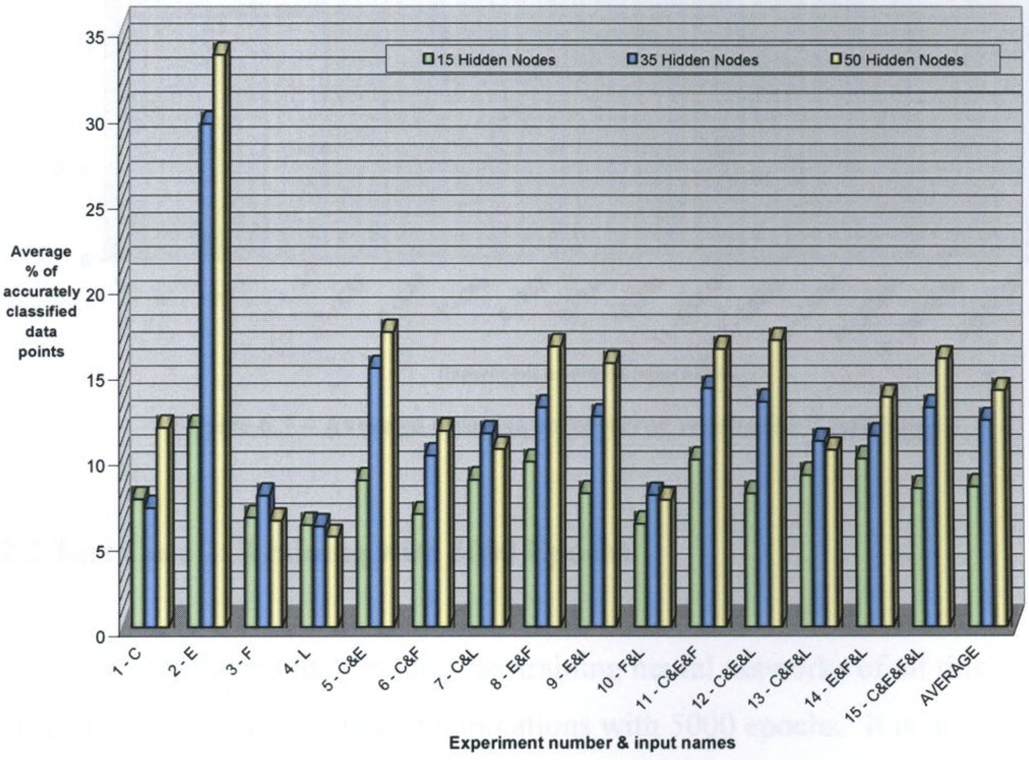


Figure 6.4 – Average classification accuracy results for Test Case 1.

The poor classification performance of the experiments is not very surprising given the high average MSE values shown in Figure 6.5. The highest average mean squared error observable is 0.34, produced when Language is the input and there are fifteen hidden nodes in the network architecture. This indicates that the average error for a given output of the network with the Language profiling attribute as input was 0.583, or 58.3% (obtained by taking the square root of the MSE). In Test Case 2, the number of training epochs was increased to 5000, to see if the MSE would decrease, resulting in an increase in the classification accuracy.

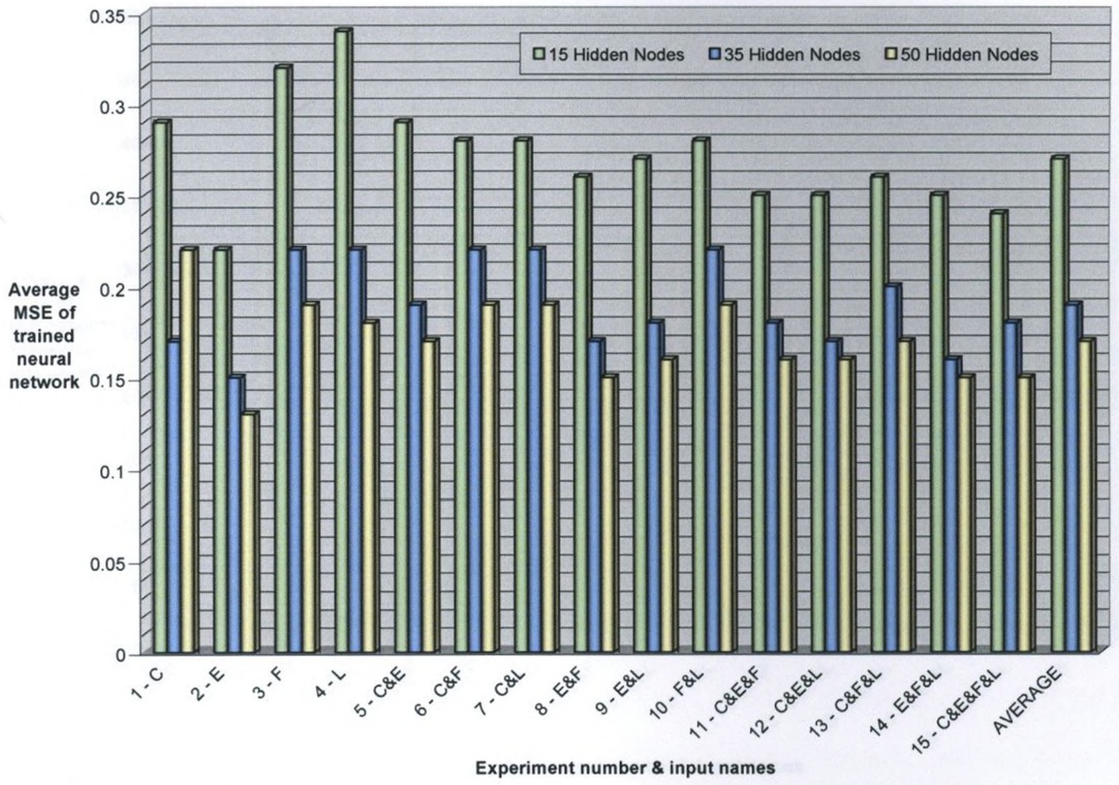


Figure 6.5 – Average mean squared error results for Test Case 1.

6.3.2.2 Test Case 2: Training with 5000 Epochs

Figure 6.6 shows the results produced by training neural networks of all three architectures and all fifteen input combinations with 5000 epochs. It is interesting to note that although the average MSE of all experiments decreased significantly (Figure 6.7), the average percentage of accurately classified data points also decreased for some of the experiments. The classification performance of the network with fifty hidden nodes and Estimated Size as the only input increased to 48%, and in turn, all networks that include the profiling attribute of Estimated Size as an input show improved classification performance. For example, in Figure 6.6, experiments 2, 5, 8, 9, 11, 12, 14, and 15 all show higher classification accuracy than the same experiments in Figure 6.4, regardless of the network architecture. However, most networks that do not include Estimated Size as an input decreased in classification accuracy. The only exceptions to this were experiments 6 and 13 when the network architecture with fifty hidden nodes was used.

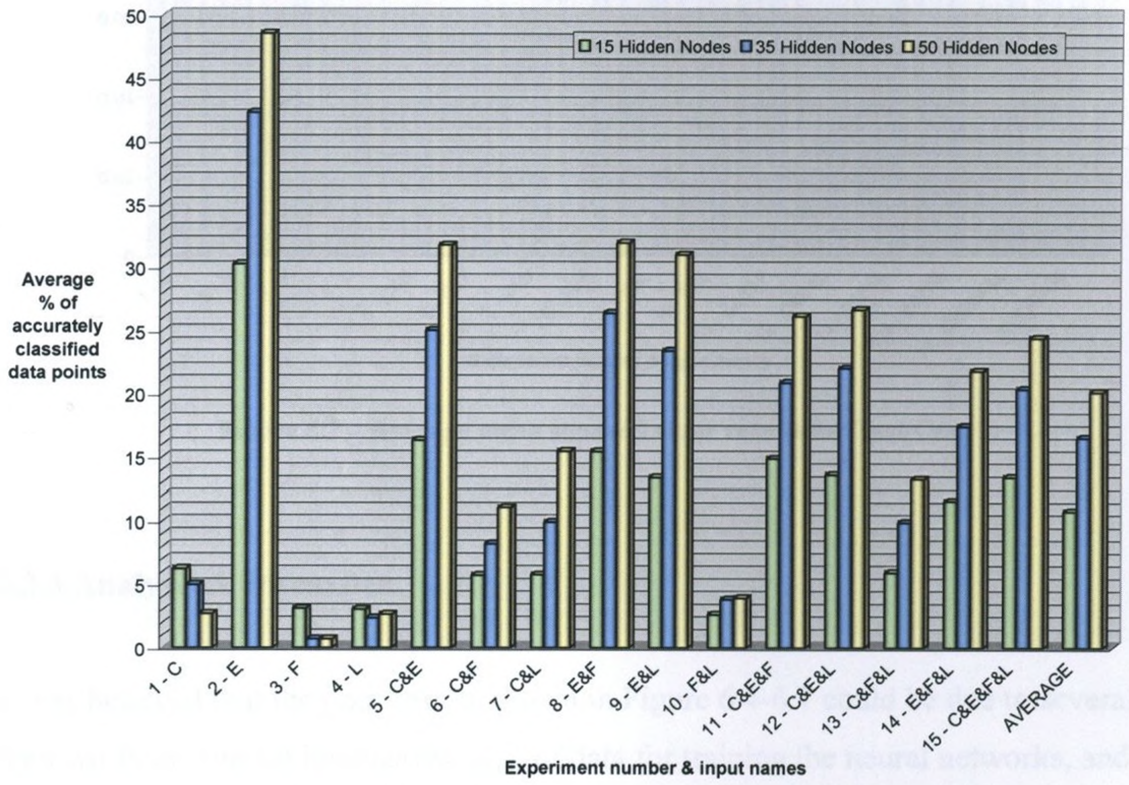


Figure 6.6 – Average classification accuracy results for Test Case 2

The lowest average MSE value observed in Figure 6.7 is 0.081, when only Estimated Size is used as an input to the network architecture with fifty hidden nodes. This indicates that the average error for a given output of that network architecture was 28.5%. Once again, this is a significantly high error rate that would not allow accurate rules to be extracted from the network. Further tests were conducted to test if increasing the number of training epochs to 10,000 would significantly increase the classification accuracy of any of the experiments, however at best, only a 3% accuracy increase was observed.

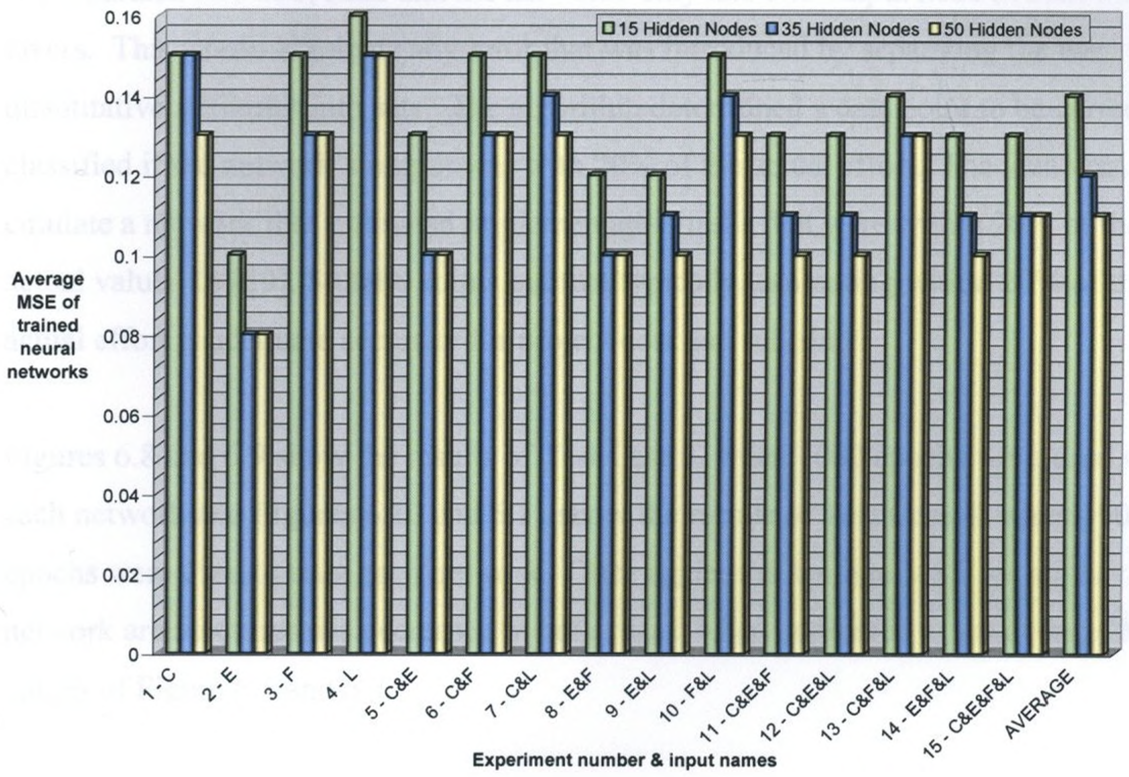


Figure 6.7 – Average mean squared error results for Test Case 2.

6.3.3 Analysis & Discussion

It was believed that the poor results shown in Figure 6.4-6.7 could be due to several reasons: Poor data set boundaries, lack of data for training the neural networks, and low data quality. Following, each of these hypotheses were examined and further tests were conducted when possible, to eliminate them and continue with the NFEM process.

The first hypothesis stated that the poor results were caused by the poorly located boundaries of the sets into which the attributes Effort and Estimated Size were separated. That is to say, the clustering of the sets of these two attributes, which was based on the equal-data criterion was inadequate. In order to test this hypothesis, two more test cases were developed. In these two cases, the Estimated Size attribute was not separated into sets, but rather, it was just normalized and entered as a value between zero and one into the neural networks. In addition, the output attribute of Effort, was also normalized and not separated into sets, such that the networks only had one output node in their third layers. This would eliminate any error that was introduced by separating the two quantitative attributes into sets. The algorithm determined a data point to be correctly classified if the network's output was with 20% of the actual effort. The idea was, to emulate a network that estimated implementation tasks that were within 20% of the actual value. In [50], Stutzke points out that typically, estimating within 20% of the actual effort is adequate accuracy for project cost and schedule.

Figures 6.8 and 6.9 show the results of Test Case 3, when 1000 epochs were used to train each network and Figures 6.10 and 6.11 show the results of Test Case 4, when 5000 epochs were used to train each network. Once again, the average MSE for all the network architectures has decreased in both cases, when compared to the average MSE values of Figure 6.5 and 6.7.

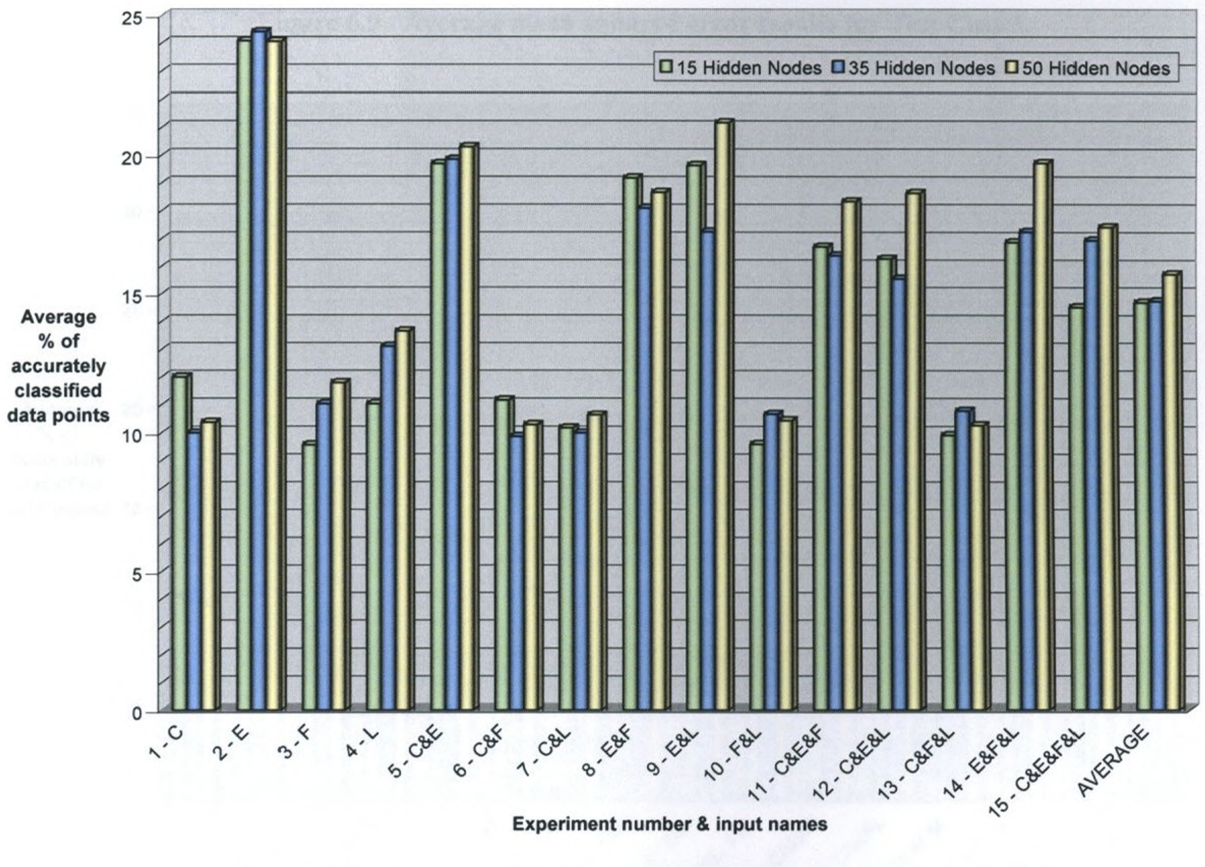


Figure 6.8 – Average classification accuracy results for Test Case 3.

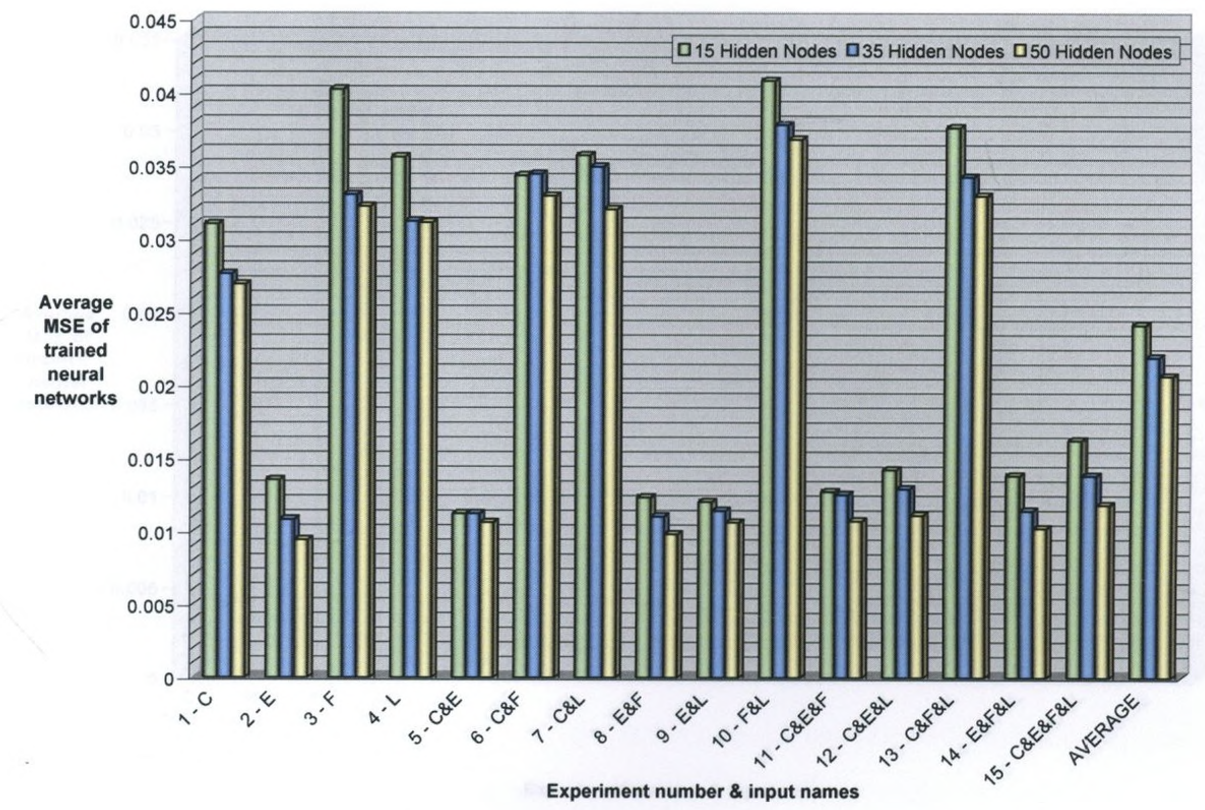


Figure 6.9– Average mean squared error results for Test Case 3.

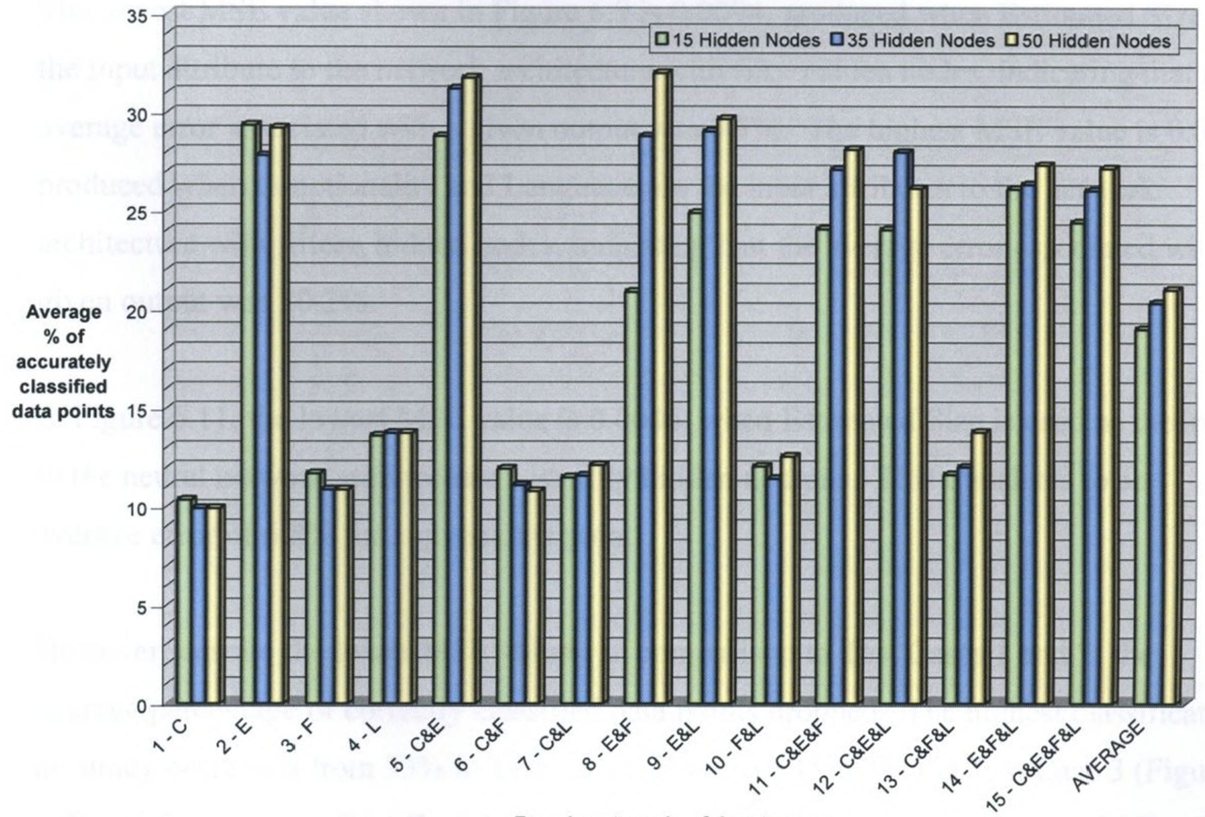


Figure 6.10 – Average classification accuracy results for Test Case 4.

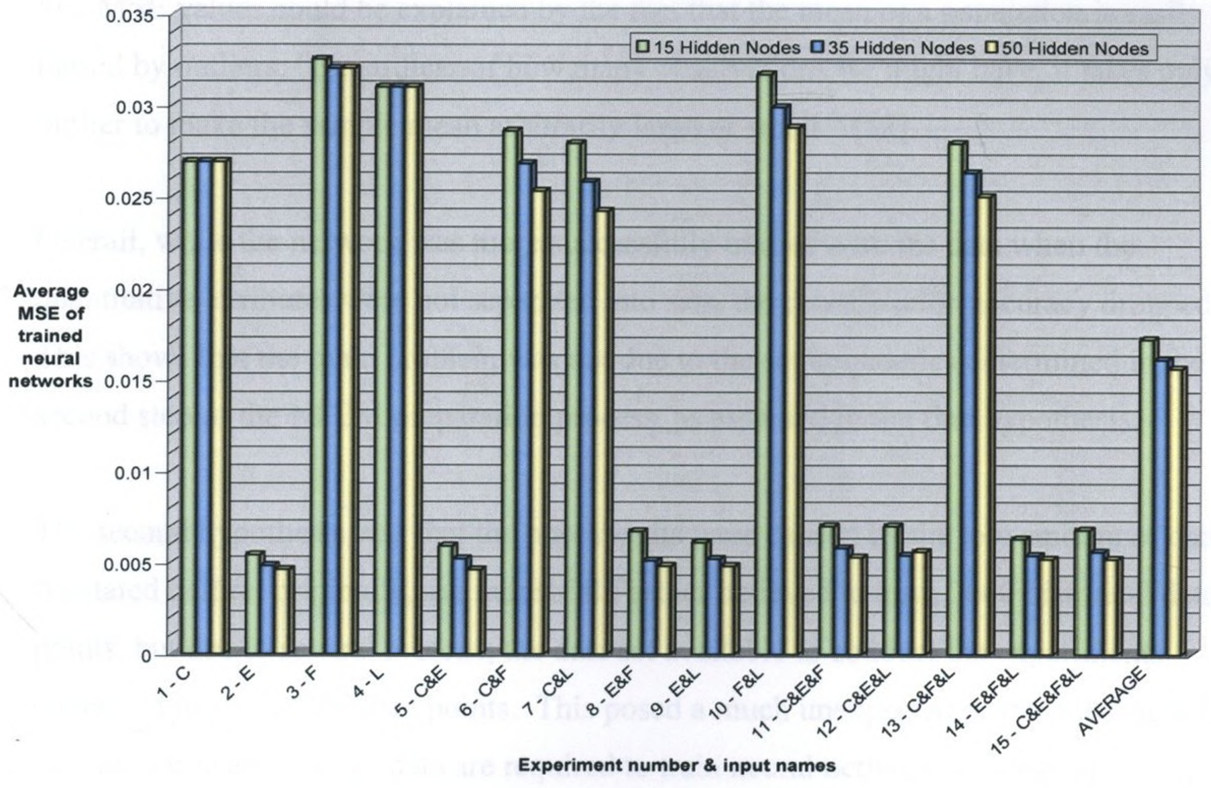


Figure 6.11 – Average mean squared error results for Test Case 4.

The lowest MSE value shown in Figure 6.9 is 0.0094, produced when Estimated Size is the input attribute to the network architecture with fifty hidden nodes, indicating that the average error associated with a given output was 9.7%. The highest MSE value is 0.041, produced when Functionality and Language are the input attributes to the network architecture with fifteen hidden nodes, indicating that the average error associated with a given output was 20.2%.

In Figure 6.11, the lowest MSE value is 0.0046, when Estimated Size is used as the input to the neural network architecture with fifty hidden neurons. This translates to an average error of 6.8% for a given data point.

However, despite the lower MSE values, in comparison to Test Cases 1 and 2, the average percentage of correctly classified data points dropped. The highest classification accuracy decreased from 33% in Test Case 1 (Figure 6.4) to 24% in Test Case 3 (Figure 6.8) and from 48% in Test Case 2 (Figure 6.6) to 32% in Test Case 4 (Figure 6.10). The low MSE values could be explained by the fact that the mean of a population is easily biased by outliers: “Regardless of how many observations we might have, it takes only 1 outlier to make the sample mean arbitrarily large or small.” [53].

Overall, while the network was more successfully trained with the data when the quantitative attributes were not separated into sets, the classification accuracy dropped. This shows that the main problem was not due to the set boundaries determined in the second step of the NFEM preparation process, as assumed in the first hypothesis.

The second hypothesis was that the poor results were caused by the low amount of data. As stated before, originally, the Industrial Partner believed to have 2000 historical data points, but due to several reasons, the data set available to conduct the experiments contained just over 300 data points. This posed a much unexpected problem because in general, large amounts of data are required to train neural networks successfully. Since there were no more data points available however, as all networks were trained with all 313 data points, there was no practical way of determining if more data would yield better results. But theoretically, the findings of [6], which were summarized in Section 4.1.2, could be used to test this second hypothesis. Assuming a fraction of error of 0.125 within the training data, for every weighted connection of a multilayer neural network, there need to be approximately ten training data points. In the best case scenario, in Test Cases 1 through 4, the networks with the lowest amount of weighed connection weights consisted of three nodes in the first layer, corresponding to the sets of the one profiling attribute used as an input, fifteen hidden nodes in the middle layer, and six output nodes in the third layer. Therefore there were forty-five weighted connections from the first layer to the middle one (obtained by multiplying the number of nodes in the first and second layer) and ninety weighted connections in the second layer (obtained by multiplying the number of nodes in the middle and third layer), totaling 135 weighted connections in the neural network. As a result, theoretically, a minimum of 1350 data points would be required to train the neural network. This proves that insufficient data was the cause of the poor network classification results.

The third hypothesis, which stated that the quality of the training data may be low, leading to poor classification results cannot be proved true or false, due to the lack of data: If sufficient data was available to train the neural networks, and the classification accuracy continued to be low, then the quality of data could be further investigated.

Overall, there was insufficient data to successfully complete the third step of the Neuro-Fuzzy Estimation Model. However, given that only 313 data points were available, when at minimum, 1350 data points were needed, the results turned out to be promising. When only a fifth of the required data points were used to train the neural networks, 48% classification accuracy was achieved. Thus, although the Industrial Partner's data was insufficient to fully implement the third step of the NFEM preparation process, the results are promising.

6.4 Step 4: Rule Extraction and ANFIS Implementation

Step 4 of the NFEM preparation process could not be implemented due to the long term process involved in collecting sufficient data. Therefore, time constraints did not allow for the full validation of the Neuro-Fuzzy Process. Nevertheless, Chapter 4 describes in detail how the full implementation of the NFEM would be completed once sufficient data was available.

CHAPTER VII
CONCLUSION

7.1 Summary of Contributions

The ability to produce accurate software development effort estimates is essential to the software industry. Based on them project scope is determined, quality standards are set in place, and cost and schedule constraints are defined. Yet, software development effort estimates are often plagued by omissions, uncertainty, and bias [50]. Existing estimation models continue to frequently produce inaccurate estimates, instigating research studies that attempt to determine the properties they lack. After decades of such studies and practical experience, a number of deficiencies have been found that hinder existing estimation models from producing accurate estimates. This thesis focused on developing a new effort estimation model that amends those deficiencies by incorporating within it the following characteristics:

1. The ability to handle diverse process and product variables.
2. The ability to incorporate empirical evidence and expert judgment.
3. The ability to determine genuine cause and effect relationships.
4. The ability to handle uncertainty.
5. The ability to handle incomplete information.

The Neuro-Fuzzy Estimation Model presented in this thesis was designed with all of these characteristics in mind. This section discusses how the NFEM accomplishes each of the above characteristics as well as other characteristics that greatly benefit the process of estimation.

7.1.1 Input Customizability

As a universal software estimation model, the NFEM offers full customizability of the inputs it uses. Each organization or team within an organization implementing the NFEM is able to choose the attributes that are believed to most influence the development effort. This is very important because depending on the development environment and type of system being built, the factors that affect development effort can greatly vary. In addition, the definitions of the scale values for each attribute are determined independently by each organization. This is an advantage that no existing effort estimation model has as it significantly expands the flexibility of the NFEM. After all, what is considered "High Reliability" by a team that develops safety critical systems is different from what a team that develops video games considers it to be. And while the NFEM does not require that Expert Effort Estimate be a mandatory profiling attribute, the inclusion of it is strongly recommended and its implementation is facilitated by the NFEM's ability to integrate quantitative attributes. Finally, the output of the NFEM is effort, but how effort is measured is left to the discretion of each organization implementing the NFEM. This NFEM characteristic offers additional freedom of customizability.

7.1.2 Incorporating Neural Networks

The use of multilayer feedforward neural networks in the preparation process of the NFEM accomplishes several feats. First and foremost, feedforward multilayer neural networks have been proven to have the ability to model any input-output relationship [19]. Therefore, training a neural network with the effort estimation data of a given environment, allows the relationships between the profiling attributes and the classification attribute to be modeled. In addition, it automatically calibrates the model with the given environment's data. Studies have shown that calibrating an effort estimation model with a given environment's data significantly increases the model's estimation accuracy [23], [27], [34]. Furthermore, using neural networks to model input-output relationships automatically filters out profiling attributes that do not have a significant effect on effort. By extracting rules from the most successfully trained neural network, those attributes that were not used as inputs to that particular network are eliminated. Finally, multilayer neural networks are robust to incomplete information, rendering the NFEM to be robust to incomplete information as well, specifically because the NFEM is implemented as a neuro-fuzzy system.

7.1.3 Incorporating Fuzzy Logic

The fuzzy logic side of the NFEM also delivers several benefits. Estimations, by their very nature contain a degree of uncertainty within them. Therefore, applying a mathematical algorithm such as fuzzy inference, that is intended to take into account the inherent uncertainty of the data allows for more accurate estimations. In addition to dealing with uncertainty, the incorporation of fuzzy logic within the NFEM allows the integration of qualitative and quantitative attributes. The nature of fuzzy logic also allows the NFEM to be able to deal with imprecise information due to the subjectivity present in the values of the profiling attributes. By nature, most factors that strongly affect software development effort are subjective. Therefore, instead of developing another model that tries to avoid subjectivity of the metrics (an impossible task), the NFEM uses fuzzy logic to take into account the imprecision present in the data. Finally, the extraction and implementation of rules into the ANFIS system allows for the Neuro-Fuzzy Estimation Model to be transparent and allows for the rules to be validated by experts. This avoids the negative aspects of using neural networks to model the input-output relationships, a structure that is often considered to be a "black-box" [44].

7.1.4 Implementing the NFEM as an ANFIS

The implementation of the NFEM as an adaptive neuro-fuzzy inference system allows for the fuzzy membership functions to be further fine-tuned to a given organization's environment. Furthermore, it facilitates a continuous process of improvement. For example, a large number of profiling attributes can be selected to be measured, but initially the NFEM can be implemented with only a fraction of those, so that a very large amount of data is not necessary. Over time however, as more data is collected, the NFEM can be re-implemented to include more profiling attributes. Also, even if new attributes are not introduced into the NFEM, as the volume of historical data increases, the NFEM can be retrained with the new available data for some improvement, or the preparation process steps can be repeated, starting from Step 2, for a more extensive improvement of the system.

7.1.5 Estimating Effort at the Task Level

The final advantage that the NFEM has over existing estimation models is that it facilitates automatic data collection at a lower level. While theoretically, the NFEM can be used as a model to estimate high-level software development effort, in practice this is not feasible due to the long period of time that it would take to collect the required training data. Therefore, the NFEM was developed as an estimation model for collecting data at the software development task level. One advantage to this is that it makes the collection of calibration data mandatory, and as discussed previously in this section, model calibration is necessary if the estimation model is to be expected to perform with some accuracy. In addition, the smaller the task size, the easier it is to accurately profile it and estimate it using expert estimation. Generally speaking, the larger a software development task is, the more difficult it is to estimate it [15], [28], [45], [50] because as the size of the task increases, so does the interdependency among various elements of the software [45]. As a result, estimating becomes harder because there is more uncertainty in the estimate. Therefore, the Neuro-Fuzzy Estimation System is able to avoid a degree of data uncertainty and imprecision, simply by dealing with more granular software development tasks.

7.2 Future Work

The Neuro-Fuzzy Estimation Model presented in this thesis provides a great foundation and much potential for producing accurate estimates. However, due to time constraints, the scope of the thesis had to be limited, hindering further research into some of the areas that could be perceived as weaknesses within the NFEM. These potential weaknesses provide future directions for this research.

First and foremost, Section 3.7 discussed the great effect the task implementer's capability has on effort, regardless of the development environment or product being developed. Yet, in the real world, evaluating individuals and making such evaluations usable in a public tool presents several major problems, from confidentiality breach to workplace morale decrease. Further research must be conducted in how to overcome these obstacles and allow the inclusion of the profiling attribute implementer capability.

In step 2 of the NFEM preparation process, the boundaries of quantitative attributes were determined based on the equal-amount of data criterion. Perhaps a better approach would be to determine the boundaries by using self-organized maps [30] or clustering algorithms [5]. These algorithms would be able to find the natural boundaries that occur within the dataset.

In step 3 of the Neuro-Fuzzy Estimation Model, different neural network architectures had to be tested to reveal the most suitable number of hidden nodes in the middle layer. A better alternative would be to use algorithms such as those proposed in [17], [35], or [36] to determine the number of hidden nodes in the middle layer. Additionally, the dataset used in the neural network training step should be separated into a training set and a testing set. This would ensure that the neural network is not too closely modeled to the training set resulting in poor performance when new inputs are entered.

For the fourth and final step of the NFEM preparation process, further research could be conducted in a couple of areas. Firstly, only decompositional rule-extraction algorithms were considered during the development of this thesis. Consequently, pedagogical and eclectic rule extraction algorithms could be explored in the future to see if they yield better results. The second area that could be explored in the future is to allow certain profiling attributes to have a greater effect on the effort estimate than others, in hopes that more accurate estimates are produced. This could be accomplished by assigning greater weights to rules whose antecedents include attributes that are believed to have a more significant effect on effort.

Finally, the Neuro-Fuzzy Estimation Model is designed to estimate software development effort estimation at a stage when some details are known about the project. This is beneficial for the later stages of estimation, when the requirement specifications or design phases have been completed. But in the early phases of project conception, many of the development tasks that the project will consist of are not known. Thus, a formal process of associating the task-level estimates generated by the NFEM with high-level project estimates produced at the commencement of a project would yield many benefits. Further research must be conducted to bridge these two different levels of estimation.

In conclusion, the Neuro-Fuzzy Estimation Model proposed in this thesis provides a successful foundation for overcoming many of the obstacles faced by existing software development effort estimation models.

REFERENCES

[1] A Brief History of Software Engineering. *History of Computing*. December 5, 2004. Obtained April 17, 2007. <http://www.comphist.org/computing_history/new_page_13.htm>

[2] Andrews, R. Diederich, J., Tickle, A., "Survey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks." *Knowledge-Based Systems*, Vol. 8, pp.373-389, 1995.

[3] Armour, P. "The Business of Software: Beware of Counting LOC." *Communications of ACM*, Vol. 47, No. 3, pp. 21-24, 2004.

[4] Armour, P. "The Business of Software: Ten Unmyths of Project Estimation." *Communications of the ACM*, Vol. 45, No. 11, pp. 15-18, 2002.

[5] Baraldi, A., Blona, P. "A Survey of Fuzzy Clustering Algorithms for Pattern Recognition." *IEEE Transactions on System, Man, Cybernetics*. Vol. 29, No. 6, pp. 786-801, 1999

[6] Baum, E. B., Haussler, E. "What Size Net Gives Valid Generalization?" *Neural Computation 1*. pp. 151-160, 1989

[7] Boehm, B. *Software Engineering Economics*. Englewood Cliffs, New Jersey: Prentice Hall, 1981.

[8] Boehm, B., Bradford, C., Horowitz, E., Madachy, R. Shelby, R. and C. Westland. "The COCOMO 2.0 Software Cost Estimation Model." *International Society of Parametric Analysts,* May 1995.

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

[10] Boetticher, G. D. "Using Machine Learning to Predict Project Effort: Empirical Case Studies in Data Starved Domains." *In Proceedings of Model-Based Requirements Workshop*, 2001.

[11] Fenton, N., Martin, N. "Software Metrics: Roadmap." *ACM 2000: Future of Software Engineering*, Limerick, Ireland, 2000.

[12] Finnie, G. R., Wittig, G. E. "AI Tools for Software Development Effort Estimation." *1996 International Conference on Software Engineering: Education and Practice*, pp. 346-352, 1996.

[13] Fuller, R. *Introduction to Neuro-Fuzzy Systems.* Hidelberg, New York: Physica-Verlag, 2000.

[14] Fu, L. "Rule Generation from Neural Networks." *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 24, pp. 1114-1124, 1994.

[15] Glass, R. *Facts and Fallacies of Software Engineering.* Boston, MA: Pearson Education, 2003.

[16] Gray, A. R., MacDonell, S.G. "A comparison of techniques for developing predictive models of software metrics." *Information and Software Technology*, pp. 425-437, 1997.

[17] Hansen, L.K., Salamon, P. "Neural Network Ensembles." *IEEE Transactions on Pattern Analysis and Machine Intelligence.* Vol. 12, No. 10,.1990.

[18] Hayashi, Y., Imura, A. "Fuzzy Neural Expert System with Automated Extraction of Fuzzy If-Then Rules from a Trained Neural Network", *Proceedings of First International Symposium on Uncertainty Modeling and Analysis*, pp. 489-494, 1990.

[19] Homik, K., Stinchcombe, M., White, H., "Multilayer feedforward networks are universal approximators." *Neural Networks Archive*, Vol. 2, pp. 359-366, 1989.

[20] Huang, X. "A Neuro-Fuzzy Model for Cost Estimation." *Master's Thesis.* The University of Western Ontario, 2003.

[21] Jang, J. –S. R. "ANFIS: Adaptive Network-based Fuzzy Inference Systems." *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 23, No. 3, pp. 665-685, May 1993.

[22] Jang, J. –S. R., Sun, C. –T., Mizutani, E. *Neuro-Fuzzy And Soft Computing: A Computational Approach to Learning and Machine Intelligence.* Upper Saddle River, NJ: Prentice Hall, 1997.

[23] Jeffery, R., Scott, L. "Has Twenty-Five Years of Empirical Software Engineering Made a Difference?" *In Proceedings of the 9th Asia-Pacific Software Engineering Conference,* pp. 539-546, 2002.

[24] Jørgensen, M. "A Review of Studies on Expert Estimation on Software Development Effort." *Journal of Systems and Software,* Vol. 70, No. 1, pp. 37-60, 2004.

[25] Kasabov, N., "Learning Fuzzy Rules and Approximate Reasoning in Fuzzy Neural Networks and Hybrid Systems." *Fuzzy Sets and Systems*, Vol. 82, pp. 135-149, 1996.

[26] Kasabov, N., "Learning Fuzzy Rules through Neural Networks." *Proceedings of the 1st New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems*, 1993.

[27] Kemerer, C. F. "An Empirical Validation of Software Cost Estimation Models." *Communications of the ACM*. Vol. 30, No. 5, pp. 416-429, 1987.

[28] Keyes, J. *Software Engineering Handbook.* Boca Raton, FL: CRC Press LLC, 2003

[29] Kira, K., Rendell, L. "The Feature Selection Problem: Traditional Methods and a New Algorithm." *In Proceedings of the National Conference on Artificial Intelligence*, pp. 129-134, 1992.

[30] Kohonen, T. *Self-organization and Associative Memory.* New York, New York, USA: Springer-Verlag, 1989

[31] Krishnan, R., Sivakumar, G., Bhattacharya, P. "A search technique for rule extraction from trained neural networks." *Patterns of Recognition Letters*, Elsevier Sciences, 1999.

[32] Laranjeria, L. "Software Size Estimation of Object Oriented Systems." *IEEE Transactions on Software Engineering*, Vol. 17, No. 5, 1990.

[33] MATLAB 6.5 Help Documentation. The MathWorks Incorporated.

[34] Maxwell, K., Wassenhove, L.V., Dutta, S. "Performance Evaluation of General and Company Specific Models in Software Development Effort Estimation." *Management Science*. Vol. 45, No. 6, pp.787-803, 1999.

[35] Moody, J. "Note on Generalization, Regularization, and Architecture Selection in Nonlinear Learning Systems." *Proceeding of Neural Networks for Signal Processing*, 1991.

[36] Murata, N., Yoshizawa, S., Amari, S. "Network Information Criterion – Determining the Number of Hidden Units for an Artificial Neural Network Model." *IEEE Transactions on Neural Networks*, Vol.5, No.6, pp. 865-872, 1994

[37] National Aeronautics and Space Administration (NASA). Parametric Cost Estimating Handbook. Obtained March 03, 2006. <http://www1.jsc.nasa.gov/bu2/PCEHHTML/pceh.htm>

[38] Nauck, D., Klawonn, F., Kruse., R. *Foundations of Neuro-Fuzzy Systems.* West Sussex, England: John Wiley & Sons, 1997.

[39] Nauck, D. Kruse, R. "NEFCLASS – A Neuro-Fuzzy Approach for the Classification of Data." *Proceedings of the 1995 ACM Symposium on Applied Computing*, pp. 461-465, 1992.

[40] Nauck, D., Nauck, U., Kruse, R. "Generating Classification Rules with the Neuro-Fuzzy System NEFCLASS." *Proceedings of the 1996 North American Fuzzy Information Processing Society Conference*, pp. 466-470, 1996.

[41] Negnevitsky, M. *Artificial Intelligence: A Guide to Intelligent Systems.* Harlow, England: Addison-Wesley, 2005.

[42] Newman, D.J. & Hettich, S. & Blake, C.L. & Merz, C.J. *UCI Repository of Machine Learning Databases.* Irvine, CA: University of California, Department of Information and Computer Science. 1998. <http://www.ics.uci.edu/~mlearn/MLRepository.html>

[43] Pedrycz, W. "Computational Intelligence as an Emerging Paradigm of Software Engineering." *Proceedings of the 14th international Conference on Software Engineering and Knowledge Engineering.* Vol. 27, pp. 7-14, 2002.

[44] Picton, P. *Neural Networks.* New York, New York: Palgrave, 2000.

[45] Pressman, R. *Software Engineering: A Practitioner's Approach*, 6th Edition. McGraw-Hill, USA: 2005.

[46] Putnam, L. H. "A General Empirical Solution to the Macro Software Sizing and Estimating Problem." *IEEE Transactions in Software Engineering.* pp. 345-361, 1978.

[47] Putnam, L. H. "Larry Putnam's Career in Software Engineering." *Quantitative Software Management Website.* Obtained. March 15, 2006. <http://www.qsm.com/provisions.html>

[48] Sethi, I. K., Yoo, J.H. "Symbolic mapping of neurons in feedforward neural networks." *Pattern Recognition Letters,* 17, pp. 1035-1046, 1996.

[49] Shaw, M. "Practical Software Engineering." *University of Calgary: Practical Software Engineering Course Website* (15. Jan. 1996). Obtained March 03, 2006. <http://ksi.cpsc.ucalgary.ca/courses/451-96/mildred/451/CostEffort.html>

[50] Stutzke, R. *Estimating Software-Intensive Systems.* Addison Wesley, Upper Saddle River, NJ: 2005

[51] Towell, G., Shavlik, J. "Extracting Refined Rules from Knowledge-Based Neural Networks." *Machine Learning*, Vol. 13, pp. 71-101, 1993.

[52] Towell, G., Shavlik, J., Noordewier, M.O. "Refinement of Approximately Correct Domain Theories by Knowledge-Based Neural Networks." *Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 861-866, 1990.

[53] Wilcox, R. R. *Fundamentals of Modern Statistical Methods.* Springer-Verlag, New York, 2001.

[54] Zadeh, L. A. "Fuzzy Sets." *Information and Control*, pp. 338-353, 1965.

[55] Zadeh, L. A. "Soft Computing and Fuzzy Logic." *IEEE Transactions, Systems, Man and Cybernetics*, pp. 48-56, 1994.

Appendix A

Attribute Skill Level of Implementer

Name	Skill Level of Implementer
Definition	The degree to which the skill level of the task implementer influences the effort estimate.
Rationale	A particularly complex feature may require someone with a lot of skill and /or experience. The person who is assigned with implementing the task should influence the size of the estimate.
Implementation	Each of the sub-attributes of this attribute will be rated on a scale of Low, Medium-Low, Medium, Medium-High, and High, which will correspond to a scale of 1 to 5. Then, all the sub-attributes will be averaged to obtain a single value for the Skill Level of Implementer attribute. To avoid inaccurate data due to personal bias, it would be best if each team member is rated by his/her technical lead and the values be saved in a separate table and updated periodically by the technical leads. In addition, a generic profile should be available for cases when the task implementer is not known but a task estimate is needed.

Sub-Attributes for Attribute Skill Level of Implementer

Analyst Capability			
Definition	The ability to investigate new strategies or defects, as well as the overall quality, reliability, and robustness of work items previously completed by the task implementer.		
Scale Values' Definitions	Low	The implementer produces low quality work.	⇧
	Medium	In general, the quality of the work produced by the implementer is adequate.	
	High	The implementer produces high quality work.	
Learning Ability			
Definition	The task implementer's ability to learn new concepts and acquire new skills quickly.		
Scale Values' Definitions	Low	The implementer takes much more time than what is considered reasonable before being able to apply new concepts/skills.	⇧
	Medium	In general, the implementer takes, what is considered, a reasonable amount of time before being able to apply new concepts/skills.	

	High	The implementer takes a minimal amount of time before being able to apply new concepts/skills.	
Efficiency			
Definition	The ability to complete a task accurately and on time (i.e. without over-analyzing the problem and the possible solutions).		
Scale Values' Definitions	Low	The implementer takes much longer than what is considered reasonable to complete most tasks.	⇧
	Medium	The implementer completes most tasks, in what is considered a reasonable amount of time.	
	High	The implementer completes most tasks ahead of time.	
Teamwork			
Definition	The ability to communicate in a timely manner with other team members and management and the ability to co-operate in terms of choosing the best possible solution for the task, while still adhering to time and quality constraints.		
Scale Values' Definitions	Low	The implementer needs improvement in his/her communication and cooperation skills and does not adhere to team decisions.	⇧
	Medium	The implementer has sufficient communication and cooperation skills and usually adheres to team decisions.	
	High	The implementer has excellent communication and cooperation skills and always adheres to team decisions.	

Attribute Familiarity with Technology

Name	Familiarity with Technology, Technology
Definition	The degree to which the implementer's familiarity with the technology, used to complete the task, influences the estimate.
Rationale	Uncertainties in employing new technology (e.g. .NET framework.) or integration with a 3rd party tool (e.g. Crystal Reports) could require the implementer to spend time installing and/or becoming familiar with them. Any tasks using new technology should be increased in effort to account for this overhead.
Implementation	Each of the sub-attributes of this attribute will be rated on a scale of Low, Medium-Low, Medium, Medium-High, and High, which will correspond to a scale of 1 to 5. The sub-attributes will then be averaged to obtain a single value for the Familiarity with Technology attribute. Note: At the time of estimation, the implementer may not be identified. Since this attribute is implementer dependent and

	the implementer's familiarity will not be known, the estimator must use his/her best judgement with respect to the overall team's familiarity with the technology.

Sub-Attributes for Attribute Familiarity with Technology

Familiarity with Documentation			
Definition	The degree of the task implementer's knowledge /understanding of the technology's documentation: Has the implementer skimmed the documentation or thoroughly read it. Documentation includes: help files, user guides, online tutorials, and books dedicated to the technology.		
Scale Values' Definitions	Low	The implementer has never looked at any of the documentation before.	
	Medium	The implementer has read and understands about half of the help files, user guides and other resources concerning the technology.	⇧
	High	The implementer has read and understands almost all the content of the help files, user guides and other resources concerning the technology.	
Usage of Technology			
Definition	How well the task implementer feels that he/she knows how to implement solutions using the technology. This is a measure of the level of comfort he/she has in using the technology.		
Scale Values' Definitions	Low	The implementer is not comfortable with using the technology.	
	Medium	The implementer is sufficiently comfortable with using the technology.	⇧
	High	The implementer is very comfortable with using the technology.	

Attribute Familiarity with Programming Language

Name	Familiarity with Programming Language, Familiarity with Language, Language
Definition	The degree to which the implementer's familiarity with the software language, to be employed when completing the task, influences the estimate.
Rationale	Some learning time may be included in using a new programming language.
Implementation	Each of the sub-attributes of this attribute will be rated on a scale of Low, Medium-Low, Medium, Medium-High, and High, which will correspond to a scale of 1 to 5. The sub-

	attributes will then be averaged to obtain a single value for the Familiarity with Language attribute. Note: At the time of estimation, the implementer may not be identified. Since this attribute is implementer dependent and the implementer's familiarity will not be known, the estimator must use his/her best judgement with respect to the overall team's familiarity with the language.

Sub-Attributes for Attribute Familiarity with Language

Familiarity with Documentation			
Definition	The degree of the task implementer's knowledge/understanding of the language's support documentation. The documentation includes: help files, user guides, online knowledge databases, and books dedicated to the use of the language.		
Scale Values' Definitions	Low	The implementer has never read any of the help files or user guides.	⇧
	Medium	The implementer has read and understands about half of the topics in the help files and user guides.	
	High	The implementer has read and understands almost all of the content of the help files and user guides.	
Usage of Language			
Definition	How well the task implementer feels that he/she knows how to implement solutions using the chosen language. This is a measure of the level of comfort he/she has in developing solutions with the language.		
Scale Values' Definitions	Low	The implementer is not comfortable developing with the language.	⇧
	Medium	The implementer is sufficiently comfortable developing with the language.	
	High	The implementer is very comfortable developing with the language.	

Attribute User Interface

Name	User Interface, UI
Definition	The degree to which the level of complexity of the user interface influences the estimate.
Rationale	As the level of UI complexity increases, the amount of time required to incorporate input validation and to manage resource/error strings increases.
Implementation	Each of the sub-attributes of this attribute will be rated on a scale of Low, Medium-Low, Medium, Medium-High, and High, which will correspond to a scale of 1 to 5. The sub-

	attributes will then be averaged to obtain a single value for the User Interface attribute. If all sub-attributes are set to NA, then User Interface attribute will not affect the effort estimation at all. However if any one of the sub-attributes is set anything other than NA, then the rest of the sub-attributes should also be set to something other than NA.

Sub-Attributes for Attribute User Interface

Amount of UI Controls			
Definition	A linguistic approximation of the amount of user interface controls needed by the functionality. User interface controls include: text boxes, list boxes, radio buttons, command buttons, menus, combo boxes, etc.		
Scale Values' Definitions	Low	Up to 5	⇩
	Medium	5 to 15	
	High	More than 15	
	NA	Not applicable	
Required Level of Validation			
Definition	A qualitative measure of the amount of input validation required by the user interface of the task's functionality.		
Scale Values' Definitions	Low	The UI controls are self-validating in nature such as radio buttons and combo boxes or the user input is just for commenting purposes and will not cause program errors.	⇩
	Medium	Some validation is required as invalid input can cause program failures.	
	High	The user input data is critical to the application and invalid input will definitely cause program failures.	
	NA	Not applicable.	
Underlying Architecture Complexity			
Definition	The overall complexity of the underlying architecture. For example, a simple registry access function will likely have a low architectural complexity whereas functionality providing the ability to insert 3rd party ActiveX controls would likely have a high architectural complexity.		
Scale Values' Definitions	Low	The underlying functionality is very simple.	⇩
	Medium	The underlying functionality is of average complexity.	
	High	The underlying architecture is very complex.	
	NA	Not applicable.	

Attribute Complexity

Name	Complexity
Definition	The degree to which the complexity of the task influences the estimate.
Rationale	Adding a piece of data (e.g. a new attribute to an existing object class) would require more effort in a complex system than in a simple one.
Implementation	Each of the sub-attributes of this attribute will be rated on a scale of Low, Medium-Low, Medium, Medium-High, and High, which will correspond to a scale of 1 to 5. The sub-attributes will then be averaged to obtain a single value for the Complexity attribute.

Sub-Attributes for Attribute Complexity

Difficulty of Definition			
Definition	The degree of difficulty involved in defining the solution such as the algorithmic complexity of the solution in terms of computational complexity (e.g. nested loops, analysis of differential equations), time computational complexity (e.g. real-time systems), space computational complexity (e.g. distributed database coordination), and information-based complexity (e.g. simple arrays in main memory vs. highly coupled dynamic relational and object structures).		
Scale Values' Definitions	Low	The solution is very easy to define. The implementation of its functionality and constraints is straightforward and easy to express.	⇩
	Medium	The solution is somewhat easy to define. The implementation of its functionality and constraints is of average difficulty.	
	High	The solution is very difficult to define. The implementation of its functionality and constraints is not straightforward.	
Interdependence with other Features			
Definition	The amount of other functions/features the current task impacts and/or the amount of functions/features the current task is impacted by.		
Scale Values' Definitions	Low	The task is mostly independent of other functionality.	⇩
	Medium	Some of the other functionality is dependent on how this task is implemented and/or the implementation of this task is dependent on how some of the other functionality is implemented.	
	High	A lot of other functionality is dependent on this task and/or this task is dependent on a lot of other functionality.	

Attribute Familiarity with Functionality

Name	Familiarity with Functionality, Functionality
Definition	The degree to which the implementer's familiarity with the functionality influences the estimate.
Rationale	If the task is different from anything that has been implemented by the development team in the past, then some extra effort will be required for research and learning.
Implementation	Each of the sub-attributes of this attribute will be rated on a scale of Low, Medium-Low, Medium, Medium-High, and High, which will correspond to a scale of 1 to 5. The sub-attributes will then be averaged to obtain a single value for the Familiarity with Functionality attribute. Note: At the time of estimation, the implementer may not be identified. Since this attribute is implementer dependent and the implementer's familiarity will not be known, the estimator must use his/her best judgement with respect to the overall team's familiarity with the functionality.

Sub-Attributes for Attribute Familiarity with Functionality

Similarity			
Definition	The degree to which the current task resembles something that the implementer has previously implemented.		
Scale Values' Definitions	Low	The implementer has never before implemented similar functionality.	⇧
	Medium	The implementer has previously implemented functionality that is somewhat similar.	
	High	The implementer has implemented very similar functionality.	
Product Knowledge			
Definition	How familiar the implementer is with the application/product being developed. This will give a measure of how well the implementer understands how the component/functionality will affect the existing components/functionality.		
Scale Values' Definitions	Low	The implementer has never before worked on the product and knows very little about it.	
	Medium	The implementer knows some of the key things about how the product is built but not all of the details.	⇧
	High	The implementer is very familiar with the product and how it is implemented.	
Component Knowledge			
Definition	How familiar the implementer is with the component the current task involves.		

Scale Values' Definitions	Low	The implementer has very little knowledge about the component involved in the current task.	⇧
	Medium	The implementer has implemented some parts of the component involved in the current task.	
	High	The implementer is very familiar with the component involved in the current task, and has been one of the main people involved in implementing it.	

Attribute Familiarity with Domain

Name	Familiarity with Domain
Definition	The degree to which familiarity with the application domain influences the estimate.
Rationale	If the task implementer has no understanding of the product objectives or the customer's/domain's goals, more effort would be required for the task, because some research and learning will be necessary before the implementation of the task can begin. If the task implementer has recently switched from the data domain (i.e. working with databases) to the HMI (Human Machine Interaction) domain (i.e. working with user interfaces), more effort will be required for the task implementer to complete UI tasks due to the lack of experience with the domain.
Implementation	Each of the sub-attributes of this attribute will be rated on a scale of Low, Medium-Low, Medium, Medium-High, and High, which will correspond to a scale of 1 to 5. The sub-attributes will then be averaged to obtain a single value for the Familiarity with Domain attribute. Note: At the time of estimation, the implementer may not be identified. Since this attribute is implementer dependent and the implementer's familiarity will not be known, the estimator must use his/her best judgement with respect to the overall team's familiarity with the domain.

Sub-Attributes for Attribute Familiarity with Domain

Product Domain Familiarity	
Definition	The level of familiarity the implementer has with the application domain (i.e. functions within the industry that the product will be used).

Scale Values' Definitions	Low	The implementer has very little knowledge about the product's domain.	
	Medium	The implementer has general knowledge about the product's domain (e.g. key goals, key problems, organizational structure, etc.) but does not know details.	⇧
	High	The implementer knows a lot of details about the product's domain.	
Software Domain Familiarity			
Definition	The level of familiarity the implementer has with the task's related software domain (e.g. database, GUI, server, web, etc.).		
Scale Values' Definitions	Low	This is one of the first times the implementer is completing tasks in this particular software domain and does not feel comfortable with it.	⇧
	Medium	The implementer has previously worked on the task's software domain and feels somewhat comfortable with it.	
	High	The implementer is very comfortable with the task's software domain.	

Attribute Estimated Size

Name	Estimated Size
Definition	The degree to which the size of the task influences the estimate.
Rationale	The task implementer generally has an idea of the amount of time the task should take. This is based on past experience and is generally somewhat accurate.
Implementation	Size is equivalent to the number of hours estimated to complete the task. By selecting one of the PERT, Function Point or Expert techniques a value previously estimated will be used.